PreProceedings of the 9th International Workshop on Design, Specification, and Verification of Interactive Systems
DSV-IS’2002
(Rostock, Germany, 12-14 June 2002)

Bricks & Blocks:
Towards Effective User Interface Patterns and Components

Peter Forbrig, Quentin Limbourg, Bodo Urban,
Jean Vanderdonckt (Eds.)

University of Rostock (Germany) - Université catholique de Louvain (Belgium)
Thematic Focus
Design, Specification, and Verification of Interactive Systems (DSV-IS) is the annual meeting of the human-computer interaction community interested in all aspects of the design, the specification, and the verification of interactive systems. It serves as the principal international forum for reporting outstanding research, development, and industrial experience in this area. The 9th DSV-IS workshop will provide a forum for the exchange of ideas on diverse approaches to the design and implementation of interactive systems. The particular focus of this year's event is on models and their role in supporting the design and development of interactive systems for ubiquitous computing.

Ubiquity and Usability
Usability of interactive systems for ubiquitous computing is a key factor of future software developments. The challenge in user interface development is no longer to implement a single (stationary) user interface from specification but rather to allow user interfaces for a wide variety of devices (e.g., mobile devices, cellular phones, PDAs, pocket PC, handheld PC, ... ) and multimodal input channels. In addition, deploying a same user interface across a wide variety of devices, appliances, and platforms raises the question of how to factor out common interaction components and patterns across the different instances of the user interface, while preserving (some) consistency. Rather than reproducing the same parts on different platforms, common bricks and blocks might be used. Some platforms are well suited for certain interactive tasks, while others are not at all able to support them. This edition is dedicated to all forms of patterns involved in human-computer interaction: cross platform, design, globalization, mobility, ubiquity, usability,....
Scope

Papers of these PreProceedings address theory, design, development, evaluation of ideas, tools, techniques, methodologies in (but not limited to) the following areas:

- Affective, emotional and game based UI
- Agent-based UI
- Component-based UI development
- Development support tools and techniques
- Domain specific model-based approaches
- Formal description of user related properties
- Formal methods in interactive systems development
- Front-end interfaces to multimedia, hypermedia, knowledge-based, personalized information, simulation systems
- Methods, metrics and tools for computer-aided evaluation of UI
- Mobile and ubiquitous interaction
- Model-based and task-based approaches to UI design
- Model-based Interface Development Environments (MB-IDEs)
- Models for Novel Interaction Techniques
- Models of context of use, specific properties of mobile and ubiquitous usage contexts
- Models for context- and situation-aware interactive assistance
- Novel Techniques for Interacting with Formal Models
- Patterns in HCI: cross-platform, design, globalization, mobility, usability,…
- Pattern languages
- User interface architectures
- UI management systems (UIMSs)
- UI for virtual/augmented/mixed reality
- UML and HCI
DSV-IS’2002 Review Committee

Ghassan Al-Qaimari, Royal Melbourne Institute of Technology, Australia
Gilbert Cockton, University of Sunderland, United Kingdom
Joëlle Coutaz, University Joseph Fourier, CLIPS-IMAG Lab., Grenoble, France
Alan Dix, Lancaster University, United Kingdom
David Duce, Oxford Brookes University, United Kingdom
Ulrich Eisenecker, University of Applied Sciences Kaiserslautern, Germany
Giorgio Faconti, CNUCE-CNR, Italy
Peter Forbrig, University of Rostock, Germany
Nick Graham, Queen's University, Canada
Richard Griffiths, University of Brighton, United Kingdom
Phil Gray, University of Glasgow, Scotland
Jan Gulliksen, University of Uppsala, Sweden
Michael Harrison, University of York, United Kingdom
Michael Herczeg, University of Lübeck, Germany
Michael Holloway, NASA Langley Research Center, USA
Chris Johnson, University of Glasgow, Scotland
Peter Johnson, University of Bath, United Kingdom
Thomas Kirste, Fraunhofer Institute for Computer Graphics Rostock, Germany
Panos Markopoulos, Technische Universität Eindhoven, The Netherlands
Christian Märtin, Augsburg University of Applied Sciences, Germany
Miguel Gea Megías, University of Granada, Spain
Erik G. Nilsson, SINTEF, Norway
Dan Olsen, Brigham Young University, USA
Philippe Palanque, LIISIS-IRIT, Université Paul Sabatier (Toulouse 3), France
Fabio Paternò, CNUCE-CNR, Italy
Ángel Puerta, RedWhale Software Corp., USA
Matthias Rauterberg, Technical University of Eindhoven, The Netherlands
Kevin Schneider, University of Saskatchewan, Canada
Ahmad Seffah, Concordia University Montréal, Canada
Pavel Slavík, Czech Technical University in Prague, Czech Republic
Chris Stary, University of Linz, Austria
Gerd Szwilus, University of Paderborn, Germany
Michael Tauber, University of Paderborn, Germany
Manfred Tsheligi, Center for Usability Research and Engineering, Austria
Bodo Urban, Fraunhofer Institute for Computer Graphics Rostock, Germany
Martijn van Welie, Satama Interactive Amsterdam, The Netherlands
Jean Vanderdonckt, Université catholique de Louvain, Belgium
Sponsors
We gratefully acknowledge the support of the following organisations
# Table of contents

## Invited paper

**Survival of the Fittest: Evolution of User Interfaces in Business Software**

*Udo Arend, SAP, Germany*

## Long and short papers

**Towards Design Guidelines for Constructing a Formal Specification**

*J.A. van der Poll and P. Kotzé, University of South Africa, South Africa*

**From a Formal User Model to Design Rules**

*Paul Curzon, Middlesex University, UK*

*Ann Blandford, University College London, UK*

**A Coloured Petri Net Formalisation for a UML-Based Notation Applied to Cooperative System Modelling**

*José Luis Garrido ad Miguel Gea, University of Granada, Spain*

**Adaptive User Interface for Mobile Devices**

*N. Mitrović and E. Mena, University of Zaragoza, Spain*

**Migratable User Interface Descriptions in Component-Based Development**

*Kris Luyten, Chris Vandervelpen, and Karin Coninx Limburgs Universitair Centrum, Belgium*

**Task Modelling in Multiple Contexts of Use**

*Nathalie Souchon, Quentin Limbourg, and Jean Vanderdonckt, Université catholique de Louvain, Belgium*

**Notational Support for the Design of Augmented Reality Systems**

*Emmanuel Dubois, University of Glasgow, UK*

*Paulo Pinheiro da Silva, University of Manchester, UK*

*Philip Gray, University of Glasgow, UK*

**Tool-Supported Interpreter-Based User Interface Architecture for Ubiquitous Computing**

*Lars Braubach, Alexander Pokahr, Daniel Moldt, Andreas Bartelt and Winfried Lamersdorf, University of Hamburg, Germany*

**VILARS. A New Dialogue Model Applying Augmented Reality**

*Toni Granollers, J. Lorés, G. Raimat, and E. Junyent, E. Tartera, Universidad de Lleida, Spain*

**Combining Compound Conceptual User Interface Components with Modelling Patterns**

*Erik Nilsson, SINTEF Telecom and Informatics, Norway*
Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model.................................................................160
Ahmed Seffah, Concordia University, Canada
Peter Forbrig, University of Rostock, Germany

Foundations of Cognitive Support: Toward Abstract Patterns of Usefulness........175
Andrew Wetenstein, University of Victoria, Canada

User Interface Design Patterns for Interactive Modeling in Demography
and Biostatistics........................................................................................................190
Sergiy Boyko, Max Planck Institute for Demographic Research and
University of Rostock,
Peter Forbrig, and Anatoli Yashin, University of Rostock, Germany

User Interface Conceptual Patterns.................................................................201
Pedro Molina, Santiago Meliá, CARE Technologies S.A.,
Oscar Pastor, Technical University of Valencia, Spain

Monitoring Human Faces from Multi-View Image Sequences .......................215
Nailja Luth, Fraunhofer Institute for Computer Graphics, Germany

Improving Mouse Navigation-A Walk through the “Hilly Screen Landscape”.....228
David Ahlström, Martin Hitz, and Gerhard Leitner,
University of Klagenfurt, Austria

Designing User Interaction for Face Tracking Applications ............................239
David Chatting and Jeremy Thorne, Btexact Technologies, UK

AWUSA-A Tool for Automated Website Usability Analysis............................251
Thomas Tiedtke, Christian Martin, and Norbert Gerth,
Augsburg University of Applied Sciences, Germany

Models for Task-Object-Based Web Site Management....................................267
Gerd Szwillus, Universität Paderborn,
Birgit Bomsdorf, FernUniversität Hagen, Germany

Dynamic and Adaptive E-Commerce Architecture Based
on Agent Technologies.......................................................................................282
Ana Gil, Francisco Garcia, University of Salamanca, Spain
Zahia Guessoum, LIP6, France

Using Declarative Constraints to Specify the Data Model
of a Multi-User Application..................................................................................296
Boris van Schooten, University of Twente, The Netherlands
Design of Handheld Interactive Support

Carmine Ciavarella and Fabio Paternò, CNUCE-CNR, Italy

Performance Evaluation as a Tool for Quantitative Assessment of Complexity of Interactive Systems

Xavier Lacaze, Philippe Palanque, David Navarre, and Rémi Bastide, University of Toulouse III, France

Blending Descriptive and Numeric Analysis in Human Reliability Design

Shamus Smith and Michael Harrison, The University of York, UK

KWARESMI-Knowledge-Based Web Automated Evaluation with Reconfigurable Guidelines Optimization

Abdo Beirekdar, Université catholique de Louvain, Monique Noirhomme-Fraiture, Facultés Universitaires Notre-Dame de la Paix, Jean Vanderdonckt, Université catholique de Louvain, Belgium

Net-Fly And Contextcontrol: Two Examples of Game Based Interfaces

Holger Diener, Mathias Mainka, and Hagen Schumacher, Fraunhofer Institute for Computer Graphics, Germany

Towards A Ubiquitous Semantics of Interaction: Phenomenology, Scenarios and Traces

Alan Dix, Lancaster University, UK

Architecture Considerations for Interoperable Multi-Modal Assistant Systems

Thomas Heider and Thomas Kirste, Fraunhofer Institute for Computer Graphics, Germany

Posters

Model-Based Approach to Control over Concurrency in Interactive CSCW Applications

Yannick Bouillon, Atlantide, France

Fabrice Wendling, Université de Rennes I, France

A Priori Usability Testing of Ubiquitous Computing Applications Designs

Chris Stary, University of Linz, Austria
Survival of the Fittest: Evolution of User Interfaces in Business Software

Udo Arend
Usability Engineering Center, SAP AG, Neurottstr. 16, 69190 Walldorf, Germany
udo.arend@sap.com

Extended Abstract

Examining the user interfaces of current business applications reveals a quite common, not to say a little bit boring, look and feel for the user. “Cool” widgets, like table lenses, magical lenses, hyperbolic browsers, starfield displays cannot be found on the user interfaces of business software. This observation also holds true for the use of advanced visualization techniques, such as those using 3D. It holds true even for the use of advanced input/output technology like speech input, speech output, or for the use of agent technology. Instead, the controls found on the user interfaces of business applications are typical of the “traditional” GUI, familiar since Apple’s Lisa.

This observation leads us to the following hypotheses, which have been adapted from Darwin (1859) and transferred to the domain of user interfaces:

(H1) User interfaces of business software are not immutable. They emerged out of an uninterrupted succession of (software) generations from the origin of interactive systems up to today’s existing user interfaces.

(H2) User interfaces of business applications are all different from each other. Among these interfaces exists wide variation for each attribute.

(H3) Every user interface in a business application is subject to natural selection. Only those that best fit best the customers’ and users’ needs have an opportunity to survive and be reproduced.

We will prove our hypotheses by following the history of SAP software’s user interfaces, starting in 1980 and ending in 2002 with pattern-based user interfaces.

While the 1980’s were the decade of character-based user interfaces, the 1990’s drove the GUI to a certain state of perfection. Perhaps this first decade of the 21st century could be the decade of “pattern-based user interfaces”, which means that the user interface is comprised of highly standardized components which are optimized for certain types of tasks.

The reader may ask, “But weren’t there complete paradigm shifts when the GUI replaced the character based user interface, or, at least, a paradigm shift when browser technology replaced the traditional GUI?”

We will show that, from our point of view, the changes happened pretty much evolutionarily, at least for business software. This means that the elements of the user interface have been improved and enriched step-by-step, and not that all elements disappeared suddenly and totally new elements and structures appeared.

This evolutionary process in user interface development has been influenced and enriched by new areas of software evolution that have arisen successively.
One area could be called the “rise of role-based user interfaces”. Having one user interface for every user of an application, or even for every customer, very quickly shows its limitations. Customers want to adapt the user interface to their own business processes and to special user groups. This happens on an individual basis. However, users also have specific roles in their organizations such as sales person, buyer or accountant. Therefore, we identified those roles and the related tasks, and, by providing a piece of technology (SAP Session Manager, SAP Easy Access), we were able to bundle applications for specific business roles.

Another area could be characterized by the “discovery of aesthetics for user interface design”. Everybody knows those old, gray GUIs with their standard controls. From the Web we learned that it is possible to visually design the elements so that they result in a pleasing user interface. Hence, we decided to visually design our user interface elements and to rearrange them. This happened during the Enjoy project, during which a major redesign of many applications took place.

The “user interfaces of the Web” also were a big challenge for UI design in another way. Because the Web is originally a publishing medium that deals mainly with the layout of text and pictures and with access to information, it questioned traditional UI design. But, as can be seen from the current generation of Web-based business applications, the influence of the Web paradigm meant only an enrichment of the traditional GUI. Thus, we learned to better deal with pictures and unstructured text, and to combine small application modules (called iViews, and including text and collaboration-based modules) and complete applications in a business portal that enhances access to data.

In the 1990’s, we learned from our numerous business applications that there must be “underlying patterns” which constitute an application. An attempt in the mid 1990’s to identify the underlying business patterns failed. Because we had no underlying pattern theory, we identified patterns on the level of visible components such as print and table dialogs. We cancelled this project because we did not feel that it could help us in application development.

We started a new pattern identification project in September 2001. This time we began with a theoretical perspective that was independent from concrete applications. Our definition of at least three levels of patterns proved to be very fruitful. We distinguish (1) the elementary action pattern level (e.g. edit a field) from (2) the component task pattern level (e.g. select an object; edit an address) from (3) the business process pattern level (e.g. create a business partner). These pattern levels are initially independent of mapping to a specific user interface. From reanalyzing more than fifty “single-screen” applications and some typical Web-based processes we were able to identify many “specific” component tasks. We discovered that the concepts of a business process fit an object tree in a very standardized manner. From these findings, we were able to extract the “generic” component tasks at level 2. Out of these component tasks we could develop the “floor plan” (see Christopher Alexander, “The Timeless Way of Building”) for each business process. The floor plan consists of the specific component tasks and their spatial relationships. We learned that a few sets of floor plans were sufficient to cover most of our business processes. These findings, and others, helped us to relate the floor plan of a business process to user interface components in a highly standardized way.
From our experience and observations we draw several conclusions from the evolution of user interfaces:

H1’) The environment in which user interfaces are built changes all the time. Thus, it is necessary to continually adapt existing products according to new paradigms to stay competitive (e.g. GUI, roles, Web, patterns).

H3a’) Selection is driven by internal problems and entropy. We have shown that the complexity of a user interface steadily increases. At a certain point, a complete redesign of the user interface is necessary to ensure that the software remains usable.

H3b’) Selection is also driven by external developments. For example, the rise of the Web totally changed the way we think about applications. We have to deal with unstructured information, graphic design and collaborative applications.

The process of building usable software is becoming more and more mature. User-centered design, with all its facets, shows this. We currently have pattern-based user interface design that aims to build and reuse highly standardized components.

What will come in the future? In fact, we do not know. We are strongly driven by the market and our customers. New software generations with better user interfaces will follow – due to natural selection.
Towards Design Guidelines for Constructing a Formal Specification

John A. van der Poll\textsuperscript{1} and Paula Kotzé\textsuperscript{2}

Department of Computer Science and Information Systems
University of South Africa
\textsuperscript{1}vdpolja@unisa.ac.za \textsuperscript{2}kotzep@unisa.ac.za

Abstract. Accepted software engineering design principles are well established, but design principles for constructing a formal specification have been relatively rare. In this paper we examine a number of formal specifications written in \textit{Z} as well as some design principles from software engineering and areas of general design, upon which we propose a preliminary set of guidelines for the construction of a formal specification. These guidelines aim to incorporate general design principles as well as those often used in the final software, already at the specification phase. We illustrate how one of these guidelines, namely the use of primitives, allows a specifier to discharge an important proof obligation arising from a formal specification, where otherwise a proof is not easily arrived at.

1 Introduction

Software engineering design principles have been around for a couple of decades, dating back to the early work on \textit{structured design} (e.g. \cite{1}, \cite{2}, and \cite{3}). In many instances the object-oriented design methodology replaced the earlier structured techniques as the preferred method of design. Numerous notational representations for object-oriented design were introduced, e.g. Coad et al. \cite{4}, Rumbaugh et al. \cite{5}, Booch \cite{6}, and Shlaer et al. \cite{7}.

Guidelines for object-oriented software design have been proposed by Yourdon \cite{8}, Love \cite{9}, and Chidamber et al. \cite{10} while general measures for high-level design were formulated by Briand et al. \cite{11}. Human Computer Interaction (HCI) design principles are presented in Dix et al. \cite{12} (amongst others), while principles for general design are developed in Norman \cite{13}.

Design principles for drawing up a formal specification have been relatively sparse. Gravell \cite{14} proposes a number of principles for constructing a specification written in \textit{Z} (e.g. \cite{15}), mainly to make such specifications more readable.

1.1 Why Design Guidelines?

The question of why a set of guidelines for constructing a formal specification might be useful boils down to asking what the specification is to be used for.

A formal specification is often used as the starting point of a subsequent refinement phase Morgan \cite{16}, and a well-designed specification could possibly be
more easily refined to code than an ad hoc specification. One of the advantages of using a formal notation during the specification phase is that the specifier can reason about the specification formally. Reasoning about the properties of a specification is an important activity early in the process of constructing a reliable program (e.g. Woodcock et al. [17]), e.g. we can show that certain undesirable properties are absent from the specification.

A specification constructed according to established design criteria could be more comprehensible to users, easier to enhance and thereby facilitate long-term maintenance, especially if principles embodied in the final software are already used at the specification phase.

Although the Established Strategy (e.g. Barden et al. [18]) goes a long way in presenting a Z specification in an intelligible way, established software design principles (e.g. cohesion [8]) are not officially part of this strategy; neither is the use of certain general design principles (e.g. make things visible to the user [13]).

In this paper we propose a number of guidelines to support the construction of a formal specification. These guidelines aim to:

1. Incorporate some established software engineering design principles normally present in the final product already at the specification phase.
2. Apply a number of HCI and general design principles in the construction of a formal specification.
3. Facilitate the initial stages of a subsequent refinement process.
4. Structure the specification so as to facilitate the process of proof.

1.2 Structure of this Paper

Section 2 introduces a preliminary set of guidelines aimed at enhancing the utility of a specification. The definition of these guidelines is based on an analysis of a number of Z specifications, some software engineering and general design principles, and considering some initial refinement stages. In Sect. 3 we build a small specification using some of the principles proposed below. We conclude with ideas about future work and applicability of our guidelines.

2 Design Guidelines

2.1 Format of a Precondition

Consider the state FS of a UNIX-like filing system. A detailed discussion of FS appears in Morgan & Sufkin [19, p. 45 - 78]. We will use this state definition and the 'open file' operation below to illustrate our first three guidelines.

In FS, FID (a set of file identifiers), CID (a set of channel identifiers), and SYL (a set of syllables) are all basic types; FILE = seq BYTE, for BYTE = 0..255; NAME = seq SYL (i.e. a sequence of syllables); and CHAN is given by:

```
CHAN
  fid : FID
  posn : N
```
An operation to open a file is given by:

\[
\Delta \text{FS}
\]

\[
\text{name? : NAME; cid! : CID}
\]

\[
\text{fid, fid' : FID}
\]

\[
\text{report! : REPORT}
\]

\[
\begin{align*}
(n & \text{name?} \in \text{dom nstore} \land \text{cid!} \notin \text{dom cstore} \land \\
& \text{fid} = \text{fid'} = \text{nstore(name?)} \land \\
& (\exists \text{CHAN'} \bullet \text{posn'} = 0 \land \text{fid'} = \text{fid} \land \\
& \text{cstore}' = \text{cstore} \oplus \{\text{cid!} \mapsto \theta\text{CHAN'}\}) \land \\
& \text{nstore}' = \text{nstore} \land \text{report!} = \text{OK} )) \\
\end{align*}
\]

\[
\begin{align*}
\lor (n & \text{name?} \notin \text{dom nstore} \land \theta\text{FS'} = \theta\text{FS} \land \text{report!} = \text{NoSuchName}) \\
\lor ( & \text{dom cstore} = \text{CID} \land \theta\text{FS'} = \theta\text{FS} \land \text{report!} = \text{NoFreeCids})
\end{align*}
\]

The precondition \( \text{cid!} \notin \text{dom cstore} \) is the negation of \( \text{dom cstore} = \text{CID} \) and vice versa in the sense that the system attempts to obtain a new unused channel identifier (i.e. \( \text{cid!} \)), and if successful, the condition \( \text{cid!} \notin \text{dom cstore} \) holds, otherwise there are no free identifiers left and \( \text{dom cstore} = \text{CID} \) prevails.

The partial preconditions of operation \text{open} \ are:

\[
\begin{align*}
(n & \text{name?} \in \text{dom nstore} \land \text{cid!} \notin \text{dom cstore}) \quad (1) \\
(n & \text{name?} \notin \text{dom nstore}) \quad (2) \\
( & \text{dom cstore} = \text{CID}) \quad (3)
\end{align*}
\]

The total precondition of \text{open}, namely \((1) \lor (2) \lor (3)\) is a tautology but not a partition since two of these conditions overlap. Often in a specification this non-determinism is deliberate because it allows implementers flexibility. However, if preconditions overlap in this way then a sequence of automatic refinement steps could generate the following (incorrect) structure:

\[
\text{if } \text{precondition1} \text{ then } S1 \\
\text{elseif } \text{precondition2} \text{ then } S2 \\
\text{elseif } \text{precondition3} \text{ then } S3 \\
\text{endif}
\]
The semantics of (4) requires the preconditions to be pairwise disjoint, leading to our first design guideline:

\textit{Guideline \#1}: Ensure that the precondition to a total operation is a partition whenever non-determinism is not required.

2.2 Communication with the User

There is a further aspect to the above discussion as far as feedback to the user is concerned: Consider the scenario where there is no free channel available (i.e. \( CID = \text{dom } c\text{store} \)) and the input file name, \( name? \), is incorrect (i.e. \( name? \notin \text{dom } n\text{store} \)). Suppose further that owing to the above non-determinism the message \( 'NoFreeCids' \) is displayed, informing the user to wait for a channel to become available before proceeding. However, once a channel is released by another process, the user can try to reconnect again, just to be faced with the message \( 'NoSuchName' \). One could argue that this message should have been displayed together with the message about the channel, so that the user could have fixed the problem in the meantime, instead of having had to just wait for a free channel. This leads to our second design guideline:

\textit{Guideline \#2}: Maximise communication with the user of the system.

The above guideline agrees with the following principle proposed by Donald Norman [13, p.140]:

‘Narrow the gulfs of execution and evaluation. Make things visible, both for execution and evaluation’.

2.3 Signature of an Operation

Next, we propose a \textit{preliminary} version of our third guideline, derived from the structure of total operations in \( \mathbb{Z} \). Often operations accept as domain elements the state and external input to deliver as range elements the state and additional output (e.g. \( \text{open} \) above). In the light of guideline \#2 above we notationally separate the message from other output as formulated in the following design guideline for a user-level operation, i.e., an operation which communicates with the user:

\textit{Guideline \#3}: Define every user-level operation, say \( \mathbf{f} \), based on the general format:

\[ \mathbf{f} : Input \times \text{State} \rightarrow \text{State} \times \text{Output} \times \text{Message} \tag{5} \]

Definition (5) is stated in a preliminary form. Further guidelines below will refine this definition.
2.4 Value of Undefined Output

Consider the following definition of a simple file system (Woodcock et al. [17]) where Key and Data are basic types.

\[
\text{File} \\
\text{contents} : \text{Key} \rightarrow \text{Data}
\]

A robust operation to read a file is:

\[
\text{Read} \\
\text{contents, contents'} : \text{Key} \rightarrow \text{Data} \\
k? : \text{Key} \\
d! : \text{Data} \\
r! : \text{Message} \\
( k? \in \text{dom contents} \land d! = \text{contents} k? \land \text{contents'} = \text{contents} \land \n r! = \text{okay} ) \lor \\
( k? \notin \text{dom contents} \land \text{contents'} = \text{contents} \land \n r! = \text{key\_in\_use} )
\]

Note that \( d! \) is unspecified under the error condition \( k? \notin \text{dom contents} \). Woodcock and Davies [17, p. 222] claim that an output variable 'can take any value' if the precondition is not satisfied. However, a possible interpretation of this claim is that the value \( d! \) could be given a value \( \text{contents} k \), for any \( k \in \text{dom contents} \) which is undesirable.

Instead, we could specify that the value of an output variable like \( d! \) above is undefined in the error case. This can be achieved by insisting that all sets from which output may be generated be 'lifted' to make provision for undefined values, much like the technique used in the semantics of programming languages (e.g. Schmidt [20, page 29]). If we denote an undefined value by \( \bot \), then we extend the set \( \text{Data} \) to \( \text{Data}_\bot = \text{Data} \cup \{ \bot \} \).

This observation leads to:

\underline{Guideline \#4:} Ensure that all sets from which output may be generated are extended to allow for undefined values.

The set \( \text{Message} \), representing the set of all messages, is of course an exception to guideline \#4, since we simply use an appropriate string to describe the situation. Therefore, we do not make the message part of the general \( \text{Output} \) parameter in (5) above, since a specifier may prefer to write this definition as:

\[
f : \text{Input} \times \text{State} \rightarrow \text{State} \times \text{Output}_\bot \times \text{Message}
\]

In line with guideline \#4, we make the undefined nature of \( d! \) explicit in the last disjunct in operation \( \text{Read} \) by adding \( d! = \bot \). We also replace the declaration \( d! : \text{Data} \) with \( d! : \text{Data}_\bot \).
2.5 Function Application

It is normally a trivial task to obtain a corresponding range element, given a function definition and a domain element, since we simply apply the function to the domain element. However, if we are given an element from the range, then it could be hard to obtain corresponding domain elements — we have to use the inverse of the function which is a relation in general. This problem is aggravated if we are presented with a second coordinate of a pair, where such a pair is the second coordinate of an enclosing, larger tuple.

Such a scenario is found in the classic specification of a telephone conference system by Carroll Morgan [21]:

Specify a telephone system whereby subscribers may engage in telephonic conferences. No phone may be used in more than one discussion group at a time. A subscriber may however engage in any number of these discussion groups. Each group is uniquely identified by a docket, assigned by the system when the first request for the group is initiated.

A conversation is a set of subscribers who participate in the conversation:

\[
\text{CONVERSATION} \ni \text{SUBSCRIBER}
\]

A request for a conversation has two components:

\[
\text{REQUEST} \\
\text{subscriber} : \text{SUBSCRIBER} \\
\text{conversation} : \text{CONVERSATION}
\]

Component \text{subscriber} represents who made the request, and \text{conversation} is what was requested.

A connection provided by the telephone system is defined by the schema:

\[
\text{CONNECTION} \\
\text{phones} : \mathbb{P} \text{PHONE} \\
\text{subscribers} : \mathbb{P} \text{SUBSCRIBER} \\
\text{using} : \text{SUBSCRIBER} \rightarrow \text{PHONE} \\
\text{dom using} = \text{subscribers} \\
\text{ran using} = \text{phones}
\]

The set \text{phones} denotes the set of phones that are connected; the set \text{subscribers} represents the conversation which the connected phones collectively support; \text{using} records for each subscriber in a conversation which phone he or she is using.

The abstract state space is given by schema \text{TS}, where \text{SUBSCRIBER}, \text{PHONE} and \text{DOCKET} are basic types:
Operation `ding_dong!`, defined by Morgan [21], takes a phone as input and returns the docket associated with the conversation which the phone is part of.

\[
\text{\texttt{\textcolor{red}{\textit{ding_dong!}}} } \\
\quad \Xi \texttt{TS} \\
\quad \texttt{phone? : PHONE} \\
\quad \texttt{docket! : DOCKET} \\
\quad \texttt{phone? \in connections(docket!), phones} \quad \text{(D1)} 
\]

Predicate (D1) above suggests a rather complex algorithm to be generated during refinement: The component `docket!` is not available beforehand, but only after the `ding_dong!` operation while the only input, `phone?`, is a range element\(^1\).

The above complication arises because input is accepted from the range of a function, and we could simplify the problem of finding the desired domain element by adding a component to `TS` that takes `phone?` as input.

This leads us to the following guideline:

**Guideline #5:** Where appropriate, ensure that no input involving a function is accepted entirely as just an element of the range of the function.

### 2.6 Cohesion

Our next guideline stems from the well-known software engineering principle of **cohesion**. Bahrami [23] defines cohesion as a measure of the 'single-purposesness' of an object. High cohesion is desirable and low cohesion is considered bad design, since low cohesion implies the grouping together of unrelated activities. Yourdon [8] states that a module has good cohesion if its purpose can be expressed with 'a simple sentence containing a single verb and a single object'.

The highest and most desirable kind of cohesion is **functional cohesion** (see e.g. Pfleeger [24]), which is the kind of cohesion described by Yourdon [8] and which we advocate in the design of a formal specification. The natural language

\(^1\) This specification style is related to the idea of *equal opportunity* ([22]) whereby something displayed by the system can be used by the user as input.
definition given by Yourdon [8] above is unfortunately too imprecise and we refine the idea below.

For the purpose of achieving high functional cohesion in a formal specification we propose breaking up every operation in the specification into a sequence of primitive operations. Thierry Scheurer [25] puts forward the following thesis: 'Set theory, based on logic, is a universal language in which all problems may be formulated and solved.' Since the Z specification language is based on first-order logic and a strongly typed fragment of Zermelo-Fraenkel set theory, we propose to define every primitive as manipulating just one component of the abstract state space of our system, using an operation or definition from standard set theory.

The above ideas on cohesion crystallise into the following guideline:

*Guideline #6:* Maintain high cohesion in a formal specification by defining every operation on the state as a sequence of primitives such that every primitive manipulates at most one state component using a standard set-theoretic operation or definition.

The use of primitives in this context is illustrated in Sect. 3 below. *Guideline #6* has an important benefit when reasoning about the properties of a specification: We show in Sect. 3.6 how this guideline facilitates an important proof obligation that arises from the interaction between a primitive and another primitive which reverses the effect of the first primitive.

### 2.7 Undo Changes in State Components

For our next design guideline we turn to the well-known HCI principle of 'undo' as advocated by Donald Norman [13, p. 131]:

Make it possible to reverse actions — to 'undo' them — or make it harder to do what cannot be reversed.

The above philosophy suggests the following guideline for specification work:

*Guideline #7:* Specify an undo counterpart for every operation that changes the state. The idea is to reverse the effect of a state change.

One could argue that the above principle is not really concerned with the actual writing of a formal specification. Nevertheless, if an undo operation is not part of a specification document then such operation will not be coded into the final software.

Note however:

1. We propose an 'undo' only if it is feasible to do so. For example, if an incorrect value is used in a calculation then we simply 'redo' the operation using the correct value instead of actually 'undoing' the erroneous result.
(2) We may have to remember some information in order to specify an undo. For example, suppose we delete an employee record using some key, only to discover that it was the wrong record. For the subsequent undo operation we still have the key available (since it would be communicated back to the user — see design guideline #2 above), but the particulars of the employee (e.g. name, address, etc.) would be lost. To avoid this problem we propose a component additional to the state space, and call it an environment. In the environment we put all auxiliary information, e.g. the detail of a deleted employee.

The use of an environment suggests the following redefinition of (6):

\[ f : Input \times Env \times State \rightarrow Env \times State \times Output \times Message \]  

(7)

### 2.8 Placing Control Statements

One of the Coad-Yourdon [8] object-oriented guidelines is called ‘Keep methods simple’, and under that heading a claim is made that if the method involves a lot of code or contains IF-THEN-ELSE statements or CASE statements then it’s a strong indication that the method’s class has been poorly factored — i.e., procedural code is being used to make decisions that should have been made in the inheritance hierarchy.

For specification work the above guideline translates into limiting control statements to the top level operations (which include our user-level operations). Our primitives therefore do not make any decisions, leading to our final guideline:

**Guideline #8**: Put the control statements in a formal specification as high up in the hierarchy as possible. In particular, put these statements in the top-level operations and not in the primitives.

To illustrate some of these guidelines, a small specification is constructed in Sect. 3 below.

### 3 A Library System

In this section we construct a small specification based on the above guidelines. Amongst others we illustrate the use of primitives, an undo operation utilising an environment, and we show in Sect. 3.6 how the use of primitives allows us to discharge an important proof obligation that arises from the interaction between an operation and its undo.

Consider a library system where users may register, borrow books from the library, and return them at a later date. A book is uniquely identified by an ISBN\(^2\). Other information pertaining to a book include the title, author, publisher and the year published. A user is uniquely identified by an identity code. Other relevant information include a user name and address.

\(^2\) For simplicity we assume that the library stocks at most one copy of a book.
3.1 Definition of the State Space

The state of the library is given by

\[
\begin{array}{l}
\text{Library} \\
\text{books : ISBN} \rightarrow \text{Title} \times \text{Author} \times \text{Publisher} \times \text{Year} \\
\text{users : ID} \rightarrow \text{Name} \times \text{Address} \\
\text{available : P ISBN} \\
\text{borrowed : ISBN} \rightarrow \text{ID} \\
\text{date : ISBN} \rightarrow \text{Date} \\
\end{array}
\]

\[
\text{available} \cup \text{dom borrowed} \subseteq \text{books}^3 \\
\text{available} \cap \text{dom borrowed} = \emptyset
\]

where ISBN, Title, Author, Publisher, Year, ID, Name, and Address are all basic types.

3.2 Definition of the Environment

Our system is concerned with books borrowed by users. A user is uniquely identified by an identity code and a book is uniquely identified by an ISBN. Therefore, we need to remember an identity code or an ISBN in the case of a subsequent undo operation\(^4\). Our environment is defined as:

\[
\begin{array}{l}
\text{LibEnv} \\
\text{id : ID} \\
\text{isbn : ISBN} \\
\end{array}
\]

One of our user-level operations, namely Borrow_Book for issuing a book to a user, is specified in the next section.

3.3 A User-Level Operation

Consider schema Borrow_Book below. Schema Borrow_Book adheres to design guidelines #2, #6, and #8 above by:

- maximising communication with the user,
- using a sequence of primitives to update state and environment components, and
- placing the control statements in the schema and not in the primitives.

\(^3\) We assume the library contains reference works that are available but cannot be borrowed by a user.

\(^4\) Our simple specification does not require the removal of books or users from the system, hence we need not keep the detail of a deleted book or user in the environment.
3.4 Definition of Primitives

The primitives in Borrow_Book are defined using ordinary set-theoretic notation. Env_isbn places an isbn in the environment to be used in the event of an undo and Env_id performs a similar function for a user id:

- Env_isbn : ISBN × LibEnv → LibEnv is given by
  \[ Env_isbn(isbn?, env) = env', \text{ where } env'.isbn = isbn? \]

- Env_id : ID × LibEnv → LibEnv is given by
  \[ Env_id(id?, env) = env', \text{ where } env'.id = id? \]

Primitive UnAvail makes a book unavailable while Borrow issues the book to a user:

- UnAvail : LibEnv × Library → LibEnv × Library is given by
  \[ \text{UnAvail}(env, library) = (env, library'), \text{ where } \]
  \[ \text{library'}.available = \text{library}.available \setminus \{ env.isbn \} \]

- Borrow : LibEnv × Library → Library is given by
  \[ \text{Borrow}(env, library) = library', \text{ where } \]
  \[ \text{library'}.borrowed = \text{library}.borrowed \cup \{ (env.isbn, env.id) \} \]

---

Note that the use of 'else if' in this way is non-standard in Z but the use of an if \text{P then } E_1 \text{ else } E_2 construct is — see Spivey [15].
3.5 Definition of an Undo

To undo the effect of operation Borrow_Book we specify:

\[
\text{Undo\_Borrow\_Book} \quad \Delta \text{Library}; \Xi \text{LibEnv} \\
\text{mes!} : \text{Message} \\
\theta \text{Library'} = \text{Undo\_UnAvail(} \text{Undo\_Borrow(} \theta \text{LibEnv,} \theta \text{Library})) \\
\text{mes!} = \text{Previous borrow operation reversed}
\]

Primitive \text{Undo\_Borrow} reverses the effect of \text{Borrow}:

- \text{Undo\_Borrow} : \text{LibEnv} \times \text{Library} \rightarrow \text{LibEnv} \times \text{Library} is given by
  \text{Undo\_Borrow}(\text{env}, \text{library}) = (\text{env}, \text{library'}), \text{ where}
  \text{library'.borrowed} = \{ \text{env.isbn} \} \cup \text{library.borrowed}

Primitive \text{Undo\_UnAvail} is specified in a similar fashion.

3.6 Discharging a Proof Obligation

Suppose an unregistered donor donates a new book to the library and thereby becomes a registered user. The following traditional schema describes this operation (see e.g. Potter et al. [26]).

\[
\text{Donate} \\
\Delta \text{Library}; \Delta \text{LibEnv} \\
\text{isbn?} : \text{ISBN} \\
\text{id!} : \text{ID} \downarrow \\
\text{title?} : \text{Title}; \text{aut?} : \text{Author}; \text{pub?} : \text{Publisher}; \text{yr?} : \text{Year} \\
\text{name?} : \text{Name}; \text{addr?} : \text{Address} \\
\text{mes!} : \text{Message} \\
(\text{id!} \notin \text{dom users} \land \text{isbn?} \notin \text{dom books} \land \\
\text{isbn} = \text{isbn?} \land \text{id} = \text{id?} \land \\
\text{books'} = \text{books} \cup \{ \text{isbn?} \mapsto (\text{title?}, \text{aut?}, \text{pub?}, \text{yr?}) \} \land \\
\text{users'} = \text{users} \cup \{ \text{id!} \mapsto (\text{name?}, \text{addr?}) \} \land \\
\text{available'} = \text{available} \cup \{ \text{isbn?} \} \land \\
\text{borrowed'} = \text{borrowed} \land \text{date'} = \text{date} \land \\
\text{mes!} = \text{OK} \\
\lor \\
((\text{dom users} = \text{ID} \lor \text{isbn?} \in \text{dom books}) \land \\
\theta \text{LibEnv'} = \theta \text{LibEnv} \land \\
\theta \text{Library'} = \theta \text{Library} \land \\
\text{id!} = \bot \land \\
\text{mes!} = \text{System error})
\]
An important proof obligation often stated in Z texts is to show that an operation followed by its undo counterpart leaves the state unchanged, i.e:

\[ \text{Donate} \equiv \text{Donate}^{-} \vdash \Xi \text{Library} \quad (8) \]

The popular and widely used first-order, resolution-based theorem prover, OTTER (see e.g. McCune [27], Wos [28]), has difficulty in proving (8) above.

If we, however, rewrite schema Donate as a sequence of 3 primitives operations

- \( \text{Capture_book} \) (say), a primitive to specify
  \[ \text{books}' = \text{books} \cup \{ \text{isbn} ? \mapsto (\text{title'?}, \text{aut'?, pub'?, yr'?}) \} \],
- \( \text{Register_user} \) (say), to specify \( \text{users}' = \text{users} \cup \{ \text{id}' \mapsto (\text{name'?}, \text{addr'?)} \} \),
- \( \text{Avail} \) (say), for \( \text{available}' = \text{available} \cup \{ \text{isbn}? \} \),

specify appropriate undo counterparts for each of the 3 primitives above, and perform 3 different proofs at the level of the primitives and their inverses, then OTTER easily finds a proof (for example) for

\[ \text{Capture_book} \equiv \text{Capture_book}^{-} \vdash \text{books}' = \text{books} \quad (9) \]

where \( \text{books} \) represents the component before primitive \( \text{Capture_book} \) and \( \text{books}' \) the same component after \( \text{Capture_book}^{-} \).

Quick proofs are also obtained for:

- \( \text{Register_user} \equiv \text{Register_user}^{-} \vdash \text{users}' = \text{users} \)
- \( \text{Avail} \equiv \text{Avail}^{-} \vdash \text{available}' = \text{available} \)

Failing to prove (8) above is more significant than may appear. A specifier may decide to leave schema Donate as is and attempt to perform 3 different simpler proofs, one which could be:

\[ \text{Donate} \equiv \text{Donate}^{-} \vdash \text{books}' = \text{books} \quad (10) \]

Again the theorem-prover has difficulty in proving (10) above, since it fails to find a proof in 30 minutes running on a Pentium III with a clock speed of 600 MHz, while a proof of (9) is found after just \( 0.15 \) seconds on the same machine.

The architecture of this last failed proof attempt is characterised by the presence of redundant information (see e.g. van der Poll & Labuschagne [29]) in the sense that changes in the state components \( \text{users} \) and \( \text{available} \) in schema Donate are irrelevant to a proof of (10). It turns out that such irrelevant information leads the theorem prover astray. Of course, the specifier can remove the redundant information from the proof attempt, which then boils down to using primitives in line with Guideline #6.

### 3.7 Summary

We demonstrated in this section how all our guidelines can be applied in constructing a formal specification. A proof obligation involving a primitive operation and its undo primitive was successfully discharged.
4 Analysis and Future Work

This paper presented a number of guidelines for drawing up a formal specification. Central to these ideas is the use of primitives, allowing a specifier to apply a 'divide-and-conquer' approach to the specification of an operation. We illustrated how these primitives facilitate the task of finding a short proof of a property of a composition where otherwise the presence of redundant information leads the theorem prover astray.

Additional implementation benefits may be realised through the use of primitives. Since every primitive changes at most one component of the state or environment we can on a multi-processor machine assign a processor to a primitive and if some of the primitives happen to be independent, then we may achieve true concurrency. (Having said this, we note that it is generally accepted that implementations issues are not to influence decisions made at the specification phase.) Also, on a threaded single-processor machine we can program a user-level operation as a task and each primitive as a thread within the task (see Silberschatz et al. [30] for a discussion of operating system threads). If a thread should block during execution then the possibility exists for another thread in the same task to start executing, speeding up the execution of the task as a whole.

An important issue emerges from the use of an environment in this work: In our model it is possible to undo only the effect of the previous update operation, and not any other (update) operation before the last one. For example, in some word processing packages (e.g. some later versions of WordPerfect®) it is possible to undo a number of previous operations, one after another. Typically a sequence of environments could be used for this purpose. Future work could concentrate on incorporating such sequences of environments.

References


From a Formal User Model to Design Rules

Paul Curzon\textsuperscript{1} and Ann Blandford\textsuperscript{2}

\textsuperscript{1} Middlesex University, Interaction Design Centre, Bramley Road, London N14 4YZ
\textsuperscript{2} University College London Interaction Centre, 26 Bedford Way, London WC1H 0AP
p.curzon@mdx.ac.uk, a.blandford@ucl.ac.uk

Abstract. Design rules can sometimes seem to contradict. In this paper we examine how a formal description of user behaviour can help explain the context when such rules are, or are not, applicable. We describe how they can be justified from a formally specified generic user model. This model was developed by formalising cognitively plausible behaviour, based on results from cognitive psychology. We examine how various classes of erroneous actions emerge from the underlying model. Our lightweight semi-formal reasoning from the user model makes predictions that could be used as the basis for further usability studies. Although the user model is very simple, a range of error patterns and design principles emerge.

1. Introduction

With the increasing ubiquity of interactive computer systems, their usability becomes increasingly important. Minor usability problems can scale to having major economic and social consequences. Usability of interactive designs has many aspects. In this paper, we focus on design principles that reduce the potential for “user error” occurring. We examine how, from the behaviour specified by a simple formal model of cognition, various potential erroneous actions emerge with poorly designed systems. Furthermore, we derive well-known design rules from the model. We use the fact that the rules are grounded in a formal model of cognition to explore the scope of application of the rules. Formal specification allows precision about the meaning of that which is formalised. Providing such precision to ensure different people have the same understanding of a concept has been suggested as the major benefit of formal models in interaction design [1]. One approach would be to formalise the design rules themselves (see [1,16]). Here, we semi-formally derive design rules from a formal model rather than just asserting them. In principle, formal derivations could also be done. By “semi-formal” we mean that we use informal high-level argument (though about a fully formal model), as opposed to the explicit application of (possibly machine-checked) inference rules of the underlying logic.

We first define simple principles of cognition. These are principles that generalise the way humans act in terms of the mental attributes of knowledge, tasks and goals. The principles considered do not cover the full range of human cognition. Rather they focus on particular aspects of cognitive behaviour. They are each backed up by
evidence from HCI and psychology studies. Those presented are not intended to be complete but to demonstrate the approach.

We have developed a formal model of these principles written in higher-order logic. This description is essentially a generic formal user model. By ‘generic’ we mean that it can be targeted to different tasks and interactive systems. The underlying principles of cognition are formalised once in the model, rather than having to be re-formalised for each new task or system of interest. Whilst higher-order logic is not essential for this, its use makes the formal specifications simpler and more natural that the use of a first-order logic would. Here we use it to make precise the general principles considered, to allow us to then reason about their consequences with respect to user error. Combining the principles of cognition into a single formal model rather than formalising them separately allows reasoning about their interaction, and how multiple minor errors might interact.

The principles, and more formally the user model, specify cognitively plausible behaviour (see [5]). That is, they specify possible traces of user actions that can be justified in terms of the specific principles. Of course users might also act outside this behaviour, about which situations the model says nothing. Its predictive power is bounded by the situations where people act according to the principles specified. That does not preclude useful results from being obtained, provided their scope is remembered. The model allows us to investigate what happens if a person does act in such plausible ways. The behaviour defined is neither “correct” nor “incorrect”. It could be either depending on the environment and task in question. It is, rather, “likely” behaviour.

We show how cognitively plausible behaviour can in specific circumstances be considered as resulting in erroneous actions. We discuss the circumstances in which such erroneous actions can result, reasoning from the formal model. In particular, we relate them to Hollnagel’s error phenotypes [13]. In doing so we identify cognitively plausible ways in which the erroneous actions could arise. We show that a wide variety are covered from even a minimal formal definition of cognitively plausible behaviour, demonstrating the generality of the approach.

Finally we semi-formally derive design rules from the formal model that, if followed, ensure that the erroneous actions identified will not be made for the specific cognitive reasons embodied by the principles of cognition. Even though the user model is capable of making the errors, the rules ensure that the environments in which they would emerge do not occur. Other errors are, of course, still possible. The design rules are well known and our contribution is not the rules themselves, but rather to show how they can be justified from a formalisation of a small set of principles. Because the design rules are derived we can be precise about their scope of applicability, for example unpacking the situations where different design rules appear at first sight to be contradictory.

We use railway ticket vending machines to illustrate the points. Such machines are ubiquitous and are increasingly replacing manned ticket offices. However, existing designs continue to exhibit design problems that encourage user error [18].

Previous work explored how the user model considered here could be used to analyse interactive system designs by treating it as a component of that system with a fixed design [7,8,9,10], proving that a specific task will always be completed eventually. This was done using an interactive proof system, HOL [12]. Here we use
the user model as the basis of reasoning about interactive systems in general. The process of doing so also acts, in part, to validate the model for formal verification. Our approach is similar to that of [2] in that we are working from a (albeit different and more formal) model of user behaviour to high level guidance. There the emphasis is on providing a semi-formal basis to underpin the craft skill in spotting when a design has potential usability problems. In contrast, we are concerned with guidance for a designer rather than for a post-facto usability analyst.

2. Formalised Principles of Cognitively Plausible Behaviour

Our user model was developed by formally modelling principles of cognitively plausible behaviour. We do not model erroneous actions explicitly (as is done for example in [11]). Instead, they emerge from an abstract description of cognitively plausible behaviour. The behaviour described could correspond to correct or incorrect actions being taken depending on the circumstances. The principles considered are: non-determinism; goal-based termination; task-based termination; reactive behaviour; communication goals; mental triggers; no-option-based termination; and relevance. This list is not intended to be exhaustive, but to cover a variety of classes of cognitive principles, based on the motor system, simple knowledge based cognition, goal-based cognition, etc. Also, some of the principles are formalised in a simple way, our intention at this stage being to test the approach, rather than modelling the full richness of the principles. In future work we will increase the richness of the descriptions. In the remainder of the paper we will refer to cognitively plausible behaviour when strictly meaning the subset of cognitively plausible behaviour embedded in the current version of our model.

By modelling the principles we are giving a knowledge level description in the terms of Newell [14]. We do not attempt to model underlying neural architecture nor the higher level cognitive architecture such as working memory units. Instead our model is that of an abstract specification, intended for ease of reasoning. The focus of the description is in terms of internal goals and knowledge of a user. This contrasts with a description of a user’s actions as, say, a finite state machine that makes no mention of such cognitive attributes.

The user model is based on a series of non-deterministic temporally guarded action rules. Each describes an action that a user could rationally make. The rules are grouped corresponding to the user performing actions for specific cognitively related reasons. Each such group then has a single generic description. Each rule has a guard-action form. They state that if the guard holds at some point then the NEXT action taken by the user is that given. By next in this context we mean the first action of interest taken by the user after the current point in time. The rules each have the form:

\[
guard \quad t \quad \text{AND} \quad \text{NEXT actions} \quad \text{action} \quad t
\]

stating that a guard is true at time \( t \) and the NEXT action performed from the list of actions relevant to the interaction (given by the list actions) is action. The action is identified by its position in the list of all actions. Here we give an overview; the formalisation is given in more detail in [8].
Non-determinism. In any situation any one of a series of behaviours that are plausible might be taken. The separate behaviours are specified as rules. Each rule is formalised in the user model non-deterministically. By this we mean it is one of a series of options, any of which could be taken. The model does not assert that a rule will be followed, just that it may be followed. Below, we present the formalisation of several such rules. They form the core of the user model. By combining them in this way, the model asserts that the behaviour of any rule whose guards are true at a point in time is cognitively plausible at that time. It cannot be assumed that any specific rule will be the one that the person will follow.

Goal-based termination behaviour. Cognitive psychology studies have shown that users intermittently, but persistently, terminate interactions as soon as their goal has been achieved [6]. It is formalised as a guarded rule as described above. We must supply a relation to the user model that indicates over time whether the goal is achieved or not. This is referred to as a special signal, goalachieved, in the formal definitions. We also use a special signal, finished, to indicate whether the user considers the interaction to be over. With a ticket machine this may correspond to the person walking away, starting a new interaction (perhaps by hitting a reset button), etc. Both goalachieved and finished are signals that, given a time, return a boolean value indicating whether the goal is achieved or the interaction terminated respectively at that time. If the goal is achieved at a time then the user model terminates the interaction next.

\[ \text{goalachieved}\ t \text{ AND NEXT actions finished}\ t \]

Note that goalachieved is a higher-order function and can as such represent an arbitrarily complex condition. It might, for example, be that the user has a particular object, that the count of some series of objects is greater than some number or a combination of such atomic conditions. In specifying the user model we just state that it is a boolean function whose value may vary over time. This makes use of the higher-order nature of the specification language.

Task-based termination behaviour. For the purposes of analysis, the model specifies that a user will terminate an interaction when their whole task is achieved. In achieving a goal, subsidiary tasks are often generated. For the user to complete the task associated with their goal they must also complete all subsidiary tasks. Examples of such tasks with respect to a ticket machine include taking back a credit card or taking change [6]. One way to specify these tasks would be to explicitly describe each such task. Instead we use the more general concept of an interaction invariant [8]. The underlying reason why these tasks must be performed is that in interacting with the system some part of the state must be temporarily perturbed in order to achieve the desired task. Before the interaction is completed such perturbations must be undone. For example, to pay at a ticket machine using a credit card requires the card being inserted and later returned. A condition on the state that holds at the start of the interaction – that the user has the card – must be restored by the end. We specify the need to perform these completion tasks indirectly by supplying the interaction invariant as a higher-order argument to the user model. The interaction invariant is an invariant in a similar sense to a loop invariant in program verification. It is an invariant at the level of abstraction of whole interactions. Full task completion involves not only completing the user's goal, but also restoring the invariant by
completing all the subsidiary tasks generated in the process. For a ticket machine the invariant might specify that the value of a person’s possessions at the end is at least as high as it was at the start of the interaction.

We assume that on completing the task in this sense of goal achieved and invariant restored, the interaction will be considered terminated by the user, irrespective of any other possible actions apart from actions already mentally triggered (discussed below). This is modelled using an if construct rather than disjunction to give it priority. If both the goal has been achieved and the invariant restored then the user will terminate the interaction, irrespective of what other non-deterministic rules may potentially be active. Otherwise one of the non-deterministic rules will be fired.

\[
\text{IF (invariant t) AND (goalachieved t)} \\
\text{THEN NEXT actions finished t} \\
\text{ELSE non-deterministic rules}
\]

**Reactive behaviour.** A user may react to a stimulus or message from a device, doing the action suggested by the stimulus. For example, if a flashing light comes on next to the coin slot of a ticket vending machine, a user might, if the light is noticed, react by inserting coins if it appears to help the user achieve their goal. Reactive behaviour is specified as a general class of behaviour: in a given interaction there may be many different stimuli to react to. Rather than specify this class of behaviour for each, we define the behaviour generically. A definition **REACT** gives the rule defining what it means to react to a given stimulus.

\[
\text{REACT actions stimulus action t =} \\
\text{stimulus t AND NEXT actions action t}
\]

If at time \(t\), the specified stimulus is active, the NEXT action taken by the user out of the possible actions, actions, at an unspecified later time, may be action.

As there may be a range of signals designed to be reactive, the user model is supplied with a list of stimulus-action pairs: \([s_1, a_1]; ... (s_n, a_n)\]. A list recursive relation, given a list of such pairs, extracts the components and asserts the above rule about them. They are combined using disjunction, in the recursive definition so are non-deterministic choices, and this definition is combined with the other non-deterministic rules. Guard and Action extract the components of a pair. “s :: stimuli” refers to the list with first element s and remainder of list stimuli.

\[
\text{(REACTIVE as [] t = FALSE) AND} \\
\text{(REACTIVE as (s :: stimuli) t =} \\
\text{(REACTIVE as stimuli t) OR} \\
\text{REACT as (Guard s) (Action s) t))}
\]

**Communication goal behaviour.** A user enters an interaction with knowledge of task dependent sub-goals that must be discharged. Given the opportunity, they may attempt to discharge any such communication goals [3]. The precise nature of the action associated with the communication goal may not be known in advance. A communication goal specification is a task level partial plan. It is a pre-determined
plan that has arisen from knowledge of the task in hand independent of the
environment in which that task will be accomplished. It is not a fully specified plan,
in that no order of the corresponding actions may be specified. In the sense of [3] a
communication goal is purely about information communication. Here we use the
idea more generally to include other actions that are known to be necessary to
to complete a task. For example, with the task of purchasing a ticket, in some way the
destination and ticket type must be specified as well as payment made. The way that
these must be done and their order may not be known in advance. However, a person
will enter an interaction with the aim of purchasing a ticket primed for these
communication goals to be addressed.

If the person sees an apparent opportunity to discharge a communication goal they
may do so. Once they have done so they will not expect to need to do so again. No
fixed order is assumed over how communication goals will be discharged if their
discharge is apparently possible. For example, if a “return ticket” button is visible
then the person may press that first if that is what they see first. If a button with their
destination is visible then they may press that first. Communication goals are one
reason why people do not just follow instructions.

Communication goals are modelled as guard-action pairs as for reactive signals.
The guard describes the situation under which the discharge of the communication
goal appears possible. It will include a label signal indicating that the input exists and
that it corresponds to the desired action. In the current version of the model, the use of
a special label signal is not built into the generic model but is included as part of the

As for reactive behaviour, a list of (guard, action) pairs are supplied to correspond
to each communication goal. A similar recursive definition to REACTIVE above is
defined and included as a disjunct with the non-deterministic rules. This determines
when a communication goal may be discharged. However, unlike the reactive signal
list that does not change through an interaction, communication goals are discharged.
This corresponds to them disappearing from the user’s mental list of intentions. We
model this by removing them from the communication goal list when done. A
daemon, separate from the non-deterministic rules, does this. It monitors the actions
taken by the user on each cycle, removing any from the list used for the subsequent
cycle. The action removed may be taken for some reason other than it being a
communication goal, such as due to reactive behaviour. All that matters to the
daemon is that it is taken. The communication goal list that a user enters the
interaction with initially is provided as an argument to the user model.

Mental triggers. A user commits to taking an action in a way that cannot be
revoked after a certain point. Once a signal has been sent from the brain to the motor
system to take an action, the signal cannot be stopped even if the person becomes
aware that it is wrong before the action is taken.

Rather than associate an external stimulus directly with an external action using the
disjunctive rules, we associate them with mental “actions” that trigger the process of
taking the actual action. Thus the actions in each of the rules described so far will not
be externally visible actions, but internal mental actions. For example, on deciding to
press a button labelled with the destination “Liverpool Street Station”, at the point
when the decision is made the mental trigger action takes place and after a very short
delay, the actual action takes place. A further category of trigger rules is then
introduced that links the mental decision to the actual action. If one of the mental actions is taken on a cycle then the next action will be the externally visible action it triggers. There is always at least a one-cycle delay between the trigger and external action. A recursive function combines a list of triggers into a series of choices as with the reactive rules. The user model must be supplied with a guard-action pair list linking mental triggers with external actions. As with task-based termination, mental triggers are given a higher priority than the non-deterministic rules. If a trigger is fired then it will be the next action taken. Only if no fired trigger is outstanding do the other rules come into play, including task-based termination.

**No-option-based termination behaviour.** A user may terminate an interaction when there is no apparent action they can take that would help complete the task. For example, if on a touch screen ticket machine, the user wishes to buy a weekly season ticket, but the options presented include nothing about season tickets, then the person might give up, assuming their goal is not achievable. The model includes a final default non-deterministic rule that models this case. The guard to this rule is constructed automatically in the model from the information supplied to create the other rules. In practice, in this situation, people could behave in a range of ways including pressing buttons at random. Our model treats a situation where no “rational” action is available as resulting in the interaction terminating – even if a possible action may become possible in the future. Note that a possible action that a person could take is to wait. However, they will only do so given some reason – that is it must be an action in an explicit reactive rule. For example, a ticket machine might display a message “Please Wait”. If they see it, the person reacts by waiting.

**Relevance:** A user will only take an action if there is something to suggest it corresponds to the desired effect. We do not currently model this explicitly: however, it is implicit in most of the rules. For example, communication goals and the termination rules are by definition only fired when relevant. In particular, the “label” signals referred to above are intended to address aspects of relevance. A button for the destination “Liverpool Street” is modelled by one signal representing whether the button is visible/relevant at a given time and a second about whether the button is pressed at each time instance.

**Putting it together:** The core rules are combined with other house-keeping rules (most notably, the communication goal filtering daemon) and a model of possessions that specifies, for example, that a user ceases to have a possession if it is given up. We omit the details here due to space constraints. A further clause added to the model is the initial conditions – notably the initial communication goal list. These are all combined using conjunction into a single relation **USER** that models the full user model – taking as arguments the various pieces of information such as the goal, interaction invariant, list of actions, etc referred to in the description above.

### 3. The Erroneous Actions that Emerge

Erroneous actions are the proximate cause of failure attributed to human error in the sense that it was a particular action (or inaction) that immediately caused the problem: users pressing a button at the wrong time, for example. However, to
understand the problem, and so ensure it does not happen again, approaches that consider the proximate causes alone are insufficient. It is important to consider why the person took that action. The ultimate causes can have many sources. Here we consider situations where the ultimate causes of an error are that limitations of human cognition have not been addressed in the design. An example might be that the person pressed the button at that moment because their knowledge of the task suggested it sensible. Hollnagel [13] distinguishes between human error phenotypes (classes of erroneous actions) and genotypes (the underlying psychological cause). He identifies a range of simple phenotypes: repetition of an action, reversing the order of actions, omission of actions, late actions, early actions, replacement of one action by another, insertion of an additional action from elsewhere in the task, and intrusion of an additional action unrelated to the task. These are single deviations from required behaviour.

In practical designs it is generally not feasible to make erroneous actions impossible. Fields [11] uses model-checking to identify errors by introducing the above problems explicitly into task specifications. A problem with this approach is that it gives many false negatives: few tasks are possible if such errors are arbitrarily made. The verifier must still determine which are real problems. A definition of what is cognitively plausible is one basis upon which to make this judgement. A more appropriate aim is therefore to ensure that cognitively plausible erroneous actions are not made. To ensure this, it is necessary to consider the genotypes of the possible erroneous actions. We examine how our simple user model can exhibit behaviour corresponding to these errors. We thus show, based on semi-formal reasoning about the formal model, that, from the minimal principles we started with, a wide range of classes of erroneous actions in the form of phenotypes occur.

We now look at each of Hollnagel’s simple phenotypes and look at the situations where it is cognitively plausible for them to occur. We are not claiming to model all cognitively plausible phenotypical actions. There are other ways each particular phenotype could occur for other principles we do not yet consider. However, not all the errors that result from the model were explicitly considered when the principles and user model were defined. The scope of the model in terms of erroneous actions is wider than those it was originally expected to encompass.

**Repetition of actions.** The first class of erroneous action is to repeat an action already performed. There are situations where this is cognitively plausible according to our user model. The current user model will repeat actions if guided to do so by the device in a reactive manner. If the guards of an action remain true then the user model may follow those instructions a second time since there is nothing in the model to prevent this. If the guidance is erroneous then the user model will make an erroneous action. Occasions where an interactive device asks erroneously for an action that has already been performed are perhaps rare (and it might be argued that in this situation the action was correct but the device incorrect). However, one way it could occur is due to a lack of feedback to indicate the action was performed successfully. The current user model would do this if reactive signals guided the action and continued to do so after the action had been completed – for example, if the reactive stimulus was not switched off immediately the action was performed. In particular, with a ticket machine, if a flashing light next to a coin slot continued to flash for some period after
the correct money had been inserted a person might assume they had not inserted enough and start to insert more.

An action originally performed as a communication goal could be repeated if a reactive prompt to do so later appeared (though not the other way round since once performed reactively the action is removed as a communication goal). For example, if a person initially pressed the button for “Liverpool Street” and was later presented with a screen asking them to select a destination they might do so again.

**Reversing the order of actions.** A further class of error is to reverse the order of two actions. This pattern of behaviour can arise from our model as a result of the way communication goals are modelled. In particular, communication goals can be discharged by the user model in any order. Therefore, if an interactive system requires a particular sequence, then the order may be erroneously reversed by the user model if they correspond to communication goals. A person might insert money and then press the destination button when a particular ticket machine requires the money to be inserted second. This does not apply to non-communication goal actions, however. For example, two actions that are device dependent (pressing a confirmation button and one to release change, for example) will not be reversed by the user model.

**Omission of actions.** The user model may omit actions at the end of a sequence. In particular, it may terminate the interaction at any point once the goal has been achieved. For example, once the person is holding the ticket they intended to buy, they may walk away from the machine, leaving their change, credit card or even return portion of their ticket. Whatever other rules are active, once the goal is achieved, the completion rule is active, so it could be fired.

The user model may also omit trailing actions if there is no apparent action possible for any period of time. If at any time instance the guard of no other rule is active, then the guard of the termination rule becomes active and so the user model terminates. There must always be some action possible though this could be to pause but only if given reactive guidance to do so. For example, if there is a period when the ticket machine prints the ticket, where the person must do nothing, then with no feedback they may abort. In this respect the user model does not quite reflect the way people behave – since if there is no action possible the user model is guaranteed to terminate, whereas in reality a person might pause before giving up. However, if the concern is for user errors, this is not a critical difference as in either case termination is possible so it cannot be guaranteed that the task was completed.

If the user model took an action early due to it corresponding to a communication goal (selecting a destination first instead of ticket type, for example) then the model would assume that the action had had the desired effect. The action (selecting a destination) would be removed from the communication goal list: the model “believes” it has been performed. It then would not be done at the appropriate point in the interaction i.e., a second error – this time an omission error – would occur. Note that in this situation the correct action would be a repetition of the earlier action – repetition is not an error in this situation.

**Late actions.** The user model does not put any time bounds on actions. All rules simply assert that once an action is selected then it will eventually occur. If any action must be done in a time critical manner, then the user model will be capable of failing to do so. In practice this is too restrictive – it means the current user model will always be able to fail with a device that resets after some time interval, for example,
as would be normal for a ticket machine. Where such time criticality is inherent in a
design, extra assumptions that deadlines are met would need to be added explicitly.

**Early actions.** If there are periods when an action can apparently be performed,
but if performed is ignored by the computer system, then in some circumstances the
user model would take the next action early. In particular, if the user has outstanding
communication goals then the corresponding actions may be taken early. This will
potentially occur even if the device gives explicit guidance that the user must wait.
This corresponds to the situation where a person does not notice the guidance but
takes the action because they know they have to and have seen the opportunity.
Similarly, if the device is presenting an apparent opportunity for reactive behaviour
before it is ready to accept that action then the user model could react to it.

**Replacement of one action by another.** Replacement can occur due to
communication goals if the device requires a specific action to be taken but its
interface suggests that a communication goal can be discharged. For example, if the
coin slot is visible but a destination selection required first, the person may insert
money as discussed earlier. The user model may make the communication goal action
rather than the required one, even if instructions are being displayed. Similarly, if
reactive signals give incorrect guidance that suggests an action should be taken then
that guidance may be followed. It can also occur due to trigger rules and
environmental changes. In particular, if a change of state in the computer system can
occur, that is not in response to a user action, then if the user model has already
committed to some action (such as pressing a button), but its effect changes between
the commitment being made and the action actually being taken, then the wrong effect
will occur. This can lead to a person doing something they know is wrong. The
change could occur, for example, due to a time-out in the machine or an
environmental change such as the time changing to off-peak travel.

**Insertion of actions from elsewhere in the task.** Insertion of an action can occur
with communication goals. They can be attempted by the user model at any point in
the interaction where the opportunity to discharge them apparently presents itself.
With reactive tasks, it will occur only if the device gives a reactive signal to suggest it
can be done when it cannot.

**Intrusion of errors not related to the task.** Actions not related to the task can
intrude with the user model as a result of reactive signals on the device. Thus, if a
device supports multiple tasks and uses reactive signals that signal an action to be
performed that is not part of the task, such an action may be taken.

In summary, the simple principles of cognition implemented within the model
generate behaviours that account for Hollnagel’s various phenotypes. Similarly, those
same principles of cognition can be used to derive and reason about design principles.

4. **Design Rules**

In this section we examine a range of usability design rules and look at how they
solve the problems identified if a design is to be usable at least by our user model (and
successfully verified with respect to it). Ad-hoc lists of design rules can easily appear
to be contradictory or only apply in certain situations. By basing them on cognitively
plausible principles, we can start to reason about their scope and/or couch them in
terms that makes this scope more precise. For example, should systems always be
permissive [17] allowing any action to be taken or only under certain circumstances?
At first sight permissiveness appears to contradict the idea of a forcing function [15]
when only certain actions are made possible. By reasoning from cognitive principles
we can untangle these surface contradictions.

**Completion actions:** The user model contains a rule to terminate if the goal is
achieved. Whatever other rules are active, this one could be activated due to the non-
deterministic nature of the rules. The user model can therefore terminate the moment
its goal is achieved. Furthermore, no messages or other signals from the device can
prevent this as it would just result in additional rules being active which cannot
preclude some other action being taken. For the user model to guarantee to not
terminate early for this reason it must only be possible for a user to terminate once the
task is completed. This means that for our user model the task must be completed no
later than the goal. Any design that requires the user model to perform extra
completion tasks must ensure they are done before it is possible for the goal to be
achieved. The rule will then only be active precisely when the task termination rule
will be active, so that termination does not occur before the task rule is achieved. In
practice – for example, when termination involves logging out from a system – it may
not always be possible to satisfy this design rule; in such situations, another means of
restoring the invariant needs to be found.

An attempted verification of a design that did not follow this design rule would fail
because there would be a path where the goal was achieved and so termination would
occur on that path, when the task was not achieved. In particular, as noted above,
providing extra information is not sufficient. For a ticket machine, taking the ticket
must be the last action of the user. They must by then have taken change or hold their
credit card, or these must be returned in the same place and at the same time as the
ticket. Multiple ticket parts (e.g., the return ticket) must also be dispensed together.

**Provide information about what to do.** Actions that are not communication goals
can only be triggered in the model if they are a response to reactive signals –
information indicating that the given rule is the next to be performed to achieve the
given task. Therefore, if an action must be performed that does not correspond to a
communication goal then information in the form of clear reactive guidance needs to
be provided to tell the user to take the action. In the case of a ticket machine, if a
button must be pressed to confirm the ticket selected is the one required, then
instructions to do this must be provided. For communication goal actions, reactive
information is not needed, though information linking the communication goal to the
specific action is needed: something (such as the presence of a visible coin slot for
inserting money) must make it clear that the communication goal can be discharged.

**Providing information is not enough.** The above design rule concerned always
providing information. This design rule is that that is not good enough – so might
appear to be contradictory. However, it depends on the situation. A simple design rule
might be to clearly indicate the order that actions should be taken. This approach is
often seen on ticket machines, where for example a panel gives instructions or lights
flash to indicate the next set of buttons to press. However, the user model is non-
deterministic. There may be several rules active and therefore several possible actions
that could be taken. Reactive signals are not modelled as having any higher priority
than any other signal. Other possible actions are, for example, to terminate the interaction (if the goal is achieved), or discharge a communication goal. If the guards of such rules are active then they will be possible actions. Making other signals true cannot make such a guard false; it can only make false guards true, so increasing the range of possible actions. Therefore, just providing flashing lights or beeps or other reactive signals is not enough to ensure correct operation.

An attempted verification of such a design would fail because it would not be possible to prove that the correct action was taken. Some other action would be possible which could ultimately lead to the user aborting the interaction. If any possible path leads to abortion before the user’s goal is achieved then the correctness statement will be unprovable as it states that the goal is achieved on all paths.

Is providing information ever enough? According to the model – yes. As noted above it is sufficient provided the user has nothing else possible to do and the relevant action clearly takes them towards their goal. In particular (for our principles) if all communication goals are discharged (the ticket has been fully specified and sufficient money inserted) and the goal is not yet achieved (no ticket is held yet) then providing information is both useful and necessary.

Forcing functions. The fact that the user model is capable of taking several different options and that giving reactive signals and messages is not enough means that some other way is needed to ensure the options are narrowed down to only the correct ones. As Norman [15] suggests, in good design, only correct actions for the range of tasks supported at a point should be possible. This suggests the use of forcing functions. Somehow the design must ensure that the only cognitively plausible actions are correct ones. This does not mean there must only be one button to press at any time, but only one button that can possibly be of use. Within the limits of the model, this means that if communication goals are not yet discharged, and should not yet be discharged, then there should be no apparent opportunity to discharge them. For example, a soft screen might be used so that the only buttons pressable correspond to ones that can now correctly be pressed. If money cannot be inserted then the coin slot should be closed. Similarly, the solution to post-completion errors is to not allow the goal to be achieved until the task is completed – forcing the user to complete other completion tasks first (where possible), as discussed above.

Permissiveness. Forcing functions follow the design principle that the options available to the user should be reduced. An alternative way of solving the same problem is to do the opposite and make the design permissive [17]: that is, it does not force a particular ordering of events. In this case, the design should be such that each of the actions that can be taken by the user model are accepted by the design and lead to the task being achieved.

With our user model, permissiveness cannot be used universally, however. For example, it is not sufficient with completion tasks to allow them to be done in any order. As we have seen, if the goal is achieved before the task is completed then the user model leaves open the possibility of termination. There is no way the design can recover – once the user model terminates it does not re-start the task. Therefore, in this situation, being permissive does not work. The ticket must be released last. That action corresponds to the goal so cannot be permissive.

At times in an interaction when communication goals are outstanding, the user model could discharge them if the opportunity is present. Thus permissiveness is a
useful design rule to apply to communication goals. In particular, permissiveness should be applied if forcing functions are not used when communication goals are active. A communication goal that appears dischargable should be dischargable. For example, a ticket machine could allow destination and ticket type to be chosen in any order.

**Visibility.** The user model provides for both reactive behaviour and directly goal-based behaviour. All user model actions are guarded by a signal indicating the presence of information suggesting it is an appropriate thing to do. If a control is not labelled then the user model will not be able to take the action. Thus all controls must be labelled if the user model is to use them. This does not mean that such labels must be written. It could be decided that the form of a control is sufficient to warrant the signal being asserted. For example, a coin slot advertises by its form that it is for the insertion of coins. This would need to be decided by a usability expert using complementary techniques. Also, it only needs to be visible at the point where the user model must take the action. Thus visibility need not be universal.

**Always give feedback immediately if the user must wait.** If there is no possible action apparent to the user model then it will abort. If the person must wait while a ticket is printed, then feedback to wait should appear immediately and nothing else should be apparently possible – no other buttons visible, for example. One possible reactive action can always be to pause provided it is guarded by the existence of a “please wait” message.

**Do not change the interface under the user’s feet.** The existence of trigger behaviour, where there is a delay between the user making a decision and acting on it, but after which they cannot stop themselves, leads to a design rule that the interface should not change except in response to user action. More specifically, a possible design rule is that no input to the computer system should change its meaning spontaneously. This is quite restrictive, however.

Less restrictive design possibilities are available to overcome the problems. For example, most ticket machines have timeouts – if no action is made in some period then the machine resets to some initial state. The user model does not strictly support such behaviour at present. However, one possibility with the current limited user model, and as used by some cash points, is to ask the user if they want more time after some delay. However, this means the buttons change their meanings. What did mean “I want to go to Liverpool Street” suddenly means “No I do not want more time”, for example. Such problems can be overcome, provided the old buttons all mean “I want more time”, and the one that means no more time previously was not linked to any action – or with a soft-button interface did not exist at all. Allowing such a design would only work with the user model if reactive signals were being used, as if the action were taken as a result of a communication goal, then that communication goal would have been discharged. The user model would only take the action again if prompted reactively to do so.

**Where possible, determine the user’s task early.** The user model can act reactively to actions intended only for other tasks. The problems can be overcome if multiple task devices determine the task to be performed at the first point there is any divergence between tasks. For example, a ticket machine that can also be used as a cash point may have a common initial sequence of inserting a credit card. However, as soon as the tasks diverge, then the next device action should be to determine the
task the user is engaged in, in a way that makes no other actions (specifically communication goals for any of the tasks) apparently possible. From that point on actions from other tasks will not need to intrude in the design. This is important because a communication goal can be discharged at any point where apparently possible. In complex situations this is likely to be difficult to achieve.

5. Conclusions and Further Work

We have outlined a formal description of a very simple user model. The user model describes fallible behaviour. However, rather than explicitly describing erroneous behaviour, it is based on cognitively plausible behaviour. Despite this we show that a wide variety of erroneous actions can occur from the behaviour described in appropriate circumstances. We have considered how devices (software, hardware or even everyday objects) must be designed if a person acting as specified by the user model would be able to successfully use the device. We have shown how well-known design rules, if followed, would allow this to occur. Each of these rules removes potential sources of user error that would prevent the verification of a design against the user model using the techniques described in [7]. We thus provide a theoretically based set of design rules, built upon a formal model. This model has very precise semantics that are open to inspection. Of course our reasoning is about what the user model might do rather than about any real person. As such, the results should be treated with care. However, errors that the user model could make are cognitively plausible and so worth attention.

One of our aims was to demonstrate a lightweight use of formal methods. As such, we have started with a formal description of user behaviour and used it as the basis of semi-formal reasoning about what erroneous behaviour emerges, and the design principles that would prevent that behaviour emerging. Such semi-formal reasoning could contain errors. We also intend to explore the formal, machine-checked derivation of the design principles. Using HOL (the proof system the user model is defined within), this would involve giving formal descriptions of design rules and proving that – under the assumptions of the user model – particular erroneous situations would not occur.

Our model is intended to demonstrate the principles of the approach and as such is still rather simplistic, covering only a small subset of cognitively plausible behaviour. As we develop it, it will give a more accurate description of what is cognitively plausible. We intend to extend it in a variety of ways. As this is done a wider variety of erroneous behaviour would be possible. For example, habitual behaviour is currently not modelled. Also many aspects of an interactive systems are parameters of the user model. However, in many cases a user actually determines this information by observation of the machine’s interface. The model could be modified so that such information is an input to the model (i.e. collected as part of the interaction) rather than supplied by the verifier. We have essentially made predictions about the effects of following design rules. In broad scope these are well known and in most cases based on usability experiments. However, one of our arguments has been that more detailed predictions can be made about the scope of the design rules, relating them
back to concepts such as communication goals. The predictions resulting from the model could be used as the basis for designing further experiments to validate the model, or further refine it. We have also suggested there are tasks where it might be very difficult or even impossible to produce a design that satisfies all the underlying principles, so that some may need to be sacrificed in particular situations. We intend to explore this issue further.

References

A Coloured Petri Net Formalisation for a UML-based Notation Applied to Cooperative System Modelling

José Luis Garrido and Miguel Gea

Dpt. Lenguajes y Sistemas Informáticos, University of Granada, E.T.S.I. Informática, C/Daniel Saucedo Aranda s/n, 18071 Granada, Spain
http://gia.ugr.es
{jgarrido,mgea}@ugr.es

Abstract. New approaches are currently being adopted to address the development of cooperative systems, although not many standards exist that can be used to develop this type of interactive system. We apply the standard Unified Modelling Language (UML) notation within a methodology aimed at the analysis and design of such systems, and present a semantic formalisation of the UML notation used to model cooperative systems. The semantics and its application are described on the basis of translation schemes to Coloured Petri Nets and the benefits of formalisation are shown.

1 Introduction

To date, Computer-Supported Cooperative Work (CSCW) [15] has comprehended various systems: Workflow Management Systems (WMS), computer-mediated communication (CMC) (e.g. e-mail), decision support systems, shared artefacts and applications (e.g. shared whiteboards, collaborative writing systems), meeting systems, etc. These can be categorised in several ways. One of these is by the function that the system performs, and another interesting one is based on two dimensions: time (synchronous or asynchronous) and space (co-located or remote). The outcome matrix is very useful to refer to the particular circumstances that a groupware application aims to address. Groupware [6,3] has been defined as a computer-based system that supports groups of people engaged in a common task (or goal) and that provides an interface to a shared environment. We note that in the set of fields and systems embraced by CSCW there are many explicit and implicit related concepts. These tend to either logical or technological aspects, such as interaction, communication, coordination, information, group, behaviour, distribution, control, etc. An additional difficulty is that the same term is frequently used in two or more fields but with different meanings and implications. For instance, human-computer interaction (HCI) particularly addresses psychological and computer issues; on the other hand, in human-human interaction within group activities, social issues acquire more relevance. An introductory study of the concepts involved in specifying the general principles and properties of CSCW systems is presented in [7].

34
The development of groupware applications is more difficult than that of a single-user application, because social protocols and group activities must be taken into account for a successful design [12], and so techniques aimed at enhancing group interaction activities should be applied. The inherent complexity of any interactive system requires a great deal of effort in the formulation of specifications, and formal methods may be used to achieve this goal [2, 13]. At the modelling stage, support should be provided to study the system being developed.

The work presented in this paper is part of a new methodology called AMENITIES [9] (acronym for A MEthodology for aNalysis and design of cooperateIVe systEmS). This methodology is based on behaviour and task models for the analysis and design of generic cooperative systems. The set of behaviour and task models (called Cooperative Model of AMENITIES) is a conceptual model which comprises the foundation of the methodology. The paper shows how to formalise the semantics for the Cooperative Model, which uses a UML-based notation, by describing translation schemes to Coloured Petri Net (CPN) formalism [14]. The paper is organised as follows. Section 2 reviews related work. Sect. 3 justifies the use of CPN formalism to define UML semantics. The conceptual framework for the Cooperative Model is introduced in Sect. 4 and an application example of this model is presented in Sect. 5. Then, on the basis of the example provided, the question is addressed of how, formally, to represent the model to be studied (Sect. 6). Finally, Sect. 7 summarises the main conclusions.

2 Related Work

Several approaches have been proposed to specify cooperative systems, focused on representing user tasks [19]. Thus, the system specification is a collection of user goals each of which is defined by the sequence of tasks that allow us to achieve a desired objective. Several notations have been proposed, such as GTA (Groupware Task Analysis) [29] and CTT (ConcurTaskTrees) [22]. GTA proposes an ontology-based system study for task world models, that is, a framework in which participants (agents and users roles), artefacts (objects) and situations (goals, events) take place. Moreover, a set of relationships between these are clearly identified (uses, performed-by, play, etc). CTT provides a hierarchical graphical notation to describe concurrent tasks, and allows us to specify cooperation by adding a hierarchical specification with temporal constraints for each cooperative task. This extension and others aim to establish common tasks for several users and the relationships between them. An approach describing both task and system models by means of a dialect of Petri Nets (ICO formalism) is presented in [21], demonstrating how formal task models improve the design of interactive systems.

These task-based approaches study the system from the user’s point of view, describing the cognitive skills required for correct use. However, most such techniques do not consider certain dynamic aspects in the problem domain [18]. For example, the user’s role may change during real situations (e.g. the responsibili-
ties in an office department), and the ways in which the objectives to be achieved
can vary (e.g. a new commercial strategy, different work organisation). Thus,
group organisation and evolution over time should be taken into account when
social organisations are described.

On the other hand, our work is based on UML notation to describe co-
operative systems by extending task-based approaches with ethnographic and
cognitive issues. The advantage of this election is the use of a standard suc-
cessfully adopted in software engineering. However, it is a semi-formal notation
and various proposals have been made to formalise UML for different purposes,
including:

– Stochastic Petri Nets, derived from UML statecharts and a collaboration
diagram, for performance analysis [16].
– Validation of architectural software design with UML collaboration diagrams
using CPN behavioural templates [23].
– Development of the CPN of a system by deriving Object Oriented Petri Nets
Models from UML statecharts and connecting them using UML collaboration
diagrams [26].
– Defining formal operational semantics by transforming UML state diagrams
into graphs [11].
– Giving execution semantics for UML activity diagrams intended for workflow
modelling [5].
– Definition of a semantics for collaboration and activity diagrams based on
Place-Transition Petri Nets with informal inscriptions [10].

In general, these proposals focus on software analysis or toolkit construction
for code generation, although some consider workflow modelling using activity
diagrams.

3 UML Semantics versus CPN Semantics

The OMG Unified Modelling Language Specification [20] (in its current ver-
sion 1.4) specifies syntax and informal semantics of the notations embraced by
UML. This specification is clearly focused on software models. In particular,
with respect to the behavioural UML notations that we use for the cooperative
model of AMENITIES, the following text appearing in this reference should be
highlighted:

1. Statecharts. "A statechart diagram can be used to describe the behaviour
of instances of a model element such as an object or an interaction... The
semantics and notation described are substantially those of David Harel's
statecharts with modifications to make them object-oriented”.

2. Activity diagrams. "An activity graph is a variation of a state machine in
which the states represent the performance of actions or subactivities... It
represents a state machine of a procedure itself".
A statechart (or activity diagram) is a graph that represents a state machine. The relationships between a state machine and its context have no special notation. The state machine also provides the semantic foundation for activity graphs. The specification document provides, for statecharts and activity diagrams, both a specification of the notation (graphic syntax) and an informal definition (in English) of their semantics.

The semantics of a UML state machine (i.e. the metamodel) is described in terms of the operations of a hypothetical machine that implements a state machine specification. Thus, this operational semantics is defined by the following key components:

- A queue of incoming event instances to be dispatched.
- The dispatcher mechanism for event processing.
- An event processor that processes dispatched event instances according to the general semantics and the specific form of the state machine in question.

Our main motivation in describing the semantics of UML notation applied to CSCW systems by using CPNs is that the existence of several variation points allows different semantic interpretations that might be required in different application domains. This is usually our case, and so high-level Petri Nets are used for the formal specification. This provides the following advantages:

- CPNs provide true concurrency semantics by means of the step concept, i.e. when at least two non-conflicting transitions may occur at the same time. It is the ideal situation for our application domain, as the model must be supported by a multithreaded state machine (several actors moving within the same space of states).
- The combination of states, activities, decisions, data, events and complex transitions (namely fork-join constructions) means that the UML state machine notation is very rich. CPNs allow us to express, in the same formalism, both the kind of system we are dealing with and its execution.
- Formal semantics is better in order to carry out a complete and highly automated analysis (i.e. validation and verification of properties) for the system being designed.

4 Framework

The cooperative model of AMENITIES, which is the core of the methodology, adopts an overall view of a system. Hence, the system embraces computer-based systems as well as the end-users themselves and related aspects such as organisation, communication, collaboration [27] and coordination [17]. This paper does not deal with the user's behaviour except that related to the interaction between different users. It is even possible to model interactions in which no computer-based system is involved [8]. The model allows us to carry out task analysis and modelling, as well as represent other related aspects such as roles, capabilities, constraints, etc. The notation used is basically that of UML [25] only that instead
of applying it to specify concrete classes for implementation, we consider other more abstract classes (group, role, ...) in the domain of cooperative systems. Because the emphasis is to study behaviour rather than structural aspects, class diagrams are not shown in this paper.

Several definitions of framework have been made, depending on the level where it is applied. In this context, a framework should give a higher common abstraction level between a family of related systems to be described in terms of general concepts. Thus, a framework is a pattern encompassing the principal common concepts of a kind of system and the relationships between them.

We define the basic terminology as follows. An action is a basic unit of work, executable atomically. A subactivity is a set of related subactivities and/or actions. A task is a set of subactivities intended to achieve certain goals. A role is a designator for a set of capabilities to carry out work, including tasks, skills, constraints and responsibilities/authorities. An actor is a user, program, or entity that can play a role in the execution of, or responsibility for, tasks. A cooperative task is one that must be carried out by more than one actor, playing either the same role or different ones. A group is a set of actors playing roles and organised around one or more cooperative tasks. A group may be composed, i.e. formed from related subgroups. A constraint (also called a law) is a limitation imposed by the system that allows it to adjust the set of possible behaviours dynamically. Finally, a capability is a constraint that directly affects an actor or group, enabling them to respond to new challenges offered by the system, as a result either of external events or of internal events produced by the interaction between participants (i.e. actors, groups or the system itself).

The method for building the cooperative model (as is shown in the next section) is based on two key concepts defined above: work and group. We use the notion of task to structure and describe the work that must be performed by the group. This provides the way to translate work, i.e. something that is tacit and implicit, into something that is concrete and explicit. Nonetheless, tasks are also considered at a very abstract level as noted below. A group, on the other hand, can be more or less explicit. Sometimes organisational aspects determine the way people work, but in other cases personal and/or operational aspects are the basis for organising people in order to perform an activity. The notion of role, in any case, allows us both to specify groups as needed and to establish dynamic relations between actors and tasks.

5 Example of Cooperative Model

As an example to introduce the syntax and semantics of statecharts and activity diagrams of UML according to the application domain (i.e. cooperative work), we have modelled the current system used in Emergency Coordination Centres in Sweden and the U.S.A. [1]. These systems were designed and implemented fulfilling the control requirements of extreme situations. The Centre distributes tasks and is responsible, in the case of large-scale accidents, for the coordination of the organisations involved (police, fire brigade and medical help), until all
units have arrived at the scene of the accident. At that point, the fire brigade
takes over responsibility for coordination. The main goal is to assign and manage
resources as fast as possible, as well as to assess the particular conditions of each
emergency. The sequence of steps to build the cooperative model is shown in the
following subsections.

5.1 Organisation Group
The first step in the method is to specify groups by means of UML state-
charts, as shown in Figure 1(a). This is based on identifying the related roles,
there is one state for each role. The concept of role allows us to state the dy-
amic connections between actors and tasks. Thus, the basic structure of the
organisation is described: actors play the roles Operator, Assistant Operator
and ResourceResponsible. In this case, capabilities (e.g. guard [operator?])
initially determine which role is played by each actor, depending on his/her
professional category, specialisation and/or skills. It also specifies (e.g. guard
[FreeOperators=0]) under which constraints dynamic behaviour changes must
be produced, sometimes as a result of interactions between members of the group.

\begin{figure}[ht]
\centering
\includegraphics[width=0.5\textwidth]{fig1.png}
\caption{(a) Group definition (b) Role definitions}
\end{figure}

5.2 Role Definition
Actors' knowledge of the system is determined by the functionality of the latter.
In Fig. 1(b) the tasks that can or must be performed are identified. Thus, each
role is actually a UML composed state including a submachine for each task that can/must be performed. Here, the defined roles collaborate on the single task ResolveEmergency. For the role AssistantOperator, the task being performed can be interrupted (section interruptible-tasks) if there is a new emergency. In this situation, the actor will behave as Operator in order to respond to this new emergency.

5.3 Task Definition

The next step describes by means of UML activity diagrams the subactivities/actions needed to carry out each task (Fig. 2). By means of sequential (arrows) and concurrent (thick bars) constructions, temporal-ordered constraints of subactivities are specified. Diamonds may also be used to specify decision points involved in certain strategies during task performance. For each subactivity or operation, the task definition includes specifying those responsible and the optional roles needed to accomplish it. Optional roles are shown between brackets and the symbol ']' specifies an inclusive-or relationship. This role specification is an extension to UML swimlanes for activity diagrams.

Fig. 2. Task Definition
5.4 Specification of Interactions Between Actors

The above task definition includes several tasks to be carried out by means of interaction protocols. For instance, the subactivity InterviewCaller specifies the type of protocol Request-Reply that should be used to accomplish this, as well as the participants involved: Operator asks and Caller answers. On the other hand, the activity DecideRescueUnits specifies a conversational protocol, i.e. any participant can take part in this activity in any order and with the same degree of responsibility.

6 Semantics for the Cooperative Model

This section states the foundation of a specific semantics for the cooperative model. The idea is to be able to automate the behavioural analysis for the cooperative system by making explicit concurrency, non-determinism, synchronization and resource sharing. CPN can be subjected to various Petri Nets analysis techniques which aid in the validation of UML behavioural specification.

6.1 Translation schemes from UML to CPN

In the following subsections, we discuss the key points in formally defining notation semantics according to the application domain. This is obtained by translating elements of the cooperative model to CPN components, as shown in Fig. 3, which is the corresponding CPN for the above modelling example. The subnet ResolveEmergency in Fig. 3(b) is for the transition with the same name in Fig. 3(a).

**State/Activity Mapping** There exist two general alternatives to apply transformations from the cooperative model to the CPN model:

1. Identify subactivities with places in the CPN, allowing them to be interrupted (if necessary) in a direct way. In a cooperative system, there are both atomic and non-atomic activities, subactivities and actions respectively. Subactivities can be interrupted (if specified), such that the actors playing them may leave and return to the subactivity later. For example, the actor playing the role AssistantOperator in Fig 2 can abandon the subactivity DecideRescue during its performance.

2. Conversely, the other possibility is to identify subactivities with transitions. At first glance, no activity could be interrupted since transition firing is considered instantaneous in Petri Nets.

We have chosen the second alternative, for several reasons. First, mapping subactivities into places poses the following problem: if a place represents a subactivity state, when the actor returns the subactivity will start again. Thus, to represent the leaving point where a subactivity continues would be impossible.
Fig. 3. CPN for the Emergency Coordination Centre
Secondly, the common Petri Net hierarchical modelling technique is by means of substitution transitions, and therefore if a transition represents a subactivity, there always remains the possibility of decomposing it into various actions (other transitions) and resting points (places) that enable interruptions and returns. This is also the ideal solution to the previous problem. Thirdly, modelling subactivities by transitions allows us to model data flow in the places of the subactivity flow more clearly (e.g. place of type D in Fig. 3(b)).

**Actors and Resources** As shown in the CPN model in Fig 3(a), a role is considered equivalent to a type of resource, which is represented in a Petri Net as a place and a specific type associated (type A). Hence, there would be one token in the place for each actor playing this role. Each one of these places is labelled with the corresponding role name. For instance, there is a place labelled Operator with a token type A and, initially, two free value tokens (inscription 2 free). For the sake of simplicity, the net does not include the role Responsible, and hence does not include the transition to take the initial decision corresponding to the diamond in Fig 1(a). The arcs from role places to the subactivity transitions and vice versa, corresponding to the roles performing activities in Fig. 2(b), have also been omitted.

**Guards** In CPN, transitions can have guards associated, which implies that the UML guards have a direct translation to CPN models, i.e. while the guard condition is not satisfied the transition cannot occur. Guards expressed in natural language must be interpreted. This is done in part (Fig. 3(b)) for the two consecutive decision points in Fig. 2. These are merged in the CPN, producing as many transitions as the total number of outgoing paths. The first condition (approved or rejected) in the guard is translated on the basis of a specific piece of information outgoing from the activity DecideRescue (type D token). With respect to the second condition (OperatorAlone or OperatorNotAlone), this is not interpreted in Fig. 3(b) in order to simplify type A as much as possible and to avoid additional arcs.

**Events** Fig. 3 includes two types of event: one internal event (place Assistance-Call) and two external ones (EmergencyCall and Brigade Took Responsibility places). Irrespective of the type of event, these are explicit signals translated into extra places receiving tokens from internal actions (Ask Assistance) or the environment (Generate). These places send tokens to subactivities/actions (transitions) that require them. Thus, the net firing rule ensures that the transition receiving tokens cannot occur before the event has happened.

### 6.2 Validation and Property Verification

CPNs allow us to validate and evaluate the usability of a system by performing automatic and/or guided executions. These simulation techniques can also carry
out performance analysis by calculating transaction throughputs, etc. Moreover, by applying other analysis techniques it is possible to verify static and dynamic properties in order to provide the complement to the simulation. Some of these properties are that:

- There are no activities in the system that cannot be realised (dead transitions). If initially dead transitions exist, then the system was bad designed.
- It is always possible to return to a subactivity if we wish (liveness). For instance, this might allow us to rectify previous mistakes.
- It is always possible to return to a state before (home properties). For instance, to compare the results of applying different strategies to solve the same problem.
- The system may stop before completion (deadlock). Thus, a work might never be finished, or it might be necessary to allocate more human resources to perform it.
- Certain tokens are never destroyed (conservation). Hence, resources are maintained in the system. This should be true for actors.

7 Conclusions

UML, with the nine notations embodied, is suggested as a general and standard notation for the analysis, design and development of object-oriented software systems. UML semantics is intended for the latter field; it is not a formal description. Neither does it define a standard process for its application, of which we are taking advantage, but is intended to be useful with iterative development processes.

In practice, the modelling power of UML is demonstrated by applying it to diverse types of systems. New approaches to address the analysis and design of cooperative systems must advance towards how to study and obtain interesting properties for systems. We argue that concepts should be correctly represented, at appropriate levels and that clear semantic links between them should be provided for useful integration, thereby resulting in a more powerful, useful and flexible system from all points of view.

Our main aim is to obtain the benefits of using an standard notation as UML:

- it allows us to model open, reactive systems [4], i.e., the main features of cooperative systems,
- there exist several UML-based tools to design and generate software (RationalRose, ArgoUML, ...), and
- new UML-based approaches are arising to model user interfaces [24, 28].

On the other hand, it is necessary the application of formal methods to the engineering of cooperative systems in order to build CSCW systems correctly. For this purpose, CPNs have a graphical representation and well-defined semantics, which allows compact and manageable representations and therefore one more powerful analysis than that of UML.
References


46
Adaptive User Interface for Mobile Devices*

Nikola Mitrović¹ and Eduardo Mena²

¹ IIS Department, University of Zaragoza, Maria de Luna 3, 50018 Zaragoza, Spain
mitrovic@prometeo.cps.unizar.es
² IIS Department, University of Zaragoza, Maria de Luna 3, 50018 Zaragoza, Spain
emena@posta.unizar.es
http://www.cps.unizar.es/˜mena

Abstract. Adapting a graphical user interface (GUI) to a variety of resources with different capabilities is one of the most interesting questions of today's mobile computation. The GUI constructed for one application should be usable on different interactive devices, e.g. WebTV terminals, WAP phones or Java-enabled devices. In this paper, we discuss existing solutions and present a solution based on mobile agents. Mobile agents construct their GUI using third-party eXtensible User interface Language (XUL), jXUL middleware and XSL transformations. Mobile agents move to host computers and then build their GUI or act as a proxy to devices without sufficient processing capabilities (e.g., WAP devices). The result is an adaptable GUI platform that can be run on multiple devices without modifications, supporting different resources and architectures. We show the application of this approach by implementing a mobile currency converter and survey.

1 Introduction

Constructing graphical user interfaces (GUIs) in mobile computing area faces many challenges. Main problems are raised from the fact that various target devices have different processing powers, GUI organization and capabilities.

Solutions in this area mainly focus on web applications with client-server architecture, creating specialized and centralized services that transform one type of user interface in another. Some solutions propose creating separate GUI solutions for each device type, that are later dispatched according to the request type (or request origin). Some authors propose XML-described user interfaces that could be later presented as Java AWT [1] or Swing [1], or that can be transformed with XSLT [2].

The idea of this work is to transparently adapt graphical user interface by using mobile agent systems. Agents are highly mobile, and are often hosted by platforms that support different models of user interface or have different processing capabilities. Agents are autonomous, and can handle network errors

* This work was supported by the DGA project P084/2001.
(unreachable hosts, etc.) autonomously; also, they can move to the target device instead of target device requesting service from a server. Agents can be sent to a home computer supporting Java and Swing. On the other side, an agent can play the role of a proxy server for a wireless device, such as mobile telephone or a Web terminal, and in that case it should produce WML [4] or HTML [3], respectively. In contrast, solutions not using mobile agents are client-server systems or use middleware programs that are installed on each user device. Therefore, new updates lead to reinstalling client programs on every user device, which does not happen when using mobile agents (only the mobile agent needs to be updated).

Our prototype adapts user interface using mobile agents [8] that process a user interface definition described in a language called Extensible User-Interface Language (XUL) [3], [13]. This interface definition is later adapted using XSL transformations to other notations (HTML, WML, etc). The XUL interpretation on Java-enabled platforms is interpreted by jXUL platform. The jXUL is a third-party middleware that renders XUL using standard Swing interface. Agents automatically adapt the interface definition to the clients’ interface, making multiple middleware implementations unnecessary.

This approach gives good results when deployment is needed not only as a web application, or only as a desktop application. This approach combines these approaches, and is truly mobile in its nature; agents can autonomously determine what kind of interface should be presented. One of the advantages of mobile agents approach is that the GUI goes mobile and can be constructed in function of autonomous operation of mobile agent. For example, we could have a user interface that is modified depending on information collected from the agent’s trip on the network, and that can be later presented on any type of the device.

The rest of this paper is as follows. Section 2 gives an overview of state of the art and the related work. Section 3 introduces extensible user interface language (XUL) and gives an overview of its possibilities and limitations. Section 4 introduces mobile agent technology. Section 5 we introduce our motivating example and explain bound between mobile agents and GUI. Section 6 describes sample scenario that shows the presented technique. Section 7 concludes the paper and discusses the future work.

2 State of the Art and Related work

Various approaches to adapting user interfaces to various devices are present. Basically the approaches are grouped into two categories: web applications and classic desktop applications. While the first category [11], [14] treats only web content and transformations of web content in order to be usable on other (mostly mobile) devices, the second category treats the problems of universally defining the user interface, so it can be later reproduced by various program implementations [9], [10], [12], [24], [25] (or middlewares) on various platforms.
2.1 Adaptable XML-defined interfaces

Several solutions for defining user interface are present at the moment of writing this article. Without providing details, we mention some approaches: language-based, grammar-based, e.g., BNF, event-based, constraint-based, UAN (User Action Notation, in particular for direct manipulation) and widget-based. However, the XML-based efforts are most interesting for us, since they provide flexibility and easy manipulation. Some of such efforts include XUL [5], the extensible user interface language, UIML (User Interface Mark-up Language) [26], [10] and XIML [32].

Luyten, et al. [24] investigated the possibility of rendering their own XML-like mark-up languages to Java AWT and Swing, or converting the interface definition to other formats using XSL transformations or XPath. However, this approach is focused on creating different middleware (or transformation) for different platforms (that are not Java compatible), and not on transparent modification of user interface. Also, their prototyped solution do not run in a truly mobile environment (mobile agents), and is focused on rendering the user definition files on multiple platforms by using different middlewares. At the time being, this prototype also lack complete language definition.

Other approaches, such the one from Meller, et al. [25], are more focused on defining the universal XML notation that can be used for platform independent interface generation.

2.2 Web applications and adaptable user interfaces

Application servers are mostly oriented on how to transform web contents to various other formats that can be used on mobile devices (cHTML [30], WML [4], etc.). However, different approaches exist.

Microsoft, one of the industry leaders, in its next-generation technology ”.NET” offers Mobile Web Forms [11]. These forms are based on restricted set of components that, to our knowledge, cannot be extended with additional widgets. Each component is intelligent component that transforms its appearance in function of available resources. The controls are highly bound with the .NET family of languages. Unfortunately, Microsoft’s solutions are still available only on the Windows platforms, the number of widgets is limited, and the desktop applications are not taken into the consideration.

Other industry leaders, such as IBM, have slightly different approaches. IBM’s Transcoding Publisher [14] actually transforms web contents to variety of other formats, giving the user possibility of customisation of the transformation parameters. Some interesting features such as JavaScript [7] transformation and automatic image format transformations are included. Users should be able to customize the transformations in order to maximize the quality the output, which can be a significant plus for complex web applications; another good side is IBM’s commitment to Java, therefore multiple platforms are supported. However, the drawback of the approach is ability to transform only web contents, and in a centralized fashion.
Other "traditional" solutions in the web-area also exist, and consist on parsing the request information, and redirecting the petition to the appropriate content [15]. The content is created separately for each device type, and is stored separately. When the user accesses the server with a mobile device, the server will recognize the request type, and will redirect the user to the appropriate content. This solution has a significant overhead, because the content should be created multiple times in order to support different formats. Scalability of this solution can be also questioned.

All these approaches support different level of customisations, but however only web applications. The user interface generation is centralized – on the server.

3 Extensible User-interface Language - XUL

Extensible User interface Language [5], [13] is designed for cross-platform user interface definition. This language is incorporated in Mozilla project [17], acting as a user interface definition language. Being part of Mozilla project, XUL is open and connectable to other Mozilla projects. The format is organized with modern user interface definition in mind, supporting variety of available controls.

XUL lacks the abstraction layer of interface definition, and is restricted to window-based user interface. It is capable of referencing Cascading Style Sheets (CSS) [18] to define the layout of elements. The user actions, property access and functionality can be stored in JavaScript (ECMAscript) [7] files. However, we found XUL as suitable open source solution for our purpose.

A simple XUL window in Fig. 1 could be defined as in Fig. 2.

![Fig. 1. Window to be constructed](image)

From this example, we can see that the interface definition is oriented to modern window-based interfaces. We are referencing a StyleSheet, JavaScript library, and using few labels, textbox and a button within the box tag. The box tag is main form of layout in XUL and is similar to Swing JPanel. This model allows you to divide a window into a series of boxes. Elements inside box will orient themselves horizontally or vertically. By combining a series of boxes,
spacers and elements will flex, and you can control the layout of a window as can be seen in Fig. 1.

4 Mobile Agents and Agent Platforms

A mobile agent [8], [27] is a program that executes autonomously on a set of network hosts on behalf of an individual or organization. The agent visits the network hosts to execute parts of its program and may interact with other agents residing on that host or elsewhere, while working toward a goal. During their lifetime agents travel to different hosts, that can have distinct user interface possibilities. Agents typically possess several (or all) of the following characteristics; they are:

- Goal oriented: they are in charge of achieving a list of goals (agenda).
- Autonomous: they are independent entities that pursue certain objectives, and decide how and when to achieve them.
- Communicative/collaborative: to achieve their goal they can cooperate.
- Adaptive/learning: agents "learn" from their experience and modify their behavior respectively.
- Persistent: agent's state (should) persist until all the goals are achieved.
- Reactive: they react to their environment which also could change their behavior.
- They can stop their own execution, travel to another host and resume it once there.

They do not, by themselves, constitute a complete application. Instead, they form one by working in conjunction with an agent host and other agents. Many agents are meant to be used as intelligent electronic gophers – automated errand boys. Tell them what you want them to do – search the Internet for information on a topic, or assemble and order a computer according to your desired specifications – and they will do it and let you know when they have finished. Mobile
Agent Systems (MAS) are the middleware that allows creating and executing mobile agents. For this project, we choose Grasshopper [19] as the most intuitive and stable mobile agent platform, which supports standards such as FIPA [20], CORBA [21] and RMI [22]. In addition, the Grasshopper's feature Webhopper [19] that enables mobile agents for web is a significant plus comparing with other platforms, like Voyager and Aglets [31].

5 Using XUL with Mobile Agents in Multiple Platforms

The idea of this work was to use XUL together with the mobile agent paradigm [8], and to make a prototype that adapts XUL for hosting platform or for remote devices (e.g. a wireless device). By achieving this, one will have a truly mobile user interface that adapts to the platform on the fly.

5.1 A Motivating Example

We present these sample applications, meant to demonstrate the possible every day uses of a mobile agent that adapts its user interface to multiple devices.

The first example is a currency converter application that can be accessed from every point on the network. This application converts among three currencies (Euro, US Dollar, British Pound). We want this application to be accessible from various different devices (Java, WAP phone, web terminal). In all of these cases, the same application should be started, and the same (or equivalent) user interface should be used in order to reduce costs of application development.

The second example is a survey application. It should make a poll of the converter application users, calculate the stats, and return the data to the software company that built the converter application. Similarly to the converter application, we expect users that are taking a survey to have all sorts of devices, different connection types, and possibly problems with network coverage/links. The application ask users to rate the converter application with three possible answers (good, normal, bad), and calculate stats on the answers. All the answers are persisted so the statistics are made on all-times data. In case of loss of network coverage or broken network links, application should re-into connection or try alternate route to the next host without prompting user.

In Section 6 we describe in detail these sample applications.

5.2 XUL implementation - jXUL

jXUL [6] is a Open Source project that interprets XUL definition and renders it to Java Swing interface, similarly to [16]. Plans for jXUL are very ambitious, aiming to support very complex controls in future releases. Other open source projects that aim to rendering XUL with Java or to DHTML exist [28]; unfortunately, at the time of writing this paper, none of these has any public prototype available.
However, available jXUL implementation lacks basic functionality, such as assigning and getting values from components or ability to connect outer classes to the JavaScript engine that runs within jXUL. Therefore, we put significant effort into redesigning the existing components to support basic functionality, and extended the JavaScript engine functionality by adding connector classes that can be externally connected to jXUL middleware. Unfortunately, jXUL is built to render only to Java Swing, and not to other types of interfaces, so we had to develop for our prototype XSL transformations that transform XUL files to HTML and WML files.

5.3 Putting it all together
The prototype built customizes mobile agents in such manner that programming a system that has adaptable user interface is almost completely transparent; programmers have only to extend the required class, connect the classes and to create interface definition files. This approach combines adaptivity with respect to allocating system functionality and adaptivity with respect to interface layout. However, level of plasticity [33] is basic and will be improved in the future work. The base classes will convert the XUL files to appropriate format and handle the communication.

Thus, we created a simple currency converter application that uses a few basic controls: labels, text boxes, radio buttons and classic buttons. In order to construct this application we need some XUL files (one for each window). The sample XUL file that we created for the sample currency converter is shown in Fig. 3.

After constructing the user interface definition, the worker class should be created. This class should carry all procedures that handle interface events, but the computation is not limited to this class. Because of Grasshopper limitations, we had to create this class as a connector class, to be used from jXUL’s JavaScript, and therefore to be accessible from the user interface.

The structure of the sample method that we implemented for currency converter agent is shown in Fig. 4.

Code in Fig. 4 is used for any interpretation of XUL files; no modifications for any platforms should be made. As we can see, this method takes three parameters that are passed from the GUI and then process the request. While processing, the window is closed, and when the result is calculated it is opened again.

What we have created is a mobile agent that transforms itself into three forms: Java Swing, HTML or a WML application, depending on the user device capabilities. Also, the agent is acting both as a server and as an application at the same time - if the originator cannot accept mobile agents (e.g. wireless devices), the agent will act as a content server to that device. However, if mobile agents are accepted, the agent will act as standard application.

6 A sample scenario: mobile calculator and survey
For our prototype we have set up the network consisting of five network nodes:
Fig. 3. XUL definition used in currency converter application

- The DesktopNode is a Java-enabled fixed computer that can render Swing; it is able to host mobile agents.
- The WebNode is a network terminal that can render only HTML. This node cannot host mobile agents.
- The WapNode is a mobile phone with Wap browser that can render WML; this node has a wireless connection to the network and cannot host mobile agents.
- The LaptopNode is a wireless laptop; it is Java-enabled, that can host mobile agents.
- The CorporateNode node is server computer, that hosts agents and can render Java Swing. Provides users with our sample applications implemented
as mobile agents. This node also serves as a server for WebNode and WapNode, since these nodes were assumed not to have possibility of running Java. Of course, this is the worst-case scenario, since there is emerging number of mobile devices that run Java.

6.1 Currency converter application

The objective of the currency converter application (described in Section 5.1) was to demonstrate the adaptive interface concept. This application adapts its appearance to the originator of the request. As we can see in the Fig. 6, if DesktopNode or LaptopNode invoke the application, the mobile agent (application) moves there and then it will render the XUL files as Swing.

However, if the WapNode or WebNode invokes the application, a different action will occur. Since these nodes were not to host agents the CorporateNode acting as server will process their requests.

As we can see in the Fig. 5, we have a mixed architecture. Clients that cannot support mobile agents (WebNode, WapNode) are using CorporateNode as server for their petitions, and therefore client-server architecture is present. However, our currency converter agent travels from CorporateNode to DesktopNode and LaptopNode, using mobile agent architecture.

In Fig. 6, Fig. 7 and Fig. 8, we show how the application looks if invoked from those three platforms. From the Fig. 6 and Fig. 7 we can see that the Swing and HTML outputs are not exactly the same. HTML output for this example could be improved by using tables, but we decided to use simple XSL transformations.

Fig. 8 shows the WML output on the M3Gate WAP browser simulator [29]. As we can see, the output differs significantly from the HTML and Swing outputs as device capabilities and rendering language are different and more limited. For
Fig. 5. Network topology and mobile agent trajectories for the currency converter application

![Currency Converter](image)

**Fig. 6.** Currency converter agent rendered with Java Swing

example, as seen in Fig. 8, radio groups are initially presented as widgets (Fig. 8, on the left) that are later expanded to a full-screen selection (Fig. 8, on the right).

### 6.2 Survey application

This application (explained in Section 5.1) shows benefits from mobile agent computing. Agents are autonomous, adaptive, learning, and mobile; survey application demonstrates these properties. The survey agent travels through the network, visiting the hosts that used our currency converter application. When it reaches the destination host, it transforms its appearance in the suitable form
to ask users for their opinion about the currency converter application. If the network host is unavailable or unreachable, the agent will autonomously decide what to do next. It could wait for host to be available, or continue with the other hosts and return later to the unavailable hosts. Statistics on collected data is calculated.

Fig. 9 shows the network topology that we established for this example. In this Figure we can see that survey application travels from CorporateNode to Desktop-Node and to the LaptopNode. Since the LaptopNode has wireless connection to the network, this link can be broken, and the agent will decide how to reach this node without reporting an error. WebNode and WapNode as we discussed have no processing power, therefore they are served from the CorporateNode. When these hosts complete the survey, the survey application returns to the CorporateNode to deliver results and statistics of the poll.
Fig. 9. Network topology and mobile agent trajectories for the survey application

In Fig. 10 and 11 we can see the appearance of the survey application, rendered for Swing and WML clients. The HTML output of this application is very similar to Swing output.

We can see that the agent is not just persisting the survey data, but in fact is calculating statistics based on current data. This distributes the processing among client nodes. There is an open possibility of taking special action depending on survey results. For example, survey agent could return home when it reaches 100 surveys.

Fig. 10. Survey question, rendered as Swing and WML

Notice that the survey application is a "push" service. The user does not have to request a survey from some central host - agent will visit the client by its own initiative. This kind of service is possible when the user platform supports mobile agents. The users of currency converter application do not need to know about the existence of a survey about the converter application. As we can see from
the Fig. 9, when the survey is launched, the agent will visit their computer and will pop-up the survey.

Another example application for our approach would be an automated software update agent that could visit hosts, check present software versions and advise users on possible updates.

7 Conclusions and Future work

In this paper we have presented an autonomous and mobile system, based on mobile agents, that transparently adapts GUI to the users. The main features of this approach are:

- We use and extend third-party middleware jXUL to render XUL to Swing.
- We have made simple XSL transformations in order to convert XUL to other mark-up languages, as HTML and WML.
- A Mobile GUI agent was prototyped, that transparently converts XUL-defined user interface to Java, Web and WAP devices.
- Our approach could be used for other mark-up languages similar to XUL.

This system is stable, gives required functionality, and because of it is mobile agent-based, it is very interested for its application to mobile environment.

However, this work has some limitations. Some platforms do not support elements that can occur in the user interface definition, such as image or sound. Design considerations should take place, and this is the area that should be investigated as a continuation of this work. Thus, our future work will be focused on:

- Transformations for more different outputs
- More efficient transformations, with back-end resource definitions
- Direct binding of XUL to Java without using JavaScript
- Data marshalling using XML metadata
References

31. The Mobile Agent List, University of Stuttgart, http://mole.informatik.uni-stuttgart.de/mail/mal.html
32. XIML (eXtensible Interface Markup Language), http://www.ximl.org/
Migratable User Interface Descriptions in Component-Based Development

Kris Luyten, Chris Vandervelpen, Karin Coninx
Expertise Centre for Digital Media
Limburgs Universitair Centrum
Wetenschapspark 2
B-3590 Diepenbeek-Belgium
{kris.luyten, chris.vandervelpen, karin.coninx}@luc.ac.be

Abstract. In this paper we describe how a component-based approach can be combined with a user interface (UI) description language to get more flexible and adaptable UIs for embedded systems and mobile computing devices. We envision a new approach for building adaptable user interfaces for embedded systems, which can migrate from one device to another. Adaptability to the device constraints is especially important for adding reusability and extensibility to UIs for embedded systems: this way they are ready to keep pace with new technologies.

1 Introduction

The market of embedded systems and mobile computing devices is a fast evolving market. New technologies are introduced at a very high rate. One of the consequences of this evolution is the constant reinvention of user interfaces (UIs) for these devices. They lack the adaptability and flexibility to be deployed for new devices (possibly using new interaction techniques) without reprogramming them. One of the results of the SEESCOA\footnote{Software Engineering for Embedded Systems using a Component-Oriented Approach, http://www.cs.kuleuven.ac.be/cwis/research/distrinet/projects/SEESCOA/} \cite{13} project is a common software platform, using components for embedded systems on a Java Virtual Machine. Using this specific component-based approach for embedded systems, we can develop a framework for UIs adapting to the environment and device specific constraints as well as encourage reuse. The SEESCOA method is a component-based development approach combined with ideas of contract-based specification for software objects.

This paper presents our ongoing research on the possibility of creating a framework that will allow for runtime migratable UIs, which are independent of the target software platform, the target device and the interaction modalities. These UIs are merely considered as a presentation of a single service or of more functionally grouped services. We try to extend the work presented in \cite{3, 14, 10} which all focus on how to abstract a UI for a platform- and device-independent
usage. Like work presented in [9, 6, 2], we also use markup languages to describe UIs. However, our work goes a step further by allowing runtime generation of UIs using a markup language. These ideas are combined with a component-based approach allowing the designer to design UIs for particular components, which can be merged automatically at a later stage. This enables UI designers to concentrate on what is important for multi-device UIs: how to present the UI in a structured and logical manner. Unlike approaches like described in [10], we try to develop a truly distributed component-based approach, without relying on a client-server architecture.

Throughout the text we will use an example case study: a small camera surveillance system using 4 cameras. Each camera will be represented by a component. It will be possible to combine the four cameras by using a Mosaic component. This should make it possible to observe four cameras at the same time. Each camera has its own properties: some cameras can zoom in and out, other also allow to change the framerate,

The next section, section 2, takes a look at how UIs for embedded systems or mobile computing devices can be described with a UI description language. An overview of related work is provided. Continuing with section 3, we show how these descriptions can be combined with software components in general, and SEESCOA components in particular. The case-study is presented in more detail to show the results of the approach proposed in this paper. In section 4, we consider how using markup languages and a component-based approach contributes to flexibility, adaptability and migratability of UIs. In particular attention is given to automatic layout management and multi-modal rendering possibilities. Finally conclusions with regard to the current work and possible extensions are formulated in section 6.

2 Describing User Interfaces For Embedded Systems

2.1 Abstracting the User Interface

When designing UIs for embedded systems, we should not take a widget-based approach, but an interaction- or task-based approach. We should be interested in how a user can interact with the offered service and how this can be instantiated afterwards using a concrete widget set. This kind of approach is thoroughly examined in [11] and is important in particular for embedded devices. Too much time is spent reinventing UIs for accessing the same services as technology evolves. One of the major enhancements we envision is the separation of UI design and low-level programming. Until now, embedded systems programmers have a dual task: implementing the actual embedded system and designing and implementing the UI for this system. The main reason for this way of working is the required technical knowledge and background of the system to provide a UI for it. Therefore we use a markup language to describe the UI for embedded systems and mobile computing devices.
2.2 An XML-Based User Interface description

To describe a UI on a sufficiently abstract level the eXtensible Markup Language (XML)[5] is used. Listing 1.1 provides an example of how a UI can be described in XML. There are already several propositions and real world examples of the usage of XML to describe UIs: [10, 1]. A list of advantages is given in [8]. One of the major advantages is that XML does not force any level of abstraction, so this level can be adapted to the requirements of the situation. Note that an XML document can be presented as a tree which turns out to be a great advantage in our approach. There are other approaches for describing User Interfaces, but we believe that an XML-based description offers the best solution in our component-based approach because of its heavy reliance on hierarchical structures.

Listing 1.1. An example XML listing for a camera

```xml
<ui>
  <title>Login</title>
  <group name="videopanel">
    <interactor>
      <video name="video">
        <text>Camera 2 video stream</text>
        <mediasource>http://twiki.luc.ac.be/camera:8888</mediasource>
      </video>
    </interactor>
    <interactor>
      <range name="zoomrange">
        <text>Zoom</text>
        <min>-100</min>
        <max>100</max>
        <start>0</start>
        <tick>25</tick>
        <action>
          <func service="Mosaic.camera2">setZoom</func>
          <param name="zoomrange/>
        </action>
      </range>
    </interactor>
    <interactor>
      <range name="focusrange">
        <text>Focus</text>
        ...
        <action>
          <func service="Mosaic.camera2">setFocus</func>
          <param name="focusrange/>
        </action>
      </range>
    </interactor>
    <interactor>
      <button name="snapshot">
        <text>Take snapshot</text>
      </button>
    </interactor>
  </group>
</ui>
```
The example listing (listing 1.1) is *not* simplified: the UI description is meant to be human-readable and machine-processable at the same time. The description allows human users to specify the UI on a high level.

On the other hand, the structured and hierarchical approach by using XML as a notational language to describe the UI allows machines to process and use these descriptions without human intervention. Our notation uses a range of tags that are easy to read and understand for humans. In the current stage, a stable Document Type Definition or XML Schema is not available because we do not consider our specification to be complete. Nevertheless care has been taken to introduce no ambiguities in the specification and to enable easy migration to other specification languages, in case a certain XML-based notation for describing UIs will evolve into a standard.

The following interactors are currently supported by the system: range interactors, single and multiple choice interactors, a text interactor, push interactors (e.g. a button) and a canvas output interactor (e.g. a video stream). These can be composed to represent a new interactor with combined functionality. The available tags are still limited, but a lot of dialog-based UIs can already be implemented using these widgets (e.g. all kinds of web forms). There are two tag types which are of particular importance: **group** tags and **action** tags. The **group** tags allow to group objects which have no meaning when they are separated. An example of this is a “date interactor”: the interactors involved for filling in a date should not be separated (listing 1.2). Groups can be nested: they can be hierarchically structured. This enables us to reuse groups of interactors, and make new composed groups. The **action** tags allow a user to specify which action to fire if the interactor (which is the parent node) is manipulated. The action tag specifies the target (this can be a class name, a server,...) and the functionality that has to be invoked from this target. It is also possible to specify parameters and use the names of the interactors or groups for these parameters. Our system will automatically extract the current content out of the interactor or group (to which these parameter identifiers point) and pass it to the invoked functionality. There is no need to indicate the type for the UI designer, the type checking will be done at runtime. This is advantageous for the level of abstraction, but demands a detailed exception handling algorithm, and allows little or no compile-time or design-time checks. Further implementation may be required to reveal more opportunities to check the validity of the description at design- or compile-time.

Listing 1.2. A date group

```xml
<group name="date">
    <interactor>
        <range name="day">...
</range>
    </interactor>
</group>
```

The example listing (listing 1.1) is *not* simplified: the UI description is meant to be human-readable and machine-processable at the same time. The description allows human users to specify the UI on a high level.

On the other hand, the structured and hierarchical approach by using XML as a notational language to describe the UI allows machines to process and use these descriptions without human intervention. Our notation uses a range of tags that are easy to read and understand for humans. In the current stage, a stable Document Type Definition or XML Schema is not available because we do not consider our specification to be complete. Nevertheless care has been taken to introduce no ambiguities in the specification and to enable easy migration to other specification languages, in case a certain XML-based notation for describing UIs will evolve into a standard.

The following interactors are currently supported by the system: range interactors, single and multiple choice interactors, a text interactor, push interactors (e.g. a button) and a canvas output interactor (e.g. a video stream). These can be composed to represent a new interactor with combined functionality. The available tags are still limited, but a lot of dialog-based UIs can already be implemented using these widgets (e.g. all kinds of web forms). There are two tag types which are of particular importance: **group** tags and **action** tags. The **group** tags allow to group objects which have no meaning when they are separated. An example of this is a “date interactor”: the interactors involved for filling in a date should not be separated (listing 1.2). Groups can be nested: they can be hierarchically structured. This enables us to reuse groups of interactors, and make new composed groups. The **action** tags allow a user to specify which action to fire if the interactor (which is the parent node) is manipulated. The action tag specifies the target (this can be a class name, a server,...) and the functionality that has to be invoked from this target. It is also possible to specify parameters and use the names of the interactors or groups for these parameters. Our system will automatically extract the current content out of the interactor or group (to which these parameter identifiers point) and pass it to the invoked functionality. There is no need to indicate the type for the UI designer, the type checking will be done at runtime. This is advantageous for the level of abstraction, but demands a detailed exception handling algorithm, and allows little or no compile-time or design-time checks. Further implementation may be required to reveal more opportunities to check the validity of the description at design- or compile-time.

Listing 1.2. A date group

```xml
<group name="date">
    <interactor>
        <range name="day">...
</range>
    </interactor>
</group>
```
3 User Interface Descriptions and Components

3.1 The SEESCOA Component Framework

Within the SEESCOA\textsuperscript{1} project a component framework for embedded systems is being developed. One of our involvements for this project is merging UI design and component-based development for embedded systems. The component system is asynchronous and uses the Java programming language as a common platform. Components communicate by sending asynchronous messages to each other, and not by using traditional synchronous message calls.

A traditional approach, making a static UI as a layer on a service or a data layer, has proven to lack flexibility. We consider components as units that contain logically grouped functionality and data, each living in their own memory space. They should offer an abstract description of how the service or data offered can be presented. Think about components as software units offering a particular service through their interface: their interface is actually a description of their functionality. It is a natural extension to also allow components to describe what they want to offer to a human user.

Each component can provide a description expressed in XML of the functionality it offers. Alternatively, they also could express in which way they could be interacted with. This is not true for all components of course (some just offer basic functionality on a lower level for other components), so only the components directly interested in human interaction should provide an abstract UI description. When building applications out of components a UI is automatically built: each component has its UI in the form of an XML description. These XML descriptions can all be seen as subtrees of the final, composed UI description. I.e. the UI will be automatically composed by connecting the UI descriptions of the components in to a bigger UI description. Figure 3 shows how this works using a small example: the Camera Mosaic component which is described in more detail the next section (section 3.3). Each component can contain a description of their UI: a description of a Camera can be found in 1.3 and of the Mosaic in 1.4. Figure 3 presents how the descriptions can be combined at runtime to create the UI out of the components.

**Listing 1.3.** UI description of a single camera component

```xml
<group name="camera2">
  <interactor>
    <videowidget name="video">...</videowidget>
  </interactor>
</group>
```

66
<interactor>
  <range name="zoomrange"><action>
    <func service="Surveillance.Controls">setFocus</func>
    <param name="camera2"/>
    <param name="zoomrange"/>
  </action></range>
</interactor>

Listing 1.4. UI description of a Mosaic component

<ui>
  <title>Camera mosaic</title>
  <group name="mosaic">
    <group name="camera1">&CAMERA1</group>
    <group name="camera2">&CAMERA2</group>
    <group name="camera3">&CAMERA3</group>
    <group name="camera4">&CAMERA4</group>
  </group>
</ui>

Notice this approach allows components to migrate and offer their services in other places. The UI integrates smoothly in the new system the component is used on. The component-based approach supports a distributed view on assembling applications out of components and generating their UI: parts of the UI are allowed to migrate together with the functionality the components offer. Finally, the UI description can be submitted to a “renderer” component in the form of an XML document.

3.2 The Rendering Component

As we take a component-based approach for designing UIs for embedded systems, there is one “basic” component: the UI renderer component. This can be compared to a web-browser: a description for an interface can be submitted to the component and it will take care of rendering this description. Nevertheless, there are some differences: the component can receive a description of a UI and render it to different kinds of output devices and widget sets. The state of the UI can be “serialised” back into XML and relocated, which makes the component approach suitable for distributed systems or remote UIs. The SEESCOA
component system takes care of the communication and makes it network transparent. Notice the rendering engine is also embedded in a component, so this component can also have a UI description of its own functionality. To show its UI the rendering component can send its UI description to itself.

There are several possible output formats and for each kind of output a different rendering component can be supplied. For example: there could be rendering components for a PDA (e.g. Palm, see figure 5), for Java Swing (suitable for use on a desktop PC) and a rendering component for speech synthesis. The date group presented in listing 1.2 is rendered using two different rendering components in figure 1: a HTML rendering component and a Java AWT rendering component. The rendering components are “self-contained”: they do not rely on other components and are suitable to migrate individually to a particular system or software platform. Their internal working relies on the same code nevertheless.

3.3 A Case Study: a Camera Surveillance system

To illustrate how components can deliver their own UI description, we developed an example case study in the context of the SEESCOA project: a surveillance system. The example surveillance system consists of 4 cameras, each camera is represented by a component. The system also contains a Mosaic component, combining the controls for each camera in a combined control. The Mosaic component communicates with a rendering component which renders a UI to an output device. The setup is presented in figure 2. Notice each camera component has its own UI description (shown in listing 1.1) presented as an XML notation. This is shown by the trees attached to the camera components in figure 2. Each camera may offer different possibilities so they can all have different UI descriptions (The camera component is a component which abstracts the hardware and presents a real surveillance camera).

Because of the possibility to specify hierarchical groups, the Mosaic component can take the four individual controls and add them as subtrees in a new
Fig. 2. Component composition example: a simple camera surveillance system

tree. The Mosaic component only needs to add a new root with 4 groups as the children of the root node. Each control can be attached to a group node (figure 3, the group nodes are coloured gray). The UI description tree produced by the Mosaic component is passed to the rendering component and rendered according to the chosen back-end. This illustrates how combining components to access their provided functionality in one application automatically results in a combined UI of these components. Notice several hierarchies can be mixed if desired: a subtree can be attached to an “open” node on another level in a new tree. This should be done with care: the chances of illogical and unusable generated UIs can increase by doing this. Our current system does not link the several subtrees across hierarchies, so no further support for mixing hierarchies is provided.

Fig. 3. The Mosaic component combining several UI descriptions

Depending on the target device the UI for the Mosaic component will be different. Suppose for example we want to access the Mosaic component using a traditional desktop computer: the rendering component for a desktop PC will load the available Concrete Interaction Objects (CIOs)[14] and try to map the
Abstract Interaction Objects (AIOs) described in the Mosaic UI description on a widget set suitable for a desktop machine: figure 4 shows this. If we want to access the functionality of the Mosaic component using our PDA, the rendering component for a PDA will do the same thing: load the available CIOs and trying to map the AIOs on this set of CIOs. This time the rendering component knows the PDA has limited possibilities, so it adapts the concrete UI to the screen space constraints. Figure 5 shows the results using a PDA (Palm IIIc). The focus of this work was not data communication but runtime UI migration, so we did not spend time investigating effective data communication between devices. The “videostream” for the PDA was actually implemented by sending separate down-scaled images to the device over its infrared connection. Of course, this can be done much more effective using other techniques or means of communication.

3.4 Extending the case study: decomposing tasks

The case study introduced in section 3.3 is a very simple “interaction session” with a single dialog. We consider an interaction session as the interaction which happens to complete a subtask, like “select camera” in figure 6 for example. Most UIs have more than one interaction session: in a dialog-based UI several dialogs are presented after each other. A design method to take this into account is required at this stage. The design method should enable the designer to decompose tasks hierarchically, and link several interaction sessions to each other.
in order to achieve the postulated goal. This method should support a device independent specification of the UI.

To solve this problem, we combine ConcurTaskTree (CTT) [11] with our component-based description method. One of the advantages of the CTT notation is that we can extend it to model context-sensitive user tasks as described in [12]. Characteristics that determine the context of use include the computing platform, the available interaction devices, available screen space,... When one or more of these characteristics change, a reconfiguration of the UI may be required to adapt to the new context of use. [12] proposes a notation to model context-sensitive user tasks. Their solution consists of a CTT task model with roughly the following parts: a non-context-sensitive CTT part and context-sensitive parts depending on some conditions.

The second advantage is the asynchronous nature of the SEESCOA component system: CTT allows to describe temporal relations, and includes concurrent tasks in its notations. A third advantage is the hierarchical structure it offers: our approach also uses an hierarchical notation to describe the UI in a device independent manner.

Now suppose a human guard has access to a security system using a regular workstation or a PDA. Some tasks he can perform on the workstation are not possible on the PDA. Suppose for example that it’s not possible to observe more than 1 camera at the same time on the PDA due to the minor screenspace provided by it. So it depends on the context of use (the device that’s being used in this case) whether the operator can pick just one or multiple cameras to observe at a time. Obviously we can say that this is a context-sensitive task. There are also a couple of non-context-sensitive tasks in this case. The operator must logon to the system before he can pick cameras. Also he can choose to stop observing or pick other cameras to observe. While the guard is observing a
camera (or cameras depending on the context) the other cameras will continue to record their video streams until the guard logs out again. The enhanced CTT tree is shown in figure 6.

![ConcurTaskTree diagram](image)

**Fig. 6.** ConcurTaskTree diagram: checking for burglars with the camera surveillance system in a context-sensitive way

While being a good solution for modeling context-sensitive tasks there are two minor drawbacks to it. The first one is that some subtrees may appear more than once in the model. For example in figure 6 the subtree *Observe Single Camera* appears in the two different contexts of use. [12] solves this by factoring out these subtrees by placing them in the context-insensitive part of the model. The second drawback is that we still have to model every possible context of use: for each device different properties have to be taken into account. In our approach we try to avoid this by using abstract UI descriptions for an interaction task. A CTT description can be saved as an XML document, which allows us to attach our own XML description at the leafs representing an interaction task. These XML descriptions are actually the composed descriptions of the components which are used at that moment. A CTT description becomes a way to describe how we want to interact with a set of given components in a particular stage of the usage of an application. We gain a model-based approach for designing the UI, extending the component-based approach for modelling the software itself. So, instead of using a context-sensitive description as shown in figure 6 we can accomplish the same thing with a non context-sensitive description as shown in figure 7. We recognise that these are just the first steps, and the method has not been tested for a wider range of devices yet. When using totally different ways of interaction (e.g. not dialog-based), we expect we need context-sensitive parts as a consequence of particular other ways to complete the subtasks.
4 Flexible and Adaptable User Interfaces for Embedded Systems

4.1 Realising a concrete UI

To transform the abstract UI description into a concrete one it has to pass several stages of processing in our approach:

1. a mapping stage
2. a specialised layout management stage
3. the rendering stage

The UI description, presented as a tree in memory, is passed to a rendering component, which will initiate the mapping stage: it tries to convert the AIOs into CIOs [3]. For each available widget set, the mapping choices are implemented ad hoc: the current implementation does not support user guidance. This is one of the shortcomings in our approach: we tend to solve this problem in a following iteration. The mapping stage will convert the abstract UI description to a “platform specific” description for one or several specific modalities (using XSLT\(^2\) or an agent component).

Once the system has built a concrete representation structure, the actual screenspace needed for this presentation can be calculated. The mapping stage already involves some calculations of the weighted values of the AIOs, and the corresponding space they may require.

The final step is to show the actual UI: this is done by rendering the CIOs on screen. The widget set used to do the mapping is provided by the target platform and therefore it defines how to represent the CIOs visually.

\(^2\) eXtensible Stylesheet Language Transformations
4.2 System independent User Interfaces

Every time a new device is used as output device, the specific UI renderer component will use the device profile, containing the device constraints and its ad hoc knowledge of the target system. The renderer changes the UI presentation according to the defined limitations.

One of the consequences of adapting the UI to new device constraints is the need for an automatic layout algorithm when GUI rendering is used. When the UI moves to a new output device, the UI should be laid out in a logical way. One approach achieving this is by using layout algorithms found in diagram rendering (like graphs and state-charts). Due to the hierarchical view on the UI, we try to adopt weighting algorithms especially designed for presenting as hierarchical data like presented in [4]. Every leaf is given a weight indicating its complexity (primarily space needs). Recursively every group (i.e. every node that is no leaf) will get the complexity of its children and is added up with a certain constant value in complexity weight. This is a simple attempt to automate the layout algorithm, without taking into account real usability issues which arise when automating this process.

Our architecture allows each subtree of a UI description tree to use a different layout algorithm. For example; we use a layout algorithm that allocates space from left to right in a rectangular space for the first level of subtrees under the root node. The space is allocated according to the weighted complexity of each subtree. On the next level, a more complex layout algorithm is used (like a GridBagLayout in the Java programming language) for each subtree. One of the advantages of this approach is a better support for fragmented UI (several parts of the UI are accessible from several devices), multi-modal UIs and dynamically changing UIs. Currently we are integrating spatial constraints for 2D UI in our system, so the UI designer can indicate how AIOs should be placed in relation to each other [7].

5 Summary of our Current Results

Current results include a rendering component, to which an XML document describing an abstracted UI can be submitted. The renderer maps this description to an actual widget set and tries to adapt the layout so the UI fits on screen. When the screen size becomes too small, the renderer will try to split different parts of the UI and put them behind each other. While doing this, logically grouped elements will not be split up. These grouping operators are specified in the abstract UI description: they group user interactions which logically depend on each other. Examples of tested target widget sets are Java AWT, Swing, kAWT and HTML (web pages).

The SEESCOA Components can be combined in order to make a fully functional application and their UI description can be combined automatically. An example of this was described using the Camera Mosaic application. This enables User Interfaces to become migratable: first of all their description can be
rendered to other output devices and second the UI can accompany the component it represents when it is sent to another system. We have only tested the system with simple UIs, so no conclusions can be made concerning scalability.

6 Conclusions and Future Work

The new component-oriented approach suggested in this paper has several advantages for developers of embedded systems and mobile computing devices in particular. It is

**Flexible**: changing the UI can be done by another renderer component or letting components provide another UI description;

**Reusable**: providing a high level description of the UI related to the functionality a component offers, allows easier reusability of previously designed UIs in contrast with hard-coded UIs;

**Adaptable**: by abstracting the UI, device constraints can be taken into account when rendering the concrete UI.

Besides these advantages we showed how attaching abstract UI descriptions to components helps to compose User Interfaces at runtime without intervention from a programmer. This is especially important when a mobile computing device has to present a new service that it was not aware of. E.g. a PDA comes near to a printer and should be able to present the accessible functionality of this printer. All these advantages make the UIs migratable: they can be easily transported from one device to another, adapting to new environments.

Future work includes adding alternative output rendering components other than a 2D screen renderer like speech output and the implementation of context-sensitive layout algorithms. We acknowledge there is a lack of support for artistic and aesthetic influences in the creation of the UIs employing the approach we presented in this paper. It is our intention to look at alternative interaction methods besides traditional interaction methods. Due to the asynchronous nature of the SEESCOA component system, it is interesting to take time-related HCI patterns into account.

Although the focus is not on the usability of the UIs, introducing these patterns can help us to ensure a minimal usability. For introducing design-time type checks an appropriate editor for this is required. Some checks can be done if a certain amount of information of the application logic is available (the editor should know which arguments can be handled by what kind of functionality). An editor for designing the UI descriptions is not available at this moment.

7 Acknowledgements

Our research is partly funded by the Flemish government and Efro\(^3\). The SEESCOA project IWT 980374 is directly funded by the IWT\(^4\). The Vrije Uni-

---

\(^3\) European Fund for Regional Development

\(^4\) Flemish subsidy organization
versiteit Brussel (Programming Lab) and Katholieke Universiteit Leuven (Distriinet) have created the SEESCOA component system.

References


Task Modelling in Multiple Contexts of Use

Nathalie Souchon, Quentin Limbourg, and Jean Vanderdonckt

Université catholique de Louvain, Institut d’Administration et de Gestion
Place des Doyens, 1 - B-1348 Louvain-la-Neuve, Belgium
{souchon, limbourg, vanderdonckt}@isys.ucl.ac.be

Abstract. The context of use in which users are carrying out their interactive tasks is continuously submitted to an evolution in the user population, the computing platforms used for the tasks, and the physical environment in which users are living. This evolution process raises a need for extending traditional task modelling to support multiple contexts of use simultaneously. To address this problem, this paper first provides a formal notation of a task model that is further refined to support the variation of conditions depending on multiple contexts of use. Key concepts are then introduced to support the task modelling process so as to create a clear frontier between the Context-dependent Task Model and the Context-Independent Task Model. The Context-Partially-Independent Task Model attempts to capture subtasks shared in many contexts of use, but not all. The use of these key concepts enable designers to build a Multi-Context Task Model, notably, by factoring out common parts from Context-dependant Task Models. All these key concepts are equally denoted with the introduced formal notation. In addition, they support designers in adopting the task modelling approach of their choice in multiple contexts of use, which is so far not allowed.

1 Introduction

For many years, user interfaces (UIs) have been developed assuming that the context of use in which they work remains constant over time: the user considered to have little or no variation, interacting with the same computing platform to carry out the same task in a non-changing physical environment. Today, this assumption is no longer satisfied as we observe:

1. A multiplicity of users: not only types of users become more numerous (e.g., more people are willing to interact with computers, tasks previously assigned to other types of users are now devoted to new types, the user population is increasing in diversity), but also types of user are subject to many redefinitions (e.g., users do evolve over time dynamically).

2. A proliferation of computing platforms: existing computing platforms, like the desktop PC, are progressively enhanced with new interaction capabilities while new platforms are emerging, such as cellular phone, Personal Digital Assistant (PDA), Pocket PC, Web Appliance, or dedicated interaction devices.
3. A continuous evolution of the physical environment: the organizational structure may change (thus leading to moving a role from type of user to another), the office location may change (thus resulting in task reallocation), the working circumstances may change (e.g., the user moves with her computing platform from one place to another or the user is moving across computing platforms at the same place).

Existing conditions in which users carry out their interactive tasks are progressively evolving, while new conditions are appearing. Therefore, the capability of task-based UI design (i.e., with a single, all-encompassing task model) to initiate the development process and to ensure user-centered design is questioned. In other words, a task model valid for a single predefined context of use may become no longer valid for multiple, possibly largely different, contexts of use or for variations of the context of use.

The aim of this paper is to address the problem of task modelling in multiple contexts of use by augmenting the capabilities of traditional single-context task modelling to support multiple contexts of use simultaneously. The remainder of this paper is structured as follows: Section 2 situates the scope of this paper and motivates it by highlighting some shortcomings of existing approaches. Section 3 selects a well-established task model that will be subject to a formal definition of its form and properties. Section 4 introduces our detailed definition of the context of use in terms of the previously defined formal notation and provides four key concepts (i.e., the Context-Dependent Task Model, the Context-Independent Task Model, the Context-Partially-Independent Task Model, and the Residual Context-Dependent Task Model) to support an original multi-context task model. Section 5 exemplifies the above concepts on a case study in tele-medicine. Section 6 concludes the paper by reporting on the benefits of the four key concepts supporting the multi-context task modelling and suggests some future work.

2. The development process of multi-context user interfaces

To define the scope of this paper, we rely on the reference framework for plastic UIs introduced by Calvary, Coutaz & Thevenin [3]. It identifies four major levels for producing context-sensitive UIs (Fig. 1).

1. A Concepts and Tasks Model connects a task model, which describes how the goals associated to the task can be fulfilled with the interactive system, and a concepts model, which describes the concepts of interest of the domain of discourse, along with their internal relationships as manipulated by the task.

2. An Abstract UI defines a computing platform-independent rendering of the above concepts and relationships as they are required by the task in terms of working spaces (or presentation units). For instance, working spaces can
be formed by grouping subtasks according to various criteria (e.g., cognitive load, semantic relationships, shared concepts), a navigation scheme between the working spaces can be decided, and Abstract Interaction Objects [22] are selected for each concept.

3. A **Concrete UI** transforms the above platform-independent rendering into a platform-dependent rendering. For instance, it may result in a UI full description in terms of Concrete Interaction Objects [22] and calls to semantic functions belonging to the semantic core.

4. A **Final UI** determines the complete piece of code required to run/execute the UI from the above concrete UI. For instance, it may result in generating C++ code to be compiled at design-time or Java code to be further interpreted at run-time.

In any given context of use (e.g., C1 in Fig. 1), each level is subject to an **iteration** that is, any redefinition or recomposition performed at the same level of abstraction to accommodate with new design options (depicted as a 'T' loop in Fig. 1). Each non-final level is subject to a **reification** that is, any transformation of an abstract level into a more concrete one with the ultimate goal of producing a final UI (depicted as a 'R' top-down arrow in Fig. 1). A second context of use (e.g., C2 in Fig. 1) can be reached at any level of abstraction thanks to a **translation** that is, any transformation of a UI description initially intended for a given context of use into another description of the same level of abstraction, but that is tailored to another context of use. The higher level a translation is applied, the wider and the richer the range of obtainable UIs can be ("the larger the plasticity domain can be" [3]).

Our approach for considering multiple contexts of use focuses on the examination of the translation at the 'Concepts and Tasks Model' level, as represented by the lens in Fig. 1). To express the impact of context variations on the task model, Thavenin [20] introduced two notions:

1. **Decoration** consists of expressing particular configurations of the task model depending on logical conditions representing variations of the context of use. Decoration is further decomposed into three sub-categories [20]: **directive decoration** to express any type of production rule, **corrective decoration** to enable designers to override initial options, and **factorisation decoration** to express any type of exception.

2. **Factorization** consists of expressing common configurations in part or whole of the task model depending on the same logical conditions.

The two notions of decoration and factorization can serve as fundamental atomic operations to compose various approaches to modelling tasks for multiple contexts of use. Among these approaches are the following examples:

1. A 'Specific-target-at-a-time' approach [20]: build one task model for each context of use one after another, combine the resulting separate task models into a comprehensive one by performing factorization and decoration, respectively.
2. A 'Factoring out' approach: build one task model for each context of use and apply factorization to separate common parts from uncommon ones. While easy to conduct, this approach can become tedious (e.g., when contexts of use are numerous), redundant (e.g., when many identical parts are considered due to the activity), unstructured (e.g., when performed with no methodological guidance), error-prone (e.g., when done by hand), or widespread (e.g., when documentation is scattered across the various models).

3. A Minimalistic approach: build one task model containing all parts common to all contexts of use and apply decoration for all uncommon parts resulting from specific contexts of use. While this alternative straightforwardly identifies the common parts by construction, it can be revealed hard to achieve (e.g., when numerous contexts of use or complex tasks are considered). For example, ArtStudio [3, 20] enables designers to start from a task model representing the intersection of all contexts of use and to apply corrective/factoring out decoration for multiple contexts of use. The 'One Model, Many Interfaces' developed by Paternò and Santoro [8] also relies on this approach.

4. A Prototypical approach: build one task model for a context of use considered as representative of most cases (e.g., a more important one, a more frequent one, or a more comprehensive one) and apply corrective/factoring out decoration when appropriate. While offering a natural starting point, this approach does not specify any stopping criteria: when and where decoration should be applied is not obvious.

For example, the polymorphic task hierarchy concept in the Unified UI Design Method [16] starts from a prototypical task model (e.g., delete a file), then apply decoration in the form of alternative task decomposition depending on the the user type (e.g., select file, then select the delete command for
a blind user, select the delete command, select a file, and confirm the command for a motor-impaired user). In this case, the context of use is restricted to the user part.

5. A Maximalistic approach: build the most comprehensive task model with all subtasks for all contexts of use, derive from this maximal model a specific task model for each specific context of use by applying corrective/factoring out decoration. The main advantage of this approach is that the designer always relies on the same maximal model to apply decoration, thus preserving consistency. However, a shortcoming of this approach is that the quality of derived specific task models highly depends on the modelling skills of the designer. In addition, structuring subtasks in the maximal model may become complicated for sophisticated task models.

For example, the designer in xCA [1] first draws up a hierarchy of concepts and subtasks in a project tree. Then, s/he drags some of these elements from the project tree and drops them into a channel tree representing a task model for a particular computing platform (e.g., a WebTV, a cellular phone mini-browser). In this case, the context of use is restricted to one component: the computing platform. Note that UIML [2] also follows a maximalistic approach without any decoration: the model is supposed to work without any variation on all intended computing platforms.

The above examples show how important a simultaneous consideration of multiple contexts of use can be. Equally important are the need for a formal notation and an appropriate way to factoring out parts that are common to different contexts and for differentiating parts that are dissimilar in these contexts. They argue for the need of a sound basis for task modelling in multiple contexts of use.

3 Task Model

3.1 Introduction

A task model describes tasks that users need to perform in order to reach a goal when interacting with a computer-based system. Tasks are typically recursively decomposed into a hierarchy of subtasks. A task model can be represented by a graph structure where:

- Nodes are the different tasks and subtasks a user has to carry out.
- Edges denote either a decomposition relation (a task \(t_i\) is decomposed into several subtasks) or a temporal relation (e.g., a task must be performed before another) between nodes.

Task modeling has been extensively researched for years without any consensus on a formal notation. Various formalisms have been proposed (e.g., formal grammars, transition networks, Petri nets) that cover different types of information for different types of task model. Some are more oriented towards identifying
the activities and their logical decomposition whereas others are including indications of temporal relationships and adding information related to various concepts such as task objects, rules or agents [13].

The selection of ConcurTaskTree (CTT) as a starting task model results from a careful analysis of several task models [12] based on the following rationale:

- CTT has a rich set of formally defined temporal operators (i.e. LOTOS operators) [14], probably the most extensive one.
- CTT is supported by a usable graphical tool (CCTE) which facilitates its dissemination and communication among practitioners.

This section sets the basis of a formal notation of a CTT task model in order to support task modelling for multiple contexts of use.

### 3.2 Definition and Properties

Let us assume that the task model is a directed graph. Let \( \mathcal{RO} \) be the set of relationship operators. \( \mathcal{RO} \) is partitioned into temporal and decomposition relationships. The Task Model \( \mathcal{T}_M \) is defined by a tuple \( \langle \mathcal{TASK}, t_0, T \rangle \) where:

- \( \mathcal{TASK} \) is a finite set, called the set of tasks. \( \mathcal{TASK} = \{t_0, t_1, ..., t_n\} \) where the \( t_i \) are the different tasks and subtasks that have to be carried out.
- \( t_0 \in \mathcal{TASK} \) is the root of the graph, that is to say the initial task.
- \( T \subseteq \mathcal{TASK} \times \mathcal{RO} \times \mathcal{TASK} \) is a set of transitions, which can be noted by the triplet \( \langle t_i, ro_i, t_j \rangle \). As it is a directed graph, \( t_i \) is the source node whereas \( t_j \) is the target node.

For example, the task tree represented in Fig. 2 would be denoted as:

\[
\mathcal{T}_M = \langle \{t_0, t_1, t_2, t_3, t_4\}, t_0, \{(t_0, ro_1, t_1), (t_0, ro_1, t_2), (t_1, ro_2, t_2), \\
(t_1, ro_1, t_3), (t_1, ro_1, t_4), (t_3, ro_3, t_4)\} \rangle
\]

![Fig. 2. Example of Task Graph.](image)

Moreover, some properties can be asserted:

82
- $\forall t_i \in \text{TASK}, \Gamma^+(t_i) = \{t_j \in \text{TASK} | \exists r_{o_i} \in \text{RO} : < t_i, r_{o_i}, t_j > \}$ denotes the set of all the successors of $t_i$.
- $\forall t_i \in \text{TASK}, \Gamma^-(t_i) = \{t_j \in \text{TASK} | \exists r_{o_i} \in \text{RO} : < t_j, r_{o_i}, t_i > \}$ denotes the set of all the predecessors of $t_i$.
- $\forall t_i \in \text{TASK}, \text{father}(t_i) = \text{set of all the predecessors of } t_i \text{ where } r_{o_i} \text{ is a relationship of decomposition in the triplet } < t_j, r_{o_i}, t_i >$.
- $\forall t_i \in \text{TASK}, \text{child}(t_i) = \text{set of all the successors of } t_i \text{ where } r_{o_i} \text{ is a decomposition relationship in the triplet } < t_i, r_{o_i}, t_j >$.
- $\forall t_i \in \text{TASK}, \text{brother}(t_i) = \text{set of all the successors or predecessors of } t_i \text{ where } r_{o_i} \text{ is a temporal relationship in the triplet } < t_i, r_{o_i}, t_j > \text{ or in the triplet } < t_j, r_{o_i}, t_i >$.
- the nodes of $\mathcal{T_M}$ will be organized in layers from the root. We define $L_i$ (the layer of range $i$) as the set of the nodes resulting from applying Deo’s level decomposition algorithm [5]. Moreover, $\forall i \forall j, L_i \subseteq \mathcal{T_M}_j$ can be verified. In the above example, $L_0 = \{t_0\}, L_1 = \{t_1, t_2\}$ and $L_2 = \{t_3, t_4\}$.
- if $\Gamma^-(t_i) = \emptyset$, then $t_i = t_0$: the root denotes the main task.
- if $\text{child}(t_i) = \emptyset$, then $t_i$ is a leaf: a leaf denotes a basic task.
- $\mathcal{T_M}_i \subseteq \mathcal{T_M}_j \iff \forall < t_i, r_{o_i}, t_j > \in \mathcal{T_M}_i \Rightarrow < t_i, r_{o_i}, t_j > \in \mathcal{T_M}_j$: $\mathcal{T_M}_i$ is included $\mathcal{T_M}_j$ iff all the transitions of $\mathcal{T_M}_i$ are included in $\mathcal{T_M}_j$.

For the purpose of this paper, the following hypotheses are stated:

- $\forall t_i, t_j \in \text{TASK}, \exists! r_{o_i} \in \text{RO} \Rightarrow < t_i, r_{o_i}, t_j > : \mathcal{T_M}$ is a 1-graph, that is to say that there exists only one directed edge between two nodes;
- $\mathcal{T_M}$ is not a tree because $\forall t_i, \# \Gamma^-(t_i) \leq 3$: a node can have up to three predecessors: its father, its brother or itself (via iteration relationship);
- $\forall t_i \in \text{TASK}, \# \Gamma^+(t_i) \neq 1$: there must be more than one successor for each task, otherwise this task should not have been decomposed;
- $\forall t_i \in \Gamma^+(t_j): \exists$ one brother$(t_i)$, a corollary of the previous property;
- $\exists! t_i | \Gamma^-(t_i) = \emptyset$: there can be one and only one root for each $\mathcal{T_M}$.

4 Task Model for Multiple Contexts of Use

4.1 Introduction to the Context of Use

Task models attempt to systematically represent the way users achieve a goal when interacting with a system. Some factors largely influence how a user performs tasks to achieve a goal. We group these factors under the term context of use.

The concept of context is extensively investigated in various areas of computer science, leading to no unique definition. Schilit et al [17] define context by three important aspects: where you are, who you are and what resources are nearby. It means that they include the computing environment, the user environment and, finally, the physical environment. Chen and Kotz [4] added to this definition the time context, because the moment the user has to perform a task is also an important and a natural factor.
Some authors consider context to be the user’s context while others consider it to be the applications environment [19]. Petrelli et al [15] define the context as any information that can be used to characterize and interpret the situations in which a user interacts with an application at a certain time.

Dey and Abowd [6] define context to be any information that can be used to characterize the situation of an entity, where an entity can be a person, a place or objects that is considered relevant to the interaction between a user and an application, including the user and the application themselves. (From this definition, almost any information available at the time of interaction can be interpreted as contextual information (e.g., social situation, physiological measurement, and schedules).

Schmidt et al [18] define context as knowledge about the user’s and IT device’s state, including surroundings, situation and location.

We define the context of use as the complete environment in which a task is carried out. Two types of characteristics simultaneously and univoquely determine the context of use [6, 7, 10, 9, 21, 2]:

- Characteristics that are internal to the system containing the application and its UI (e.g., the computing platform, the software/hardware parameters, the interaction devices, the network bandwidth, the latency, or the screen resolution).

- Characteristics that are external to this technical system (e.g., the type of user, her skills and knowledge, her preferences, the sound and light conditions, her geographic position in a building, the stress level, the organization structure, the information channels).

The concept of context of use is partitioned into three models:

1. The User Model (UM) is a finite set \{u_1, u_2, ..., u_n\} where each u_i represents a specific stereotype of user;
2. The Platform Model (PM) is a finite set \{p_1, p_2, ..., p_m\} where each p_i represents any property of the computing platform, such as screen resolution, operating system, or network bandwidth.
3. The Environment Model (EM) is a finite set \{e_1, e_2, ..., e_p\} where each e_i represents a specific configuration of physical conditions (e.g., light or pressure), location-, social and organizational environment (e.g., stress level or social interactions) in which a task is carried out;

A context \(C_i\) is denoted by a tuple <\(u_i, p_i, e_i\)>. A context variation appears when, at least, one element of a context tuple is modified.

A Contextual Task Model (CTM) is defined as a task model associated with a specific context of use. A CTM is denoted by a tuple < TASK, T, \([C_{dim}]\) >, where \([C_{dim}]\) is a matrix of context of use which holds one element: \(C_i\). Several CTMs can be defined for a same application, they are referred as CTM_i.

(From the example of Fig. 2, a CTM can be denoted as follows:

\[
\text{CTM} = < \{t_0, t_1, t_2, t_3, t_4\}, t_0, \{(t_0, r_{o_1}, t_1), (t_0, r_{o_1}, t_2), (t_1, r_{o_2}, t_2), \ldots\}\>
\]
\((t_1, r_{o1}, t_3), (t_1, r_{o1}, t_4),(t_3, r_{o3}, t_4)\), \([C_1]\) >

where \(C_1\) would be for instance: \(<u_1, p_1, e_1>\).

If an application is used in different contexts of use, a matrix \([C]\) would have more than one element of context. Some properties of an application can be asserted from its matrix of context. An application is said to be \textit{mono-user}, respectively \textit{multi-user} when \((U, M) = 1\), respectively \((U, M) > 1\). By analogy an application is said to be \textit{mono/multi-environment} and \textit{mono/multi-platform}.

### 4.2 Context-Independent and Context-Partially-Independent Task Model

In task modelling for multiple context of use, we notice that some tasks or subtasks are carried out the same way in all (or several) different contexts of use. Thus, isolating context-dependent tasks from context-independent ones may be considered useful.

In this section, two new concepts are defined to support this isolation: the Context-Independent Task Model (\(\text{CITM}\)) which is a task model valid for all considered contexts of use and the Context-Partially-Independent Task Model (\(\text{CPTIM}\)) which is a task model valid for a subset of considered contexts of use. Links between different task models will be also considered.

**The Context-Independent Task Model.** A \textit{Context-Independent Task Model} (\(\text{CITM}\)) integrates tasks and transitions that are common to all different contexts of use. The \(\text{CITM}\) is defined by a tuple \(<\text{TASK}, t_0, T, [C_{c itm}]>\), where:

- \(\text{TASK}\) is a finite set of tasks \(\{t_0, t_1, \ldots, t_n\}\) where the \(t_i\) are tasks and subtasks that belong to each \(\text{CTM}\).
- \(t_0 \in \text{TASK}\) is the root of the graph and of each \(\text{CTM}\).
- \(T \subseteq \text{TASK} \times R \times \text{TASK}\) is a set of transitions common to all \(\text{CTMs}\).
- \([C_{c itm}]\) is a matrix containing all the different contexts of use.

\[
[C_{c itm}] = \begin{pmatrix}
C_1 \\
C_2 \\
\vdots \\
C_n \\
\end{pmatrix} = \begin{pmatrix}
u_1 & p_1 & e_1 \\
u_2 & p_2 & e_2 \\
\vdots & \vdots & \vdots \\
u_n & p_n & e_n
\end{pmatrix}
\]

The following conditions must hold:

- \(t_0 \in \text{CITM} \iff t_0 \in \text{CTM}_j \ \forall j:\) in order to find a \(\text{CITM}\), all the different \(\text{CTMs}\) need to have at least the same root. Indeed, two \(\text{CTM}\)s having parts in common but not their root can not be considered to form a \(\text{CITM}\) as their main purpose is different.
- \(\text{CITM} \subseteq \text{CTM}_i \ \forall i\) and \(\forall t_i \in \{\text{CTM} \setminus t_0\} \Rightarrow \exists \text{father}(t_i): a \text{CITM}\) is included in all \(\text{CTMs}\). Moreover, each task in the \(\text{CITM}\) (except the root) must have a father.
- $\# L_i$ of the CITM $\geq$ threshold: the Context-Independent Task Model must have at least threshold layers. Indeed, the number of desired layers in our CITM should be adjustable by the designer. The relevancy of the CITM depends indeed on the granularity of task analysis.
- if $I^+(t_i) = \emptyset$, then $t_i$ is a leaf task or a fork task. A fork task $t_i$ is a task which is the source node of at least one conditional relationship with a task belonging to another task model.

The Context Partially Independent Task Model. A Context-Independent Task Model is made up of tasks that must be carried out in all different contexts of use. But how do we represent a task model valid for only some of those contexts of use? For instance, if we want to develop a multi-platform application for a laptop, a desktop PC and a handheld PC, it is likely that factoring out common tasks between a laptop and a desktop PC would be useful.

A Context-Partially-Independent Task Model (CPTM) integrates tasks that are valid in a subset of considered contexts of use. A CPTM is defined by a tuple $<\text{TASK}, t_0, \mathcal{T}, [C_{prim}]>$, where $[C_{prim}]$ is a matrix containing the different contexts of use $C_i$ with $1 \leq i \leq m$ and $m \geq 2$ and $[C_{prim}] \subseteq [C_{prim}]$.

Moreover, the following conditions must hold:
- $t_0 \in \text{CPTM}_i \iff \exists t_j \in \{\text{CITM or CPTM}_j\} \mid t_j$ is a fork task and $\exists <t_j, ro_i, t_0>$ where $ro_i$ is a conditional relationship
- $\forall j, \text{CPTM}_i \subseteq \text{CTM}_j$ where $[C_{prim}] \subseteq [C_{prim}]$ and $\forall t_i \in \{\text{CPTM} \setminus t_0\} \Rightarrow \exists \text{father}(t_i);
- if $I^+(t_i) = \emptyset$, then $t_i$ is a leaf task or a fork task.

We can now define more precisely a fork task. $t_i$ is a fork task iff $\exists t_j \in \{\text{CITM or CTM}_j or \text{CPTM}_j\} \mid I^-(t_j) = \emptyset$ and $\exists <t_i, ro_i, t_j>$ where $ro_i$ is a conditional relationship between two graphs ($\text{CITM}$ and $\text{CTM}_j$ or ($\text{CITM}$ and $\text{CPTM}_j$) or ($\text{CPTM}_i$ and $\text{CTM}_j$) or ($\text{CPTM}_i$ and $\text{CPTM}_j$).

Remark on CITM and CPTM. Two properties of the general TM have been relaxed in order to obtain a transient representation that shows intersection between CITMs:
- Unique children are allowed. A TM is said to be well-formed iff the minimal number of children for a task is set to two. In other words, it does not make sense to decompose one task into a single task. In a CITM or a CPTM, a task having only one subtask is just the sign that only one subtask is common between the different CITMs from which the CITM (or CPTM) is constructed;
- Isolated brothers are allowed. Each task of a well-formed TM has to be related at least with one of his brother. In a CITM (or a CPTM), only common transitions between CITMs are shown. As temporal relations between two brother tasks can vary from one context to another, it is admitted that two brother tasks may share no temporal relationship with each other in a CITM (or a CPTM).
4.3 The Multi-Context Task Model

The Multi-Context Task Model (MCTM) represents all possible variations of a task model for a given application. The MCTM components are presented in Fig. 3. A MCTM is the union of identified CITM, CPITMs and residual parts of CITM. All components are linked with conditional relations. The residual part of CITMs represents parts that could not be factored out in a CITM or CPITMs.

A residual CITM for a context Ci is defined as the set of ti ∈ TASK and <tj,rok,ti > ∈ T, such that

\[ \forall i, \forall j, \forall k, \forall t, t_i and <t_j,ro_k,t_i > \in CITM \setminus (CITM \cup \bigcup_i (CPITM)) \]

where Ci ∈ [C ∪ m]. A residual CITM can be a well-formed subgraph, a single task or a single transition.

![Diagram](image)

**Fig. 3. Multi-Context Task Model concepts.**

To relate the different components of the MCTM, a conditional expression is introduced. This condition relates a CITM to a CPITM or a residual CITM; a CPITM to another CPITM or a residual CITM. A condition has the form X/p, where X specifies the contexts of use for which a subgraph is valid and p specifies a relationship type (decomposition or temporal) between two tasks situated on different task models (Fig. 4).

To take into account the condition, relationship type of RO must be sub-typed into two types: simple and conditional. Four types are thus obtained: simple decomposition relationship, conditional decomposition relationship, simple temporal relationship and conditional temporal relationship.

5 A Case Study

To illustrate how this can be applied, a case study is introduced that refines a set of scenarios taking place in a medical institution. In all scenarios, a patient is treated in an hospital and a medical staff needs to obtain all the information relative to the patient’s case. Two types of person can access this information:
doctors and nurses. The computing platforms on which they have to carry out their task are various: a desktop PC, a handheld PC and a Cellular Phone. Three different contexts and associated scenarios are defined:

1. **A doctor with a desktop PC (context 1):** A doctor, in her office at the hospital, wants to prepare the visit she has to do to a patient during the afternoon. In order to do this, she logs in into the system and queries a database to access the patient’s medical information (Fig. 5). This information consists in medical files which are composed of text and/or images (e.g., x-ray pictures). She may want to update this information, by adding additional observations on the patient state for instance. Moreover, for severely ill patients, the doctor also wants to monitor real-time information on the patient state (for instance, vital parameters like heart rate, body temperature).

 ![Fig. 5. The CTM for the doctor using a desktop PC.](image)

2. **A nurse with a handheld PC (context 2):** A nurse is working in her service with a handheld PC. She wants to access the medical file of a patient. After logging in, the nurse queries the system to check the medical file of the patient. Considering the size of the screen of the handheld PC, the nurse can only visualize text or images one at a time. The nurse is not allowed to modify the file. Like the doctor the nurse has access to real-time parameter of a patient (Fig. 6).
3. **A doctor with a Cellular Phone (context 3):** At lunch time, the doctor wants to check a patient’s medical file. After logging in into the system, she views the available textual information. As she is particularly worried about this patient, she monitors real-time information (Fig. 7).

![Diagram](image)

**Fig. 6.** The CTM for the nurse using a handheld P.C.

![Diagram](image)

**Fig. 7.** The CTM for the doctor using a Cellular Phone.

The CTM (Fig. 8) is defined as : < { Access and Manage Patient Information, Identification, Insert Login, Insert Password, Manage Patient Information, Manage Patient Medical File, Request Patient File, Monitor Real Time Patient Information, Close Session }, Access and Manage Patient Information, {{Access and Manage Patient Information, dec, Identification}, (Access and Manage Patient Information, dec, Manage Patient Information), (Access and Manage Patient Information, dec, Close Session), (Identification, ≫, Manage Patient Information), (Manage Patient Information, ≫, Close Session), (Identification, dec, Insert Login), (Identification, dec, Insert Password), (Manage Patient Information, dec, Manage Patient Medical File), (Manage Patient Information, dec, Monitor Real Time Patient Information), (Manage Patient Medical File, ≫, Monitor Real Time Patient Information), (Manage Patient Medical File, dec,
Request Patient File) > where

\[
[C_{citm}] = \begin{pmatrix}
\text{Doctor desktopPC} & e_1 \\
\text{Nurse handheldPC} & e_1 \\
\text{Doctor CellularPhone} & e_1
\end{pmatrix}
\]

Fig. 8. The CITM for the three different contexts of use.

A CPITM for context 1 and 2 is (Fig. 9) : < \{ Visualize Patient File, Visualize Pictures, Visualize Text \}, Visualize Patient File, \{ (Visualize Patient File, dec), Visualize Pictures \}, (Visualize Patient File, dec, Visualize Text) \} > where

\[
[C_{cpitm}] = \begin{pmatrix}
\text{Doctor desktopPC} & e_1 \\
\text{Nurse handheldPC} & e_1
\end{pmatrix}
\]

Fig. 9. The CPITM for two contexts of use.

A MCTM can be defined from the different CITM, CPITM and residual CTMs (Fig. 10).

6 Conclusion and Future Work

Thanks to the approach developed in this paper, a UI intended to cover multiple contexts of use can be related to several CTMs depending on the different contexts of use, having small or large differences depending on:
- **tasks**: (i) the task remains the same while the context of use changes; (ii) some tasks (or subtasks) are removed when the context of use changes, because either there is no possibility to perform the removed task in the new context or some tasks appear to be unnecessary or irrelevant for a certain context of use; (iii) task ordering is modified without modification of the tasks themselves. In this case, only the transition differ; (iv) some tasks (or subtasks) are added because a new context requires more tasks to achieve the same goal;

- **relationships**: the temporal relationship between two tasks may differ from one context to another. In the case study presented in Section 5, the two subtasks of the "log in" task are concurrent in one case and sequential in another case.

In order to formally represent those possible variations, several key concepts have been defined, each of them associated with a formal notation:

- The **Context-Dependent Task Model** ($CTM$) associates a task model with a context for which it applies.
- The **Context-Independent Task Model** ($CITM$) represents common parts between all $CTM$ of a same application;
- The **Context-Partially-Independent Task Model** ($CPITM$) represents common parts between some $CTM$ of a same application;
- The **Residual Context-Dependent Task Model** represents parts of the $CTM$ that can not be factored out into a $CITM$ or a $CPITM$;
- The **Multi-Context Task Model** ($MCTM$) is a view that represents conditional relations (depending on the context variation) in the set \(\{CTM \cup \bigcup CPITM \cup \text{Residual } CTM\} \).
The formal notation introduced for a general task model, based on CTT, along with their use for all components of a task model for multiple contexts of use are the original contribution of this paper. They enable designers to adopt any approach discussed in Section 2 in a more formal and structured way. In particular, there is now a clear frontier between task model elements that change or do not change when the context of use is varying. The formal notation also makes it appropriate for inclusion in a tool like CTTE as it provides an internal format that can be manipulated by an automata.

With respect to the reference framework presented in Fig. 1, this work can be situated at the 'Concepts and Task Model’ level and deals with the translation relationship. This study could be extended by defining a formal concept model, analyzing <task, concepts> relationships and considering the influence of context of use variations on these relationships. In particular, it could be worthwhile to represent constraints imposed by a computing platform on the selection of presentation elements (e.g., availability vs. unavailability) or preferred by a user type. Furthermore, there is a need for a formal abstract UI model and concrete UI model that could in their turn be subject to a study on context of use variation. In addition, some patterns should be identified to represent the translation relationship in prototypical context variation. Finally, the notation developed here should be extended to represent run-time adaptation mechanisms, as run-time subtask switching, branching, or migrating.

References


Notational Support for the Design of Augmented Reality Systems

Emmanuel Dubois¹, Paulo Pinheiro da Silva², and Philip Gray¹

¹ Department of Computer Science, University of Glasgow,
17 Lillybank Gardens, Glasgow, G20 6HW, United Kingdom
(emmanuel, pdg)@dcs.gla.ac.uk
² Department of Computer Science, University of Manchester,
Oxford Road, Manchester M13 9PL, United Kingdom
pinheirp@cs.man.ac.uk

Abstract. There is growing interest in augmented reality (AR) as technologies are developed that enable ever smoother integration of computer capabilities into the physical objects that populate the everyday lives of users. However, there is little tool support for the design of AR systems. In this paper, we explore the use of two notations, ASUR and UMLi, to capture design-significant features of AR systems. ASUR is a notation for designing users interactions in AR environments. UMLi is a notation for designing user interfaces of interactive systems. We use each notation to specify the design of an augmented museum gallery. We then compare the two notations in terms of the types of support that they provide and consider how they might be used together.

1 Introduction

The integration of digital (virtual) information and actions with the physical (real) world of users through the use of augmented reality (AR) techniques is becoming a crucial challenge for designers of interactive systems. AR is becoming widely used in a number of domains, including leisure [24], maintenance [8], construction and architecture [25] and surgery [4]. Despite the increasing development of AR systems, neither tools nor methods have been proposed specifically for the design of AR systems. Furthermore, AR systems remain largely ad hoc and exploratory.

In [5], we proposed a classification space for “Mixed Systems”, interactive systems combining physical and digital entities, that identify two kinds of such systems:

- systems that enhance interaction between a user and his/her physical environment by providing additional computer capabilities or data to the physical objects of the environment; these are Augmented Reality systems, AR;
- systems that make use of physical objects to enhance the user’s interaction with a computer; these are Augmented Virtuality systems, AV³.

³ Although we focus in this paper on AR systems, many of the observations we make apply equally to Mixed Systems in general.
The notion of Mixed Reality, introduced by Milgram and Kishino [13], refers to systems that mix digital and physical entities into a single digital representation; for example, in the representation of an interior design, by merging pictures of a real chair within a 3D graphic model of the room [26]. However, this combination of digital and physical properties or entities is achieved on a monitor, so the perception of the physical world is not direct. As opposed to Mixed Reality, augmented reality approaches developed in the HCI community focus on the integration of computational capabilities with physical objects involved in the user's interaction. Users benefit from complementary computer capabilities when interacting with their usual physical tools and objects. These HCI approaches are user- and interaction-centred although they differ in the aspects used to characterise an interaction. Four distinct aspects that may have an influence on the user's interaction with AR systems are identified in the literature:

- Type of data provided to the user [2, 8, 14]: it may be textual, 2D or 3D graphics, gesture, sound, speech or haptic data;
- Potential physical targets of enhancement to combine physical and digital data [11]: users, physical objects and the environment are the three main targets identified;
- Adequacy of the provided data to the task, as well as the location where they are perceivable [1];
- Ability of the system to bridge the gap between physical and digital entities [22].

As this research suggests, developing an AR system is different from developing other sorts of interactive systems. It is often neither obvious nor easy to design and implement appropriate combinations of physical and digital entities, especially in settings where (i) the user may be mobile, (ii) other artifacts may be manipulated in complex ways and (iii) the interaction must be sensitive to complex aspects of the context of use. In this paper we address one aspect of this problem – namely, the description of such systems. Without a means of specifying the features that make an AR system distinctively AR, we cannot communicate nor explore design solutions that benefit from these features.

In this paper we present two complementary notations for capturing design-significant aspects of AR systems:

- ASUR, a graphical notation that can be used to describe, characterise and support the analysis of mixed environments; and
- UMLi, a conservative extension of the Unified Modeling Language (UML) for interactive systems.

Each notation enables a designer to construct models of an AR system. ASUR models identify the key objects and agents in an AR environment along with their physical and informational relationships. UMLi models describe behavioural and structural aspects of the software systems that make up AR systems. In particular, UMLi models include abstract descriptions of user interface presentations implemented by the software systems. These models, and the notations in which they are expressed, offer assistance in several respects:
– Making salient the AR systems-specific characteristics of a design;
– Providing a medium in which to reason about such designs and to communicate them to others;
– Potentially bridging the gulf between different elements of the design of an AR environment, viz., physical and digital elements.

We present the roles that each notation might play in the design of AR environments and their underlying software systems. Further, we discuss the potential benefits of combining both notations to design complete AR systems. The remainder of this paper is organised as follows: First we describe the example used as a case study in this paper. The next two sections describe ASUR and UML, respectively, each presented via its use to characterise the example system. We then compare the modelling capabilities of ASUR and UML for AR systems. Finally we conclude the paper with a brief discussion of future developments.

2 The Mackintosh Project Scenario

As a vehicle for introducing and comparing ASUR and UML, we use an example taken from the City Project, a project developed within the Equator consortium [7]. Based on the work of Charles Rennie Mackintosh, a Glaswegian architect of the early 1900’s, the City Project has been exploring the augmentation of the permanent Charles Rennie Mackintosh Interpretation Centre, a gallery situated in the Lighthouse, an architecture and design centre in Glasgow. The gallery contains exhibits related to Mackintosh’s life and work. The aim of the project is to study the impact of combining multiple media to support visitors’ activities, especially collaborative activities involving users in the real museum interacting with users exploring a digital version of the same museum (“co-visiting”). For the visitor to the real museum, the system being created is aimed at providing visitors with digital information tailored to a visitor’s current context. This information tailoring mainly relies on tracking a visitor’s movement in the museum and the location of the exhibits. Visitor activities are thus embedded with computational capabilities. To do so, the Lighthouse has been equipped with a radio-frequency localisation system that gives the location of the visitors. The augmented gallery in the Lighthouse is only part of the City Project; it is this augmented gallery that will serve as our example and will refer to it hereafter as the Mackintosh Project.

There are a number of services that can be provided by the Mackintosh AR system. In this paper, we consider the FollowSelectedPath service offered to visitors of the Mackintosh Interpretation Centre. The FollowSelectedPath service provides AR support to guide visitors through a pre-defined path of exhibits. A Path is composed of an ordered set of Exhibits. Visitors can select and follow a Path from a set of saved Paths. Each Exhibit is at a Location inside the museum.

4 Terms in italics refer to concepts that will appear in UML class diagrams presented in Figure 2 below.
In addition, the service assumes that the Visitor following a predefined path is already connected to the system and that s/he has already chosen a Path. Under these conditions, the Visitor can get information related to:

- The Path to follow: it consists of textual directions and distances separating the current position of the Visitor from the next Exhibit of the followed Path.
- The Exhibits: once the Visitor reaches the next Exhibit of the path he/she is following, the system provides him/her with information about the Exhibit using specific Media, e.g., Image, Video, that may not be perceivable in the museum (e.g. building material, previous exhibition locations, etc.)

Technically, the Visitor uses a PDA to perceive both kinds of information and to confirm the visit to an Exhibit before getting directions to the next Exhibit on the Path.

It is also important to note that this example does not represent the design of an existing system, nor is it a history of an actual design development. Rather, we have chosen this scenario because it represents a realistic design problem (viz., the design brief is a real one). However, the goals of our example scenario do not correspond to the goals of the City Project and the design alternatives that we present below are our own and do not correspond to any that have been developed during the Project.

3 ASUR Description of the Mackintosh Scenario

3.1 ASUR Background

ASUR (Adaptor, System, User, Real Object) is a notation designed to address the need for a lightweight notation for describing AR systems. Apart from ASUR, we are not aware of any language or notation specifically designed to describe such AR systems.

When designing AR systems, the physicality of the setting becomes crucial. The designer must consider:

- where objects are located in the physical world,
- how they might move,
- their intrinsic physical constraints, such as size, weight, position, etc.
- what can be modified or digitally enhanced,
- how users perceive, manipulate and perhaps carry objects, etc.

Existing notations, such as UML, Concur TaskTrees, etc. are designed to capture properties of computational entities; there is no way to express the potential physical properties of such entities. Therefore, these notations are ill-equipped to capture exactly those aspects that make AR systems special. UML is an example of one way of dealing with this problem, viz., extending an existing notation. ASUR illustrates another way, starting from scratch and bringing together exactly those characteristics that are needed to capture AR system-related design issues. In fact, ASUR combines the characteristics identified in previous interaction-oriented studies of AR systems, as presented in the introduction.
3.2 ASUR Concepts

An ASUR description models an interactive system as a set of four kinds of entities, called components:

- **Component S**: computer System;
- **Component U**: User of the system;
- **Component R**: Real object involved in the task as tool (R_{tool}) or constituting the object of the task (R_{object});
- **Component A**: Adapter for Input (A_{in}) and Adapter for Output (A_{out}) bridge the gap between the computer-provided entities (component S) and the physical world entities, composed of the user (component U) and of the real objects relevant to the task (component R_{object} and R_{tool}).

A relation between two ASUR components may describe a physical or an informational association between them. A relation may stand for an exchange of information (represented by an arrow) or a physical collocation (represented by a double line).

An *interaction facet* consists of an ASUR component and an ASUR relation between this component and the user. Information relevant to the task is provided by the ASUR component to the user, or the user provides information to the component. Arrows connected to the component U in Figure 1 are examples of such interaction facets. Table 1 presents ASUR characteristics for both the component and the relation that constitute an interaction facet.

These characteristics are related to several aspects of the interaction that may constitute the basis for the evaluation of usability properties. A more detailed description of ASUR is presented in [6].

3.3 Illustration of ASUR using the Macintosh Scenario

The diagrammatic representation of the ASUR description is presented in Figure 1. In terms of ASUR, the visitor is the component U, an exhibit is a component R_{object}→task observed by the visitor (R_{object}→U) and the database, represented by V→R_{object}, containing the path and the information related to the exhibits, is included in the component S.

To follow the chosen path, the visitor must be able to perceive the “guidance” information provided by the system. An adaptor for output (A_{out}) is thus required. From this component one relation is connected to the visitor (component U), denoting the transfer of information, related to the path to follow: A_{out}(path)→U. Furthermore, an ASUR relation from the component S to the component A_{out} is required because information provided by the A_{out} component is issued by the database (component S): S→A_{out}.

Exactly the same reasoning applies to the transfer of information related to the exhibits, leading to the identification of a second adapter for output. However, the scenario stipulates that the PDA has to be used to carry both kinds of information. Consequently, there is only one component A_{out} but two
Characteristics of ASUR components | Characteristics of ASUR relations
--- | ---
**Perceptual/Action location:** | **Concept:**
It denotes the physical area where the user has to focus in order to perceive information provided by the component or perform an action on it. | The considered relation carries information about a given concept manipulated by the application.

**Perceptual/Action sense:**
This designates the human sense required by the user to perceive an information provided by the component act on the component (speech or physical (visual, audio, etc.) or to action). | **Concept relevance:**
It denotes the importance of this concept for the realisation of the considered task.

**Share:**
It represents the number of users that can simultaneously access to the component to perceive or provide information. | **Representation language:**
A set of properties characterising a language may be applied. In ASUR we mainly refer to the dimension of the representation and to Bernsen’s representation properties [3].

**Representation frame of reference:**
It describes the point of view from which data are perceived or expressed.

| Table 1. Characteristics of the ASUR components and relations that constitute the different user’s interaction facets. |

relations from the component S to A\textsubscript{out} and from A\textsubscript{out} to the component U, each of them representing respectively the transfer of information related to the path to reproduce and to the exhibits.

In order to provide the right direction information to the visitor, the system has to be aware of the user’s location in the museum. Consequently, an adaptor for input (A\textsubscript{in}1) is required to get the position of the visitor in the museum (U→A\textsubscript{in}1) and transfer it to the computer system (A\textsubscript{in}1→S).

Finally, once the visitor has observed an exhibit of the path and potentially read the additional information provided by the system, he has to “validate” this step of the path, so that the system can provide direction information to go to the next exhibit of the path. An adaptor for input (A\textsubscript{in}2) is thus required and establishes a bridge between the user’s acknowledgment (U→A\textsubscript{in}2) and the state of the system (A\textsubscript{in}2→S). The fact that the acknowledgment and information visualisation occur on the same PDA is encoded by a double-relation between the components A\textsubscript{in}2 (acknowledgment device) and A\textsubscript{out} (screen of the PDA): A\textsubscript{in}2 =A\textsubscript{out}. The same relation exists between the user (component U) and the components A\textsubscript{in}2 and A\textsubscript{out}, because these two last components are handheld (U=A\textsubscript{in}1, U=A\textsubscript{out}).
The second main aspect of ASUR lead us now to characterise the different interaction facets, i.e. the ASUR components and ASUR relations denoting the user’s interaction.

Four components are involved in this case: the screen of the PDA (A\textsubscript{out}), its tactile area (A\textsubscript{in}2), the exhibit (R\textsubscript{object}) and the localiser (A\textsubscript{in}1). The location where the user will perceive and act on the two first components is his/her own hand, since the device is handheld. The visual sense is required to perceive the path and exhibit information provided by the PDA (A\textsubscript{out}), while physical action, a finger click for example, will be used to acknowledge using the PDA. No information should be shared among users, since different users may have a different path to reproduce. Physical information about an exhibit is perceived on the exhibit itself. This requires a visual sense and this perception must be available for several users at the same time. Finally, the localiser gets information from a tracking area (defined by the technology used for the tracking). When moving in this tracking area, the user will implicitly communicate his/her position to the adaptor. Finally, several users may use this adaptor at the same time. These characteristics are summed up in the Table 2.

Concerning the interaction facets, four relations are highlighted in the ASUR diagrammatic description of the situation: perception of the path and exhibit digital information (A\textsubscript{out}→U), perception of the physical entity (R\textsubscript{object}→U), user’s position localisation (U→A\textsubscript{in}1) and acknowledgment by the user (U→A\textsubscript{in}2). The first relation carries information expressed in a textual mono-dimensional language, in a frame of reference linked to the visitor so that he/she can read it. The second relation denotes the natural observation of an exhibit; it is thus based on a real 3D language and observed in a user-centred frame of reference. The two last relations correspond to output interaction facets. The user will act with natural 3D actions to either implicitly communicate his/her position to the
localiser or click on the PDA to acknowledge. The frame of reference is again user-centred. A summary of the relation characteristics is shown in Table 3.

<table>
<thead>
<tr>
<th>Interaction facets</th>
<th>Concept</th>
<th>Concept Relevance</th>
<th>Representation Language</th>
<th>Representation Frame of Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{out} \rightarrow U$</td>
<td>Path and Exhibit</td>
<td>High</td>
<td>3D, textual</td>
<td>Visitor</td>
</tr>
<tr>
<td>$R_{object} \rightarrow U$</td>
<td>Exhibit</td>
<td>High</td>
<td>3D, real</td>
<td>Visitor</td>
</tr>
<tr>
<td>$U \rightarrow A_{in 1}$</td>
<td>User’s location</td>
<td>High</td>
<td>3D, real</td>
<td>Visitor</td>
</tr>
<tr>
<td>$U \rightarrow A_{in 2}$</td>
<td>Acknowledgment</td>
<td>Medium</td>
<td>3D, real</td>
<td>Visitor</td>
</tr>
</tbody>
</table>

Table 3. Characteristics of the relations forming the different facets of the user’s interaction with the system when achieving the Mackintosh project feature “Following a path”.

4 UMLi Description of the Mackintosh Scenario

4.1 Toolkits and Model-Based Interface Development

Model-Based User Interface Development Environments (MB-UIDEs) are a state-of-the-art approach for modelling and implementing running user interfaces from user interface models [18,23]. MB-UIDEs provide models that are effective at capturing user interface functionality [9,16,21], but offer only limited application modelling facilities. Thus an important weakness of MB-UIDEs is in an area of specialism for UML, namely application modelling, while the main strengths of MB-UIDEs align with an area of weakness for UML, namely user interface modelling [19].

Several researchers have investigated the integration of interface modelling techniques with UML. For example, [10] discusses how interface modelling constructs, influenced by those in the author’s TACTICS system, in particular relating to the description of tasks, might be incorporated into UML. A more recent paper along similar lines [12] assesses several UML models for use in interface modelling, comparing them with a collection of specialist interface modelling
notations. In [17] it is suggested how several UML models, in particular class
diagrams and use case diagrams, can be used in conjunction with the author's
ConcurTaskTrees task model for user interface modelling. The integration of
ConcurTaskTrees with UML is work in progress, but can be seen as an alterna-
tive to the approach proposed in this paper. In the Unified Modeling Language
for Interactive Systems (UMLi) [19] tasks are modelled using extended activity
diagrams rather than through the incorporation of a completely new task
modelling notation into UML. Wisdom [15] is probably more mature than the
proposals in [12] and [17], in that the relationship between use cases, tasks and
views are considered in the papers. UMLi also addresses the relationships be-
tween use cases, tasks and views, but introduces fewer new models into UML and
addresses more thoroughly the relationship between tasks and the data on which
they act than Wisdom. Overall, the emphasis in Wisdom is probably on earlier
parts of the design process than UMLi, and Wisdom models tend to be more
abstract than those produced using UMLi, too abstract to generate running user
interfaces. Therefore, UMLi is the approach used in this paper as a framework
to incorporate AR systems facilities, leading in this way to the development of
the first model-based development environment for AR systems. Nevertheless,
the UMLi metamodel has been fully integrated with the UML metamodel and
the UMLi models can be built and integrated with other UML models in an
extension to the Argo/UML toolset [20].

4.2 UMLi Modelling of the Mackintosh Scenario

Many aspects of an interactive system can be described by models. Therefore,
there are many possible combinations of models that together can describe an
interactive system. Structural and behavioural aspects of systems should be in-
cluded if the intention is to build UI models that can be used, for instance, to
generate running user interfaces.

Structural Models. Classes and objects are the main structural elements of
a system. Relationships between classes and objects, i.e., associations, com-
positions and generalisations, are also structural elements. Thus, structural
models describe properties of classes, objects and their relationships.

- Domain models. Classes and objects modelling the entities of a system
  are elements of the domain. Domain models are used to describe prop-
  erties of classes and objects of the domain.
- Presentation models. Classes and objects responsible for the visual ap-
  pearance of user interfaces are structural elements. Interaction objects
  are usually called widgets. Presentation models are structural models
describing properties of widgets and their classes.

Behaviour Models. Dynamic elements used to alter the states of structural
elements, i.e., tasks, actions and events, are the behavioural elements of a
generic system. Thus, behavioural models describe properties of behavioural
elements.
We now present more precisely the domain, presentation and behaviour models of the system to explain how UMLi can be used to model the supporting system of an AR environment, and illustrate them within the Mackintosh scenario.

**Domain Models** Figure 2 presents the UMLi class diagram representing a schema for the domain of the functional core of the Mackintosh system used to support the computational capabilities used to provide information to visitors. The classes in Figure 2 identify the elements of the Mackintosh project previously described in the paper.

![UMLi class diagram](image)

**Fig. 2.** UMLi class diagram representing the system domain.

**Presentation Models** User interface diagrams are introduced in UMLi to model abstract presentations of user interfaces. As explained in [16], user interface diagrams are an alternative notation for class diagrams providing additional support for interaction classes, which are the classes representing widgets. Namely, the diagram provides visual representation for containment between interaction classes and visual identification for the main role that an interaction class is playing in a particular user interface. Different roles identify categories of interaction classes. There are six UMLi constructs for user interface diagrams used to represent this category of interaction classes:

- **FreeContainers**, which are rendered as dashed cubes, are top-level interaction classes that cannot be contained by any other interaction class (e.g., a top-level window).
- **Containers**, which are rendered as dashed cylinders, provide a grouping mechanism that brings together interaction classes other than FreeContainers (e.g., a frame within a window).
- **Inputters**, which are rendered as downward triangles, are responsible for receiving information from users.
- **Displayers**, which are rendered as upward triangles, are responsible for sending information to users.
- **Editors**, which are rendered as upward rhombi, are responsible for the two-way exchange of information.
- **ActionInvokers**, which are rendered as right pointing arrows, are responsible for receiving direct instructions from users.

In the Macintosh Project, the user interface diagram in Figure 3 represents the *ExecutePathUI FreeContainer*, which is the abstract presentation model of the user interface of the *FollowSelectPath* service. There, *ConfirmExhibit* is the *ActionInvoker* where visitors can confirm that they have reached the next exhibit in the path and *Quit* is the *ActionInvoker* used to finish the service, returning to other functionalities of the system. Moreover, *Direction* and *Info* are *Displayers* describing the route to the next exhibit of the path and presenting further information about the last reached exhibit. Finally, *GetLocation* is an *Inputter* receiving information about the location of the visitors.

As can be observed, *ExecutePathUI* represents relevant design decisions concerning the user interface of the service, avoiding early commitment to concrete properties of the interface. For instance, there is no specification of which kind of widget is going to implement each interaction class. Thus, the *Info Displayers* used to visualise objects of several media (using a PDA during the visit or visiting the digital version of the museum), as suggested by the *info* attribute of the *Exhibit* class in Figure 1, may be implemented by more than one widget.

Regarding AR systems, the *GetLocation Inputter* exemplifies an interesting kind of support that UMLi can provide to this category of interactive systems. Indeed, the *GetLocation Inputter* explicitly represents the localisation system mentioned. Thus, due to its simple mechanisms of abstraction, the user interface diagram can provide an appropriate description of how the localisation system, which is a component system of the AR system, interfaces with the rest of the AR system of the Macintosh Project.

The user interface diagram in Figure 3, along with the domain model in Figure 1, describe the structural properties of the AR system. A behavioural description of the *FollowSelectPath* service is required to complete its specification in UMLi.

**Behaviour Models** Task models are typically used for modelling interactive system behaviours in MB-UIDEs [16,23]. However, the notion of task, as conceptualised in the MB-UIDE community, is represented by use cases and activities in UMLi [16]. Using use cases and their scenarios, designers and expert users can elicit user interface functionalities required to allow users to achieve their goals. Using activities, designers can identify the possible ways to perform actions that support the functionalities elicited using use cases. Therefore, the mapping of use cases into top-level activities describes a set of interface functionalities similar to that described by task models in other MB-UIDEs.
Fig. 3. The user interface diagram of the `ExecutePathUI` FreeContainer.

In the Macintosh Project, the `FollowSelectedPath` service is represented by a use case and visitors are represented by an actor who communicates with the use case. Thus, the `FollowSelectedPath` use case is mapped into the `ExecutePath` activity in Figure 4.

In Figure 4, object flows in activity diagrams, e.g., the `vs` object of type `Visitor` and the `qt` object of type `ActionInvoker`, are used to describe the use of instances of classes to perform actions in action states. For instance, the `pt.startPath()` action state is an invocation to the method `startPath()` of the class `Path` in the object `path`. Thus, using object flows, designers can incorporate the notion of state into activity diagrams primarily used for modelling behaviour. In the case of UMLi, the use of instances of interaction objects can also be described by object flows, as in the case of the `qt` object. However, object flow states, which are rendered as dashed arrows connecting objects to action states have a specific semantics when used to associate interaction objects to activities and action states. UMLi specifies five categories of object flow states specific for interaction objects described as follows:

- **`<<interacts>> object flows** relate primitive interaction objects to action states, which are primitive activities. They indicate that associated action states are responsible for interactions where users are invoking object operations or visualising the result of object operations.
- **`<<presents>> object flows** relate FreeContainers to activities. They specify that the associated FreeContainers should be visible while the activities are active.
- **`<<confirm>> object flows** relate ActionInvokers to selection states. They specify that the selection states have finished normally.
- **`<<cancels>> object flows** relate ActionInvokers to composite activities or selection states. They specify that the activities or selection states have not finished normally and that the application flow of control should be re-routed to a previous state.
Fig. 4. The ExecutePath activity representing the behaviour of the FollowSelectPath service. The figure is a snapshot of the ARGO\textsubscript{i} model that supports UML\textsubscript{i}[20], where the FollowSelectPath service has been modelled along with other functionalities of the Mackintosh system.

- \textit{<<activates>> object flows} relate ActionInvokers to activities, thereby making the associated activities triggered ones, which are effectively started on the occurrence of an event.

The activity diagram in Figure 4 exemplifies the use of most of these interaction object specific object flows. For instance, the ExecutePathUI FreeContainer is made visible and the Quit ActionInvoker is enabled when the ExecutePath activity is active. Then, the loc Intputter starts to constantly collect information about the location of visitors producing directions on the dr Displayer and exhibit’s information on the in Displayer. In parallel with this process of producing directions and exhibit information, the service is able to receive a message either from the ce ActionInvoker saying that the visitor has reached the next exhibit in the path or from the qt ActionInvoker saying to leave the ExecutePath activity. The service finishes when the system control-flow leaves the ExecutePath activity.

5 ASUR and UML\textsubscript{i} Comparison

5.1 Model-based design of AR systems

UML\textsubscript{i} provides the ability to express interface presentations of software systems in an abstract way, identifying the abstract interaction objects required for the
user’s interaction with the underlying software system. Moreover, the user interface diagram allows the composition of abstract interaction objects. Together, these UMLi facilities constitute a step towards an integration of user interface design with underlying system design. In the case of AR systems, however, the user’s interaction with the system requires a comprehension of AR environments beyond the specification of windows-based interactions. Indeed, the designs involve the use of physical entities and it has to take into account the behaviour of such entities.

ASUR can fill this gap, since it highlights the components required to support the whole human-AR environment interaction, as well as the exchange of information among them, which represents the different facets of the user’s interaction with the system. Moreover, ASUR characterises the entities by comparing and taking into account their physical properties. ASUR does not support the expression of component refinement and description of software systems of AR environments (i.e., detailed specification of computer-based components); UMLi diagrams, however, provide a solution. Furthermore, ASUR contributes to the specification of UI presentations by specifying those aspects of interaction that relate to the use of physical entities.

As a result, we identify three layers that must be considered when designing an AR system:

- **The underlying software system layer**: as any software system, an interactive system has a set of core components, methods and behaviours that have to be designed. UML diagrams constitute an approach to design this layer.

- **The human-computer interaction layer**: this corresponds to the traditional interface presentation specification. The UI diagrams proposed by UMLi along with some ASUR concepts help designers to model the presentation of software systems, enriched by usability studies and guidelines to ensure the development of a usable UI.

- **The human-AR environment interaction**: this part of an AR environment is specific to AR systems, in which a part of the interaction relies on the use of physical entities rather than on the use of software system’s interface. ASUR describe this part of the environment as well as the human-computer interaction in terms of entity-relation models and characteristics of entities and relations. ASUR provides a framework to support the reasoning about different design issues for AR not covered by conventional design solutions for interactive systems.

A model-based AR system design may rely on the specification of properties presented in both ASUR and UMLi notations as illustrated in this paper. The combination of these three layers and bridges among them would link both notations, resulting in a first step towards a Model-Based Design Environment for AR Systems. Thus, between the computer system and the external specification, the bridge is quite straightforward, since UMLi, useful for the external specification design, is based on the UML diagrams, already adapted to the design of
the functional core. Bridging the AR system external specification design and conventional external specification design is required too. We detail this links in the next section.

5.2 Links between ASUR and UMLi

The use of ASUR and UMLi to model the “Following a path” service shows that these notations can be used to describe AR environments. However, the ASUR model in Figure 1 is obviously different from the UMLi models of the same service in Figures 2, 3 and 4. In the case of this service, the differences between the models indicate that ASUR and UMLi can support the construction of complementary models of AR environments. In fact, the ASUR model presents an AR-user-centred perspective of the same AR environment that in presented under an AR-system-centred perspective in the UMLi models.

A notation for supporting the modelling of AR environments would ideally require a mix of concepts used in both ASUR and UMLi notations. Table 4 is an attempt to identify similarities between constructs used in ASUR and UMLi by comparing their constructs to concepts informally described in the “Concepts” column of the table. Thus, concepts supported in both notations indicate the similarities between ASUR and UMLi constructs. For example, Table 4 shows that the ASUR A_in construct and the UMLi Inputter construct are both representations of the concept “Input device”.

The identification of the similarities of constructs is also a powerful indication of how the notations could be used to complement each other to comprehensively support the design of AR environments. For instance, as an ASUR construct, the A_in contains information such as action location and action sense, as described in Table 1. In the other hand, as an UMLi construct, the Inputter identifies how an object representing an input device collaborates with objects that support the functionality of the software system of the AR environment, as described by the activity diagram in Table 5. Therefore, the differences between the properties of the constructs used to represent common concepts supported by ASUR and UMLi in Table 4 strongly suggest the expected properties of the constructs of a notation which is intended to support the design of augmented reality systems.

ASUR and UMLi also exhibit some mismatches, highlighted in Table 4. Thus, for example, concerning the digital space, a difference between the notations is the absence of refinement of the digital entities in the ASUR notation. Concerning the physical space, another difference resides at the level of the characteristics of the interactions and I/O devices. UMLi does not support the specification of this aspect, while ASUR does, and this aspect constitutes the basis of potential usability analysis, discussed and illustrated in [6].

6 Conclusions and Perspectives

The design of AR systems demands new notational tools to deal with the central role in AR systems, and AR systems design, of the physical properties of the
<table>
<thead>
<tr>
<th>Context</th>
<th>Concepts</th>
<th>Concepts Representation</th>
<th>ASUR Construct</th>
<th>UMLt Construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical Space</td>
<td>User</td>
<td>U</td>
<td>Actor</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input device</td>
<td>$A_{in}$</td>
<td>Inputter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Input device interaction characteristics</td>
<td>$A_{in}$ characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output device</td>
<td>$A_{out}$</td>
<td>Displayer</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output device interaction characteristics</td>
<td>$A_{out}$ characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real tool entity</td>
<td>$R_{tool}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real object entity</td>
<td>$R_{object}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Real entities characterisation</td>
<td>$R_{object}$ characteristics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction between users and real tool or object</td>
<td>$R \rightarrow U, U \rightarrow R$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction between users and input devices</td>
<td>$U \rightarrow A_{in}$ &lt;&lt;presents&gt;&gt; stereotype</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction between users and output devices</td>
<td>$A_{out} \rightarrow U$ &lt;&lt;presents&gt;&gt; stereotype</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Characterisation of interaction</td>
<td>Characteristics of ASUR relations (facets)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td>Interaction between input device and the computer system</td>
<td>$A_{in} \rightarrow S$ Interaction object flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digital Space</td>
<td>Interaction between output device and the computer system</td>
<td>$S \rightarrow A_{out}$ Interaction object flow</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Digital entities</td>
<td>$S$</td>
<td>Classes and Objects</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Digital entity properties</td>
<td></td>
<td>Attributes and Operations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Relationships between digital entities</td>
<td></td>
<td>Associations and their specialisation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Goals</td>
<td></td>
<td>Use cases</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tasks</td>
<td></td>
<td>Use cases + Activities</td>
<td></td>
</tr>
<tr>
<td></td>
<td>User interface presentation</td>
<td></td>
<td>User interface diagram</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. A comparison of how concepts of AR environments are implemented in ASUR and UMLt.
interaction entities and the relationship of physical with informational entities. ASUR provides a potentially useful candidate, but it does not have the capability for describing the user interface(s) that are part of such an AR system. UMLi, on the other hand, is tailored for this job, but, unlike ASUR, it cannot capture the physical properties of components and their relationships with other entities and with information flows.

In our comparison of ASUR and UMLi we identified three levels of design involved in AR systems: human-AR environment interaction, human-computer interaction, functional core. ASUR, and other HCI design tools, deal with the first two levels. Different design alternatives can be described and subjected to analysis in terms of the physical environment (what moves, what touches), the interaction (perception, action, cognition) and the implementational approach (sensor deployment, wireless vs. wired communications, use of wearables). UML and UMLi, on the other hand, offer a method of specifying precisely the behavioural aspects of the interactive system. Of course, ASUR and UMLi offer a way of capturing some of the important features of AR systems, but they will probably not be sufficient on their own. We can expect that additional notations may be needed to capture other aspects of the design (e.g., movement patterns of the users and artefacts).

We envisage several parallel developments from this point in our work, including:

- Empirical studies of the use of ASUR and UMLi with realistic AR system design problems. We need validation of the analytic results presented here and also need to identify gaps in our understanding of the requirements for AR notations.
- Enhancements to the notations. Apart from adding to the expressiveness of each notation, we also need to look at the effects of handling multiple collaborating users and augmented artefacts and dealing with the scalability issues involved in “industrial strength” design problems.
- Exploring further links between the notations. This may lead to the generation of transformations on descriptions (e.g., transforming ASUR adaptors into UMLi Inputters and Displayers).
- Providing tool support, not only for editing, but also for linking related aspects of designs, comparing alternative designs, carrying out analyses, and generating descriptions in the other notations (semi)automatically.

Also, as we stated at the beginning of this paper, our ultimate goal is to develop a systematic approach to AR system design — a design method. Our exploration of ASUR and UMLi and their links may constitute a starting point towards this aim.

Acknowledgements We would like to thank our colleagues in the Equator Consortium for the inspiration for our scenario, especially Matthew Chalmers, Ian McCall, Barry Brown, Areti Galeni and Martin Ritchie. It was the Mackintosh Project that initially stimulated our interest in capturing alternative AR designs.
We also thank Laurence Nigay for her constructive comments on an earlier draft of the paper.

References

7. Equator Project Web Site: http://www.equator.ac.uk/
Tool-Supported Interpreter-Based User Interface Architecture for Ubiquitous Computing

Lars Braubach, Alexander Pokahr, Daniel Moldt, Andreas Bartelt, and Winfried Lamersdorf

Distributed Systems Group, Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
{braubach, pokahr, moldt, bartelt, lamersd}@informatik.uni-hamburg.de
http://vsis-www.informatik.uni-hamburg.de/

Abstract. With the upcoming era of Ubiquitous Computing (UbiComp) new demands on software engineering will arise. Fundamental needs for constructing user interfaces (UIs) in the context of UbiComp were identified and the subsumed results of a survey with special focus on model-based user interface development environments (MB-UIDEs) are presented in this paper. It can be stated, that none of the examined systems is suitable for all the needs. Therefore a new architecture based on the Arch model is proposed, that supports the special UbiComp requirements. This layered architecture provides the desired flexibility with respect to different implementation techniques and UI modalities. It was implemented in a user interface development environment called Vesuf. Its usability was approved within the Global Info project [20], where heterogeneous services had to be integrated in a web portal.

1 Introduction

As covered below, UbiComp applications differ inherently from conventional applications. In [8] were identified several special UI needs for this kind of systems, which will be presented in the following. First of all it can be stated that UbiComp applications are somewhat more complex than comparable conventional programs, because they have to cope with dynamical changes during runtime. Therefore it should be an important objective for a UI construction system to hide some of this complexity (simplification).

In [2] Banavar et al. explain that the application development for UbiComp has to be device independent, because a single application should be usable from distinct entry points, e.g. from a laptop, handheld or even phone. This implicates some advantages for the user such as synchronous data and familiar handling across different devices. To achieve this vision within a UI construction system it is necessary to provide mechanisms for cross-platform user interfaces. Therefore a clean separation between user interface and functional core is needed, as well as mechanisms for connecting the separated parts. Moreover it should be possible to create different interface modalities for an application (flexibility), and the
system itself should be open for the integration of new interface modalities and implementation techniques (extensibility).

Banavar et al. further point out that the UbiComp paradigm will lead to substantial changes on how users perceive applications. They will understand the application as a composition of services, which takes into account the current context of use, e.g., the location, time or weather. To be able to support interfaces for this new kind of applications it is necessary to address the dynamic adaptation and composition of UIs.

The next section takes a look into what existing tools offer for the special UbiComp needs. In Sect. 3 a new model-based architecture and a concrete system for better accomplishing these goals are presented. Thereafter system details are introduced in Sect. 4, which help to achieve to some degree the UbiComp demands mentioned above. In Sect. 5 as an example a metadictionary service is described. Finally in Sect. 6 follows a summary of the results and an outlook.

## 2 Survey subsumption

In search of a suitable system for constructing UIs of UbiComp applications several different categories of tools have been researched [8]. The two most promising approaches are frameworks and model-based systems, which were further investigated by typical representatives (see Table 1). The consolidated results of [8] as shown in Table 2 are presented next.

The goal of a simplified UI construction process is currently achieved by frameworks only. Because of the low abstraction level framework implementations offer a flat learning curve for the developers and additionally whitebox frameworks allow easy programmatic extension. MB-UIDEs suffer from the lack of well established standards in the field of some partial models for UIs, especially for presentation and dialogue control facets.

It is stated that the clean separation of UI and functional core is achieved by frameworks as well as by model-based systems. While most frameworks utilize agent-based architectures like MVC [10] and PAC [11] the MB-UIDEs accomplish separation in a natural way through their models. The connection between these two components, relevant for an executable interface, is established by

---

**Table 1. Surveyed systems**

<table>
<thead>
<tr>
<th>Frameworks</th>
<th>Suggested literature</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVC-Client</td>
<td>[30]</td>
</tr>
<tr>
<td>SanFrancisco</td>
<td>[34]</td>
</tr>
<tr>
<td>JWAM</td>
<td>[6]</td>
</tr>
<tr>
<td>MVP</td>
<td>[27]</td>
</tr>
<tr>
<td>Amulet</td>
<td>[23]</td>
</tr>
<tr>
<td>MB-UIDEs</td>
<td></td>
</tr>
<tr>
<td>Janus/Jade</td>
<td>[1]</td>
</tr>
<tr>
<td>Mobi-D</td>
<td>[28]</td>
</tr>
<tr>
<td>FUSE</td>
<td>[22, 5]</td>
</tr>
<tr>
<td>TRIDENT</td>
<td>[7]</td>
</tr>
<tr>
<td>TADEUS</td>
<td>[14]</td>
</tr>
<tr>
<td>Tealbach</td>
<td>[15, 17]</td>
</tr>
<tr>
<td>MASTERMIND</td>
<td>[9, 31]</td>
</tr>
<tr>
<td>BC-Prototyper</td>
<td>[35]</td>
</tr>
</tbody>
</table>

---
programming within the framework context, whereas model-based systems use

descriptive techniques to link the interface to the functional core. Some of

test MB-UIDEs like TRIDENT or FUSE do not establish the connection

to functional core, leading to substantial additional effort for manually

linking the parts.

Frameworks do not address the extensibility with regard to various UI modal-

ties, as there are different frameworks for different purposes, e.g. for web-services

or for interactive systems. Regarding the extensibility with respect to further

implementation techniques, some frameworks address concepts for integrating

database systems. Model-based systems are conceptually qualified for both exten-
sibility issues. In practice only few of the tested systems exploit the potential

of the model-based approach for supporting different interface modalities, or regard

extension mechanisms to integrate more than one implementation technique

like Janus and Teallach).

When considering the flexibility of frameworks it is obvious that these types of

systems are not able to support easy mechanisms for changing interface modal-

ities, because the glue between UI and functional core has to be programmed. Also

few MB-UIDEs utilize their declarativeness for flexibility (except TADEUS). The

same is true for adaptation and composition aspects, which should be considered

when building constructing systems for UbiComp. MASTERMIND is the only

system, which addresses adaptation with respect to display properties. No tested

framework even tries to cope with one of these aspects.

From this comparison it can be concluded, that the non-declarative ap-

proaches (frameworks) are not well suited for UbiComp UI construction sys-

tems, because they do not offer enough flexibility and are settled on a too low

abstraction level. Model-based systems overcome these conceptual problems and

possess all features which make them appear fully qualified for UbiComp needs.

Within this group the interpreter-based systems (see [12]) seem to be the most

promising, because they are able to handle the UbiComp needs with respect to

dynamic changes of the UI during runtime. Before utilizing model-based sys-

tems for UbiComp some problems have to be solved. One important aspect is

the lack of established UI standards. Further on the support of various interface

modalities must be improved, and the research in the field of adaptation and

composition must be carried on.

Table 2. Subsumed UbiComp-characteristics of the system categories [8]

<table>
<thead>
<tr>
<th></th>
<th>Simplification</th>
<th>Separation</th>
<th>Extensibility UI/Connection</th>
<th>UI Flexibility</th>
<th>Adaptation</th>
<th>Composition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frameworks</td>
<td>x</td>
<td>x/x</td>
<td>o(x)</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>Model-Based</td>
<td>o</td>
<td>x/(x)</td>
<td>(x)/(x)</td>
<td>(x)</td>
<td>(x)</td>
<td>o</td>
</tr>
</tbody>
</table>

x: Nearly all systems  (x): Few systems  o: None of the tested systems have the property
3 Vesuf System

As a starting point for the design of a new system, a wide research in the field of tool-categories, techniques and architectures and their applicability with regard to UbiComp has been carried out [8], and a vision of a UbiComp development environment has been conceived [26]. This section will present the Vesuf system, a first step towards realizing the vision.

3.1 Vesuf Overview

As pointed out in the last section, interpreter-based MB-UIDEs are potentially well suited for all UbiComp requirements. Thus the Vesuf system consists of the components shown in Fig. 1. To realize an application within the Vesuf system the functional core can be implemented system independent (application implementation). This means that no system specific code-intrusions are necessary. In addition the different submodels for the UI-specification have to be created. To assist the developers in this process several tools can be used (design-time tools). At runtime the model-information will be evaluated by the interpreter which utilizes different automation tools. It constructs an executable UI by using further system features which are presented in detail in Sect. 4.

There is no fixed methodology to follow when developing applications with Vesuf and the models can be defined in an independent manner. The Vesuf system is designed to support what in this paper is called “slinky automation”. This allows to start the development process with minimal specification effort, and utilize the automation tools to a high degree, in order to have executable interfaces in early design stages. During the further development the model specifications will become more elaborated, and therefore the need for automation will decrease, while the UI quality increases (see [32]). With this approach inspired by HUMANOID [33], rapid prototyping is possible.
To be of use in a broad UbiComp context the system offers two different interpreter-modes. Active interpreters such as the GUIrunner and the VesuApplet are suitable for interactive applications and manage the transition between views by opening and closing application windows. Passive interpreters like the ApplicationServlet are mediator components which pass on all collected changes from the UI to the functional core and backwards.

3.2 Runtime Architecture

The layered architecture of the Vesu system is pictured in Fig. 2. Primarily it must be stressed that a strict separation in so called model and instance elements is established in all layers. In [21] Kent et al. propose to introduce this separation as a foundation of the UML metamodel. Model elements are used to describe the various models, whereas instance elements represent concrete occurrences of the model elements at runtime.

Furthermore the horizontal partitioning in five separate components is conceived after the paragon of the Arch model [4]. The corresponding names of the Arch model components are denoted above each Vesu layer. The Arch concept assures the independence from the implementation- and toolkit-specific side through the introduction of two adaptor components. In the domain-adaptor component all information about tasks and domain entities is encapsulated, while in the presentation component abstract and concrete display information is held. A central dialogue component is responsible for managing the global state of the UI.

In the following the models and their connections will be sketched. The domain-adaptor component is specified by a composite domain model, which
consists of tasks and domain objects. This domain model is adopted from the
UML metamodel and use cases are used for describing task behaviour.\footnote{It is planned to use the ConcurTaskTrees\cite{25} as foundation for the task model in a subsequent release.}

The dialogue layer contains a dialogue model, which currently is built upon
the UML statechart semantics. Each application view is represented by one state.
For concurrency purposes the concept of control flows is established. Each control
flow is an autonomous state machine, which controls one part of the UI. To
determine what will be displayed to the user, states are connected to domain
entities (e.g. tasks or objects) via paths.

To map tuples of domain element and dialogue state to presentation elements
a flexible mapping is used. The presentation metamodel extends the UML and is
based upon functional roles. A presentation element is linked to a toolkit element
which is responsible for its appearance, and to a domain element for realizing
the information flow between implementation and user.

4 System Details

An overview of the proposed architecture has been presented, but it is yet to
show, how it fulfills the identified requirements. Ubiquitous Computing demands
the clean separation of device-specific and device-independent parts of an appli-
cation. The Arch model provides five components, that can be abstractly de-
veloped and reused across a range of different environments for any application, or
be specialized, e.g., to enable the use of device-specific features, according to the
suitability in the application context. The border between context-, device- and
user-independent and -specific parts can be flexibly shifted across the compo-
nents of the Arch model. For easy integration of the components, a combination
of several concepts is proposed.

These concepts heavily rely on the interpreter-style of the architecture, and
therefore have not been found in recent generator-style model based systems (like
Mobi-D, TADEUS, Janus, etc.). They have been inspired by other interpreter-
style model-based systems such as ITS\cite{36}, HUMANOID and framework ap-
proaches like Amulet. In the following it is stated how these concepts aid in
fulfilling the requirements presented in Sect. 1.

4.1 Domain-Adaptor Layer

Most model based systems try to solve the problem, how to connect the UI
to the application implementation. In Vesuf the domain-adaptor layer conceals
the implementation layer, and represents the application functionality from the
viewpoint of the user interface.

With a flexible mechanism called “implementation accessor”, any type of
implementation technology (i.e. legacy system) can be integrated into the Vesuf
environment. Currently the system has full support for Java implementations
and prototypical support for implementations in relational databases.
The use of the UML metamodel as foundation of the domain-adaptor layer establishes a unified view on top of implementation details, and allows the specification of user interface related meta information (e.g. constraints) as first class objects together with the domain entities.

4.2 Paths

Puerta and Eisenstein [29] describe the mapping problem between what they call abstract (task, object) and concrete (dialogue, presentation) elements. They regard the solution to this problem as “essential for the construction of model-based systems”. In Vesuf the connection between abstract and concrete elements is established using paths.

The Vesuf path language allows to specify navigational paths across all elements of the metamodel (e.g. classifiers, attributes, constraints). It is comparable to XPath [37] which provides navigational access to different node types of XML documents, such as element, attribute and text. XPath introduces the concept of axes which specify DOM associations to follow (e.g. child, attribute, descendant).

The associations in the UML metamodel (e.g. attributes of classifiers) provide the axes of the Vesuf path language. For example to refer to the value of an attribute (starting from a point object), one would write Point.<attribute>X.<value>, or refer to the constraints of the attribute by writing Point.<attribute>X.<constraints>. Note that the identifiers in angle brackets are the axes that denote references of the UML metamodel, while the other identifiers (e.g. “Point”) denote elements in the domain model of the application.

To be evaluated, a path is instantiated with an instance of the starting element (e.g. point). As the references in the object graph may change, the endpoint element of a path instance may also change. Paths hide the problems of dynamical changes, because they provide a static way to refer to dynamically changing elements. For example a presentation element uses a path as reference to the domain element it displays. While the displayed element may change over time (as described in Sect. 4.4), the path will always be the same.

Although paths are evaluated at runtime they are statically typed, and their correctness can be checked against the domain model at design time. Since paths can be used to navigate across all different models (object, task, dialogue and presentation model), they provide the glue, to stick together the different components of an application. Therefore most of the higher level concepts rely on paths.

4.3 Extended Constraint Semantics

In UML a constraint is a semantic condition or restriction expressed in text, represented in the metamodel by a boolean expression on an associated model element [24]. In Vesuf constraints are used to stipulate possible user interactions and valid user input. As in the Seeheim model [16] this places the validation of user input into the application interface layer and not into the presentation layer. This facilitates the reuse of input validation for all interface modalities.
Fig. 3. Use of constraints for presentation elements

The semantics of constraints are extended to allow the specification of supplementary information, that is utilized by presentation elements. For this purpose constraints include additional properties with special meanings. The additional properties are specified as constant literals, or as paths referring to domain attributes or operations, which are evaluated dynamically at runtime. Since constraints can be realized as operations in the implementation layer, tools can be used to generate constraint implementations, e.g., from OCL-specifications. Currently Vesuf defines five different types of constraint properties which may be used independently or together. These are described next:

**valid** specifies the boolean expression to validate user interaction (e.g. for operations or navigational events) or input (for attribute and parameter values). This represents the standard UML semantics of constraints.

**check** specifies via a path an operation, that may throw an exception, when the constraint is not valid. The exception object can include additional information, why the constraint is invalid, e.g. a text message as in Fig 3b.

**values** specifies the set of possible values, e.g., for an attribute or parameter. The set of possible values can for example be used by presentation elements to create radio buttons or to fill in lists or drop-down boxes (see Fig 3a).

**range** alternatively to a set of possible values a range can be specified. All Vesuf built-in data types (e.g. integer, float, date) support this. Range handling for application specific data types can be added (e.g. for IP-address). The range constraint can be used to handle interaction with scrollbars (see Fig 3c).

**active** For dynamically prohibiting access to certain elements (e.g. attributes and operations) of individual objects in the domain-adaptor layer, the active constraint is used. It causes enabling and disabling of interaction elements and therefore provides a way for realizing intra-dialogue behaviour in the domain-adaptor layer, that is “inherited” (i.e. mirrored) by all specialized interface modalities.

When specified with paths, constraint properties are evaluated dynamically and internal changes are propagated by events as described in Sect. 4.4 and exemplified in Sect. 5. Besides the predefined types, custom constraint properties can be specified and then be used by the presentation layer.
The use of constraints enforces a certain level of usability, because they provide meaningful error messages. When used in conjunction with widget-mapping techniques, appropriate interaction elements are automatically selected based on the type of constraints specified for domain-adaptor layer elements.

4.4 Events and Dependencies

In Vesuf an event dispatcher component manages the collection, generation and multicasting of events. Events are initiated by instance elements in the domain layer (e.g. value of an attribute changed) or dialogue layer (e.g. a state change in a dialogue state machine). Event handlers can be registered on any type of instance element, and are used to couple loosely elements from different models. Besides propagating events to the appropriate handlers, the dispatcher manages the generation of dependent events. With dependencies the need for explicit event handlers in the presentation layer, reacting to changes in the other layers, is reduced. The system manages two types of dependencies, as described next.

Dependencies can be specified explicitly in the domain model using the UML dependency element. It is augmented by a tagged value, that specifies a path from the client to the supplier element of the dependency. The path enables the system to determine at runtime the supplier instance elements, that participate in any dependencies, and take the appropriate actions, when these supplier elements initiate events. An example for a dependency that has to be specified explicitly is the area attribute of a rectangle object, that depends on the values of the width and height attributes. When the width or the height value is changed, the dependency will cause an update of any presentation elements displaying the area attribute.

The second type of dependency arises from the use of paths for specifying properties of elements. When a section of a path changes, the endpoint of the path, and therefore the element that specifies the path (e.g. a view) also changes. When for example in a circle object a new center point is set, presentation elements displaying the x and y attributes of the old center point object are automatically adjusted to refer to the new center point. This is handled by Vesuf internally, and events for any dependent elements are automatically generated and published in the next event multicast (together with the initial event, that triggered the dependency).

4.5 Presentation Metamodel Based on Functional Roles

Since the behavioural aspects of the UI are captured in the domain-adaptor (using constraints and dependencies) and the dialogue layer, the Vesuf system features a very lightweight presentation layer, thus supporting the flexibility and extensibility requirements. The UML metamodel is extended with new elements rooted in an element called part. Besides this generic interface element four different types of elements are introduced (see Fig. 4). The system utilizes UML [18] for specifying presentation models, because of its genericity and tailorable to specific environments.
The motivation behind the lightweight approach is, that the presentation elements only provide the glue between the concrete interaction elements in the toolkit specific layer, and the application specific elements in the domain-layer. Therefore, in contrast to other proposals such as UMLi [13], the elements of the presentation metamodel are not classified by their specific interaction capability (e.g. input, output, ...) but rather by the intention behind the element, i.e., the functional aspect of the connection between a toolkit element and a domain element. Three basic functional roles of atomic UI elements have been identified: Delegates, labels and descriptions. These elements are atomic in the sense, that they do not contain other elements.

For interaction elements, the concept of UI-delegates as self-contained representatives of domain elements, proposed by Holub in [19], is adopted. Delegates are the most important parts of the UI, as they enable the user to interact with elements in the other layers. In Vesuf, the actual domain-adaptor and dialogue layer elements, that are represented by a delegate are referenced via paths, which can be resolved at runtime to yield the corresponding instance elements. Delegates mediate, e.g., between attributes in the domain-adaptor layer and textfields in the toolkit layer.

The other two basic part-types (label and description) are not used for interaction, but to provide structural and usage information to the user. Labels are designations of elements in the other layers, usually placed near delegates, to designate which domain-adaptor or dialogue layer elements the delegate refers to. In the toolkit layer, labels are realized by texts, icon images or characteristic sounds. Description elements provide usage information related to elements in the domain-adaptor or dialogue layer, and can be used to provide context sensitive help (e.g. as toolips). In addition to description texts these elements use information provided by constraints to inform the user about the current state of the interaction (e.g. invalid input).

To organize presentation elements in groups, container elements are used. They recursively aggregate the atomic presentation elements (delegates, labels and descriptions) that are to be displayed as a presentation unit. The container hierarchy is usually, but not necessarily, reflected by similar structures in the toolkit layer.

The lightweight presentation metamodel leads to simple, and easy to maintain interface descriptions. Furthermore, it allows the system to be easily extended by new interface modalities.
4.6 Interoperation of the Concepts

It has been shown, how the aforementioned concepts aid in fulfilling the requirements posed by UbiComp. The domain-adaptor component is the backbone of the architecture and features a runtime environment which manages model and instance elements with automatic handling of constraints, events and dependencies. Paths are defined on top of the modelled domain structure, to establish the connection with the other layers. The presentation elements use the potentially dynamic information of the domain-adaptor layer elements, to extract the properties of their widgets.

The dependency mechanism allows for intra dialogue control (e.g. en- and disabling of buttons) to be specified abstractly in the domain model (e.g. allowed parameter values for method invocations). The event mechanism automatically updates widgets that are dependent in this way. This mechanism of inheriting behaviour from the domain-adaptor facilitates robust and consistent behaviour in all interface modalities, and avoids redundancy in the different presentation models of an application.

5 Example

To prove the practical utilizability of the Vesuf system, it has been applied in the context of the GlobalInfo project [20]. It was used to integrate several services into the PublicationPORTAL [3]. One of these services is the metadictionary service that allows to query several online dictionary web sites (Fig. 5). The realization and integration of the services is described in detail in [8].

Using the example of the metadictionary Java application (see Fig. 5b), it is shown, how the concepts allow behaviour defined in the domain-adaptor layer to be mirrored in the presentation layer. The first example is the invocation of the Translate operation that uses the constraint / dependency mechanism. The operation is represented in the presentation layer using a button delegate. Choice delegates are used to represent the To and From parameters of the operation. These two delegates are self contained and not connected to each other. Nevertheless, when the user supplies parameter values, that are valid on their own but invalid in combination (i.e. a translation not supported by any dictionary), the button representing the operation will be disabled, since an appropriate constraint is specified in the domain model. This behaviour will appear in all presentations that support dynamic en- and disabling of operation delegates. In static presentations (e.g. web forms as in Fig. 5d) the user will be able to invoke operations with invalid parameter combinations, and will subsequently be presented an error page describing the constraint violation.

The second example utilizes the dynamic path concept. Consider the lower half of the metadictionary window in Fig 5b, where a left hand side list box allows for selection of a result set, which is then presented in detail at the right hand side. This is realized in Vesuf by just using a path to the same attribute denoting the selected result set for both delegates. When the attribute value
Fig. 5. Screenshots of the metadata
tionary service: Executed by the Motorola Mobile
ADK for Voice (a), as Java application (b), displayed in the Yospace SmartPhone
emulator (c), embedded in the PublicationPORTAL (d)
represented by the list box is changed, all interaction elements in the detail panel will be notified that a new result set has been selected and will therefore update their presentation appropriately. The same behaviour is exhibited in the web-portlet (Fig. 5d), where the current result set can be selected with the *Show* button and is subsequently presented at the bottom of the page.

6 Conclusion and Outlook

In this paper six major goals for UI construction tools in the context of UbiComp were presented. The proposed model-based architecture has been developed to address the arising needs. It is now shortly summarized how the architecture and the system characteristics help to fulfill these requirements.

Simplification of the development process is addressed by using standards to a high degree. The system is based on UML wherever applicable (domain-, task- and dialogue model) and uses UML as a standard notation for the simple functional role based presentation model. Furthermore the development process is simplified by using automation tools and applying the slinky automation idea.

The architecture enforces the separation of the UI in five components according to the Arch model and introduces a generic mechanism (paths) to connect elements of these components at runtime. The extensibility with respect to new UI modalities is supported in a natural way by the underlying architecture and the usage of the delegate concept. To allow simple extensions with regard to different implementation techniques the implementation accessor concept is introduced. Flexibility with regard to UIs is achieved by applying constraints and dependencies. They relieve the presentation components from complex responsibilities such as input validation and intra dialogue behaviour. The resulting lightweightness of presentation components was one of the main goals with respect to UbiComp, featuring a minimum of redundancy between different UI specifications and a maximum of reusability of information rich domain models.

For adaptation and composition a sound foundation is set by the architectural separation of model and runtime layer and the interpreter-style is well suited to react to dynamical changes imposed by these demands.

Further research within the Vesuf project\(^2\) will cover reusability aspects, especially the construction of UIs in a “LEGO” like manner. This becomes possible by the removal of all behaviour from the presentation layer and allowing UI components (LEGO bricks) to be placed on the domain-adaptor layer (LEGO ground plane). The connections between the bricks and the plane are established with paths and therefore allow to build up very complex interface elements from simpler ones in a declarative way.

References


\(^2\) The Vesuf project is available at: [http://vesuf.sourceforge.net](http://vesuf.sourceforge.net)


Vilars. A new dialogue model applying Augmented Reality

Granollers, T.¹; Lorés, J.¹; Raimat, G.¹; Junyent, E.²; Tartera, E.²

¹ Departamento de Informática e Ingeniería Industrial. University of Lleida.
² Departamento de Historia. University of Lleida.

Tél: 973 702 700 Fax: 973 702 702
E-mail: {tonig, jesus}@griho.net URL: http://griho.udl.es, http://www.diei.udl.es/

Abstract. New interaction paradigms such as Ubiquitous Computing or Augmented Reality are emerging. We believe that it is evident that the change of paradigm has to suppose a change in the design methodology of these interactive systems. The purpose of this paper is to present our proposal about how this change has to be done in a real Use-Case. We offer the use of Augmented Reality during a visit to the Vilars archaeological site (Spain), as an example. The document is focused on the design of the system, explaining how to migrate from the Desktop paradigm to the Augmented Reality one adding the location attribute in order to organize a visit to the site, enabling the establishment of a relationship with the interaction by direct manipulation. This relation establishes a different kind of visit to those carried out at present, based on guides, books or web sites, among others. We present how the functionality and the user dialogue have to be done for the accomplishment of a site visit in Augmented Reality using the User-Centred Process Model.

1 Introduction

During a visit to an archaeological site the visitor obtains the information of the diverse elements from the site by means of informative fixed panels -arranged at the key points of the tour-, leaflets, books, devices audio (individual) and expert guides.

In the case of the panels, the visitor follows the site searching for the ones that show the characteristics of the point where he/she is located.

Fig. 1. Informative panel of the fortress and a “small” actual map of the site.
These visits do not differentiate between types of users and as a consequence they limit the visitor who is looking for additional information.

Another source of information—with the need to cross the whole site—can be a its Internet web site. In this case the user must interact looking for the information that he/she wants to obtain.

Some sites also have additional information points, which allow to complement the real visit once concluded the tour or even before beginning it. These areas tend to be external to the site itself and they can show videos, 3D reconstruction, multimedia systems, as well as provide guides or leaflets to the visitors, etc.

Why does the idea of this project arise?

As we just have seen, all the available information that actually a visitor can obtain are separated and disconnected one each other. Having to move from the place that he/she visit to obtain additional information has a negative influence on the visitor. It provokes the loss of mental images, so the visitors vaguely remember all the parts that they have visited, and can confuse elements and lose details.

In consequence, there is the possibility of creating a new concept of visit that does not need additional information off-site which in this way guarantees the complete satisfaction of all the visitors [1].

The idea arises of creating a type of visit that allows greater precision in the information depending on the needs of a user, greater interactivity for the visitor, who is the one who determines the quantity and quality of desired information about a point in question, in this way adapting the visit to his/her own profile.

This way the visitor will move around the site being able to give greater attention to the points that he/she considers of special interest, allowing him/her to contemplate its reconstruction by means of a virtual image of a selected period. It offers, besides, all the corresponding archaeological, urban development and sociological information.

This new type of visit will be applied to the "Els Vilars" archaeological site, which we will detail below.

2 “Els Vilars” fortress: General Characteristics

Els Vilars site was discovered in 1975 and from the year 1985 it has been an object of systematically excavations up to the present time. The fortress was constructed on an extensive plain located in the northeast of the municipality of Arbeca, in the region of Les Garrigues (Catalonia) [2][3].

The works developed till now have enabled a complete definition of its perimeter and have revealed important features of its internal organization.

The singularity of the fortress resides in its particular defensive system, formed by a 5m. thick wall with twelve attached towers, a barrier of stones dug into the ground and a ditch with walls of stone that measures 13m. wide and 3m. deep.

The interior urbanism of the enclosure was organized around a space or “central square” presided by a monumental uncovered cistern, lined with stone and with a corridor that allowed access to the changeable level of water.

We find five urban development and architectural relations that constitute the main element to systematize the stratigraphic sequence, that has been divided into five

3 Augmented Reality

The Augmented Reality paradigm of interaction tries to reduce the interactions with the computer using the information on the environment as an implicit entry. The user is capable of interacting with the real world, which appears increased by the synthetic information on the computer [4] [5].

But, how does it differ from Virtual Reality?

In Virtual Reality the user is inside a virtual “not real” world that completely replaces the real exterior world, whereas in Augmented Reality there is a mixture of the real world and the virtual one. The real world takes part in the virtual world thereby obtaining an improvement in the relations and interactions of people with reality [6].

To obtain this connection of worlds, Augmented Reality recognizes the situation of the user using recognition methods such as position, orientation and “archaeological” time. It’s a technological question that increases and improves the user’s real world vision and where the area of attention is the environment and not the computer. Thereby, the computer improves the interaction between the real world and people.

In fact Augmented Reality takes the information inside the real world, obtaining an improvement in the interaction between human beings and the real world, instead of taking the user inside a computer’s virtual world, as Virtual Reality would do.

4 Current Experience

Our current work is only a step in order to develop the whole system that will enable real visitors to carry out the visit of the site with our new proposal.

The prototype. Until now we have developed a first prototype of tablet or device – made of wood and paper – with which a visit would be carried out.

3D reconstruction. A complete 3D reconstruction of the site was developed focused on the reconstruction of the Iron Age fortress [3].

Scenarios. Planning the best script to record the videos we prepared different scenarios with the purpose to envision the way to simulate a future site visit.

The videos. Two videos were developed; the first one was a promotional video with images from the real site mixed with the 3D reconstruction. The second video – recorded in Els Vilars site, with real actors using the above-mentioned prototype – shows the first simulation of what a future visit in Augmented Reality [2] would be like. This second video served us like a simulation that gives us a better understanding of what we are trying to achieve and to improve with this new type of visit [7].

The evaluations. Using the 2nd video and the tablet, four “Focus Group” evaluations [8] has been carried out. The meetings that were held were very helpful to reach a better understanding of the possible problems it would encounter [9].
The opinion about the prototype presented was generally positive and the evaluations favoured improving the proposed options, as well eliminating or creating newer ones. We also received opinions about the position of the buttons as well as the size and shape of the tablet, etc. Also they showed the disadvantages about certain tools that are not interesting—as would be the use of the zoom—.

All the collected opinions will facilitate the creation of a new prototype that will better adapt to the needs of the users.


When designing an interactive system we base it on a User-Centred Process Model and for that purpose we have developed our own proposed methodology called “Usability & Accessibility Engineering. User-Centred Model Process” (Fig. 2).

![Fig. 2. Usability & Accessibility Engineering. User-Centred Process Model.](image)
Lifecycle”) [11], even though some ideas have been taken from them. Next points can briefly summarize our diagram:

- **conceptual organization**, that locates everything where and when it is needed, facilitating a methodology during the design of an interactive system.
- the **three basic columns** that take Software Engineering inside a Usable and Accessible process are clearly separated. These 3 columns are: Classical Software Engineering (left column), Prototyping (centre column) and Evaluation (right column).
- the scheme gives the **importance to the user** that a User-Centred Process Model should have.
- the arrows specify the **iterativity** among the different stages and where the users might participate.

In the initial stages of the design we encourage the user to become an active participant. This enables us at an early stage to see the faults, virtues and improvements of what we will implement in the future. This characteristic is common with all the rest of User-Centred Process Models.

The prototypes to evaluate a system—in order to imply users—can be made of very diverse materials as for example paper, cardboard, wood ... [12] so what is expected of them is that with a minimal effort in their design they can help us to represent a simulation of the final system. We experimented—in a previous paper [2]—the use of the envisioning design methodology [13] that uses the scenes as a design tool, which is used as a base from which to develop, in a close future, a system of Augmented Reality applied to the visit of an archaeological site. For these scenes different storyboards and video prototypes were developed—as has been mentioned above—.

Nielsen observes [14] that simple user’s tests are frequently more useful than theoretical designs or more complex testing methods.

6 New approach: Analysis and design

During the lifecycle of an interactive project diverse loops going through the different stages of the scheme, that fig. 2 show, can be followed. Before this paper a first loop has been traced. This loop has gone across Requirements Analysis, Design, Prototyping and Evaluation nodes of the Process Model.

In the current work we give a second loop moving towards system functionality and dialogue, so our main work has been developed in Design node.

We present the important aspect that using the mentioned archaeological site like a test bench has enabled us to verify and to study how the location, as an input attribute coming from the change of paradigm, has influenced the design model. This confirms our supposition that the paradigm change needs a model change and we will show a proposed method for that.
6.1 Collecting Requirements

As a first step diverse visits to the site have been carried out to study the guided visits that the archaeologists are giving with the purpose to analyse better the possible problems that a real visitor would meet.

We have carried out an ethnographic study of the site as a study of the different profiles of the users that might make the visit.

The urbanism and social life study about Els Vilars allows us to summarize a wide range of information from the site. All this information has been added to the most attractive points of interest for a future visitor, understanding as points of interest the elements that compose the site such as the doors, the ditch, the wall, etc.

The visitor’s profile is another important factor to take into account in this phase of the process. Every person acquires the knowledge about the system’s functioning with which he/she interacts in a different way due to the different Mental Models [15] that he/she has. This Mental Model has a special relevance at the moment of designing interactive systems. Analysing the different mental models which we will meet will allow us to adapt the interface to the user's different levels in order to personalize the visit more depending on each one.

With the summarized information we will carry out a Tasks Model based on the Hierarchical Task Analysis method.

Part of this work was done during the previous work (the first loop mentioned above) and was enlarged with this second loop.

6.2 Analysing the New Paradigm

6.2.1 Introduction

The user who accedes to the site for a visit will have to obtain a device —at the information point of the site— that will allow him/her to interact with the environment. This device will be a tablet computer light enough for the visitor to be able to take it comfortably during his/her tour around the site.

Fig. 3. Tablet Prototype (device that the visitor will have to obtain in the information point of the site) and a visit simulation in Augmented Reality (image from the video prototype).

Once the language, the kind of visit -guided or not- and the user level have been selected, the visitors begin the site tour. As they approach the fortress, the device will
show the reconstruction of the point where they are, recognizing the archaeological
time to which it dates back giving the possibility to change this time if they are
interested, being able to carry out a diachronic comparison between them. Also they
will be able to obtain additional information at any time that they think suitable, as
well as to find their exact position on a map of the fortress.

6.2.1 Multimodal Interaction Style

Analysing the new situation we have met questions like these:
• the visitor is looking for information with the direct manipulation style. What
  happens if he/she moves around the fortress? Is his/her situation lost?
• alternatively, the visitor is situated at a specific point, can he/she accede to other
  information that especially interests him/her and that does not go with the current
  environment?

We propose a multimodal interaction style for the user consisting of the Direct
Manipulation interaction style [16] inside the Augmented Reality paradigm [17][18],
which allows both the use of Direct Manipulation as well as the use of Location in the
interaction in the course of a visit.

Direct Manipulation: it allows the user to choose which aspects of the site he/she
wants to explore and the possibility to be informed about other external elements of
the site; for example, he/she might obtain, during the visit, information related to the
culture of the Iberians and to know other Iberian sites if they are interested.

The use of the Location in the interaction: The location (position, orientation and
“archaeological” time) is an attribute of the Augmented Reality that gives us the
information of the scenario where the user is. This means that once the user is located
at a specific point of the site, the device is able to determine the position and the
orientation where he/she is.

Once this has been obtained, the device reconstructs the relevant scenario and
determines what information it can show (in the case of one being interested).

So we depart from the Desktop Paradigm with the Direct Manipulation interaction
style to go on to the Augmented Reality Paradigm.

6.2.3 Design: Task Analysis

The task analysis can be defined as the study of what a user must do in terms of
actions and/or cognitive processes to obtain an objective [19]. It is a methodology that
is supported by a set of techniques to help in the information collection process, to
organize it and to use it for decisions in the design.

A task is the activity necessary to obtain an objective, and this is the state or profit
that the user wants to reach within an application. For example, an objective could be
to know the details of a wall.

In the tasks analysis process there are two phases. The first one is information
collection, as we have already seen throughout this paper, helped at any moment by
archaeologists, and the second phase is the modelled one where all the obtained
information will be represented.

The Hierarchical Task Analysis (HTA) developed by Annett and Duncan [20] is
the most well known tasks analysis technique. It is a graphical method where the tasks
decomposition or information that we arrange in subtasks is represented. This
structure will give us an organization of the amount of information that we have gathered, allowing us to know the site in all its aspects and with all their influences.

The information description is represented as a tree diagram that describes the relations between tasks and subtasks based on the necessities.

The resulting hierarchical analysis will be an extensive structure, due to the amount of information that will be obtained from the site.

Once the hierarchical information representation has been done we will be able to make the dialogue model with which we will interact with the information.

Fig. 4. Hierarchical Task Analysis.

6.2.4 Functionality Design: Architectural Model.

We start from a site map in which we will place a collection of points –not physically visible– that will be defined in situ and which will correspond to what might be a future visitor’s situation in front of an element of the site or a general interest point, such as the wall, the ditch, etc. as shown in Fig. 5.

The location of these points will be studied in depth, as for every point the radius or the influence zone around it will have to be decided, in such a way that the total set of points and their corresponding influence zones completely cover the site map. In this way the visitor will always be under the influence of a specific point. With the user’s orientation inside the above-mentioned point, the system will determine what visualization of the environment will have to be shown.

Fig. 5. Example about the Site Map with some Points of Interest indicated and its zones of influence definition: two points in the site.
It is evident that the distribution and location of the points in the map, as well as their respective influence areas will depend on the zone of the site. A typical example is the exterior zone of the site –specifically the one that includes the ditch–, which on not having a diversity of elements that might be of interest, we will have fewer points to locate, and these, therefore, will have a larger influence area. On the other hand, in the interior zone of the fortress the points will be near each other due to the variety and quantity of architectural elements.

An important aspect to take into account is that an object has six degrees of freedom through which it can be moved, which correspond to the position and orientation of this inside a simulation. The first three degrees correspond to the position of the object inside the virtual world and the other three to the orientation movement [14].

![Fig. 6. Six degrees of freedom representation.](image)

This peculiarity means that every located point on the map has in turn different orientations where a user might be, therefore if we know the point and the orientation where the user is we will know the observation point on which it is acting and on which the system gives him/her the corresponding information. Every orientation can have in turn different views, but in order to simplify we have considered a single view with each orientation.

For the amount of points that we will define and for all their respective orientations we can deduce that we will obtain a large number of observation points.

To clarify this concept we suppose a point with a defined influence zone (C) located on the map as shown in Fig. 7. From this point we get 5 observation points (O. P.), which correspond to all 5 possible orientations that a user can take at this point and that fit the 5 elements of the site that we will be able to visualize and to obtain information about. These elements are: diverse Towers, the wall and a house. In this case, from the point, we get 5 O. P., but these change depending on the zone where the point is located, that is to say, the number of orientations of a single point changes depending on the situation in which it is.

For every observation point we will define a structure that will allow us to obtain all the information relating to it, as well as a visualization of the environment in any of its different phases.
Supposing that a user is situated at a specific point, such as Fig. 8 shows, orientated in such a way that he/she observes the wall and the “chevaux-de-frise”, in other words we already know his observation point.

What information might be of interest?
- The Wall and the Doors.
- The “chevaux-de-frise”.
- Other defensive systems.
- Construction materials.

The defined structure will help us to obtain all the possible information to which the user can accede. The structure is connected to the dialogue model, which provides us with all the information. In Fig. 9 we observe what such a structure would be like.
6.2.5 Conceptual Model.
The conceptual model is an extern abstraction describing, by means of formal diagrams and notations, the knowledge about the system that a user must have. This conceptual model reflects the most relevant aspects of an interactive system (feeling, events, communication, etc.) and must be complete, consistent and precise [21].

At this point a fundamental aspect is the dialogue structure or the user’s communication and a model for a correct specification is advisable.

The States Transition Network (STN) [19] is the most used method for dialogue modelling. We can represent the possible states of the system as well as the transitions among them.

![Fig. 10. Part of the Dialogue Model.](image)

The result is a network formed by nodes and connections from the transitions diagram, where the nodes represent the possible states of the system and the connections are the possible transitions between states.

The big peculiarity in our proposed system is that the location changes that Augmented Reality gives to the system will enable the transitions between the nodes.

6.2.6 System Representation: Augmented Dialogue Model.
At this point we have defined all the observation points by means of a structure that allows us to know all the available information.

Moreover, we have also defined the dialogue model corresponding to the Direct Manipulation applying the Desktop paradigm.

The final Architecture, called by us “Augmented Dialogue Model”, enables the connection between both interaction styles, connecting the information of the observation points with the dialogue model. This architecture is reflected in Fig. 11.
The place where both styles are connected will be formed by a three-dimensional structure. On the one hand we will have a model dialogue that will allow us to change from one state to another by means of transitions, on the other we will have a structure that will have all the information that we could know about an observation point.

The connection between the dialogue model and the observation point’s structure is made by means of pointers.

Let us suppose the previous example where the observation point was in front of the wall and the “chevaux-de-frise”. In Fig. 12 we observe a pointer between the context “wall” of the O.P. and the dialogue model having the wall as the objective. Thanks to this pointer we can be moved by the model with the purpose of getting the needed information and always maintaining the connection between both styles without losing our current situation.
This three-dimensional structure will be very large due to the amount of observation points that will have been defined and can become very entangled because of the mentioned pointers, but *the most important factor was to find the way to represent the information and the dialogue maintaining all the consistency and always bearing in mind that the system will be used by real visitors.*

Fig. 13 helps us to understand the final architecture with which it works.

The collection of points is the location on the map of the points with their respective directions, that is to say, the *observation points.*

This collection of observation points gives us the structure that contributes the possible information to get from each one of them.

The Dialogue Model gives us the states that can be given in the system.

Finally, the pointers allow us to make the connection between the collection of observation points and the Dialogue Model.
Although this structure is simplified the amount of pointers that we will need in its real size will be high.

7 Conclusions

We are moving from a Desktop paradigm to an Augmented Reality one. The current state of the art of Human-Computer Interaction is centred on the Desktop paradigm and the models developed up to now are focused on it.

In this paper we have presented the visit to a real archaeological site that serves us as a perfect test bench to study how this change of paradigm needs a change of the models and how this change—with the direct manipulation dialogue style—has to be carried out. Therefore, we propose a new context aware Dialogue Model (we have called it “Augmented Dialogue Model”) that includes the location as a new attribute.

We have analysed this new paradigm and made our proposal about how the information has to be structured and how the dialogue model has to be approached, giving a solution that constantly preserves the system’s consistency. Moreover, this
analysis always centres the attention on the user—Usability and Accessibility (the visitor in our sample) and for that we also have proposed our methodology. We are aware that the work is still unfinished—new attributes will probably appear—but we have covered a lot of terrain given that the model at this point has been fully developed.

References

Combining compound conceptual user interface components with modelling patterns – a promising direction for model-based cross-platform user interface development

Erik G. Nilsson, SINTEF Telecom and Informatics, Norway

Abstract. In this paper we examine why model-based user interface development languages and tools only have had a limited dissemination outside the research communities, and argue that there will be an increasing need for cross-platform user interface development in the future. To meet these needs, user interface development languages and tools must use new approaches. We examine some alternatives, and conclude that an approach based on pattern-based abstract compound user interface components as building blocks is the most promising. We describe this approach in some detail, and give an example showing how three quite different instantiations of one modelling pattern may be mapped to different running user interfaces using a number of mapping rules to two different implementation platforms with significant differences. Then we discuss what is needed for modelling languages and tools following the described approach to be successful, and we give some concluding remarks.

1 Introduction

Model-based user interface development [15, 19, 23, 27, 28, 29, 30] has an unexploited potential – the principles hold possibilities to obtain large gains, but these gains have not been fulfilled by any language or tool yet. Some years ago, the user interface modelling field was trying to solve very small or non-existing problems. Now, important real-world problems have emerged for which user interface modelling is indeed an appropriate solution – but the community is still trying to solve the problems using the same approaches as the ones used to solve the non-existing problems. Expressed in a less polemic way, one may say that the increased focus on mobility and ubiquitous computing has on the one hand made user interface modelling much more important than before. But on the other hand it has made the problems that need to be solved much more complicated. This causes a need for new approaches to user interface modelling.

In the late eighties and early nineties, the market focus on cross-platform user interfaces was quite high, as many organizations was in a transition phase from terminal based to graphical user interfaces. During the nineties, the market focus on and needs for cross-platform user interface development decreased as MS Windows emerged as the standard user interface platform on desktop computers. The last years,
one may say that cross-platform user interface development face a potential renaissance – and the focus will be even stronger in the future.

One of the most important trends that are likely to enhance the needs for cross-platform user interface development is user mobility. Access to and dissemination of powerful ultra portable equipment used in high capacity wireless networks, enables users to operate far more nomadic than what is feasible with mainstream equipment today. This trend will cause increased variety in what type of equipment users will exploit. This applies both to equipment exploited by a given user in different situations, but even more to an increased degree of variety in types of equipment used by different users. It will no longer be possible to presume that most users have equipment based on the same technological platform (which has been the case for desktop computing the last five to ten years). Thus, there will be an increasing need for developing user interfaces that must be available across a set of machine platforms, user interface styles and / or a varying set of modalities, etc. [16].

Another trend that supports this is a move to increased flexibility in how work is performed. Firstly, the physical location on which the work is performed gets more diverse. Secondly, the division between work and leisure time becomes less distinct – i.e. work can be done on more diverse moments in time. Both these aspects of increased work flexibility cause a need for accessing applications and services (both used in work and private contexts) in various situations, exploiting different versions of the user interface for the application/service exploited. A way of seeing this is that where the user is and what the user does to an increasing degree is uncoupled.

The challenge addressed in this paper is to have specification languages and tools that render it possible to have common specifications when developing user interfaces (UIs) that are to run on user interface platforms with significant differences regarding screen size, available modalities, and/or available user interface components. This means that the specifications should at least cover common specifications of UIs on traditional GUI platforms, on Web platforms, on Personal Digital Assistants (PDAs) and on mobile phones. It is also a presumption that the languages and tools shall facilitate specification of UIs that are “richer” and more dynamic than what is possible using HTML/XML technology today [16, 14].

Further on in this paper, we examine why model-based user interface languages and tools fail to meet these new challenges for cross-platform user interface development and outline an approach which has the potential to solve them. By this we also propose a direction for the user interface modelling field to move in the future.

2 Problems with model-based user interface development languages and tools

Recently, work within user interface modelling has put an increasing focus on modelling mobile user interfaces [5, 6, 11, 12, 18, 21]. Despite this, model-based user interface development has an unexploited potential as a means for supporting cross-platform user interface development. The main reason for this is that the conceptual level offered in the languages and tools is too low and that the building blocks are too simple.
Offering abstractions based on simple building blocks (like buttons, text fields, menus, etc.) are done on different abstraction levels. At a low level, a language may offer a radio group concept that is an abstraction of a set of radio buttons in a frame component on different implementation platforms. At a higher level, a language may offer a choice element concept that is an abstraction of radio group, drop-down list box, list box, etc. On different implementation platforms only a number of the concrete components may be available, but as long as at least one of them are available, the abstract specification may be mapped to the platform.

Using this scheme, a user interface specification is an instance hierarchy of such modelling constructs on the given abstraction level. This works well as long as the same instance hierarchy is applicable on all the platforms. Problems arise when this is no longer the case. If the specification is to work across platforms with a certain level of differences – e.g. with large differences in screen size – there may be a need to have different instance hierarchies on each platform.

One way of handling this is to divide the specification of a given user interface in two parts, one describing the commonalities across the platforms and one describing the specialities on each platform. Of course, there will one common model and a certain number of platform-specific ones. In many languages and tools, e.g. in UIML [26] – one of the languages with broadest platform support – this division between the common and the platform-specific models must be done at a quite early stage in a user interface specification [15]. Furthermore, the amount of specification code constituting the platform-specific parts tends to be much more voluminous than the common part. In such a situation, it is relevant to question whether a model-based approach gives any benefit over the most relevant alternative, which is to develop the user interface on each platform from scratch [16].

A second problem with most model-based user interface modelling languages and tools is a lack of usability in user interfaces generated from the models [23, 17, 34]. Part of the problem is that the user interfaces do not exploit all specialities available on the target platforms. Abstractions across a number of platforms tend to use concepts that are closer to the intersection than to the union of the possibilities on the different platforms. A related topic is to what degree the concrete user interfaces adapt to the look and feel standard on the various platforms. It is argued that solutions developed across platforms to a lesser degree follow the look and feel standards on the individual platform than solutions developed from scratch.

A third problem with most model-based user interface modelling languages and tools is that they are fairly restricted with regards to which type of user interfaces that may be expressed. A common restriction is to forms-based data base applications [2, 3, 10, 13, 20, 22].

3 Approaching a possible solution

The main challenge for a modelling language that shall work across platforms with significant differences is to have abstraction mechanisms that combine being general enough to cover the different interaction mechanisms on the different platforms, yet being powerful enough to act as a meaningful specification for the concrete platforms. Thus, there is a need for a combination of generalization and specialization.
A model-based language that is abstract enough to be able to describe UIs with significant differences may run the risk of being *banal*. By this we mean that the model is not able to describe sufficient number of aspects of the user interfaces in a way that renders it possible to transform the models to concrete user interfaces without adding so much additional information to the mapping process for each platform, that the interfaces might as well have been developed from scratch on each platform. Which indeed is the same problem we described above for languages and tools on a lower abstraction level. To avoid this pitfall we need something in between abstractions of fairly similar “atomic” components and models that are abstract in the sense that they are stating that there exist some dialogs, but not much about the contents of them. One approach may be to use *compound* user interface components.

Compound UI components on a concrete level work very well as a means for *reuse*. But as with reuse in other contexts, it is only useful when the reuse problem is the same as or a specialization of the original problem that the component was design to solve – or it solves some subtask that is orthogonal to the main problem (like a file open dialog). I.e. if we have implemented a compound component for handling time sheets, this component will hopefully work well in different applications where the users are to report time expenditure, but cannot be used in an application for reporting expenses, even though the underlying model may have a quite similar structure.

This example points towards the need for patterns support – as the similarities between the applications may possibly reveal a common model structure. So, the solution is maybe user interface patterns [4, 9, 24, 25, 31, 32, 33]. Most work on user interface patterns operates on a concrete realization level, and is usually quite guidelines-oriented (i.e. if you have this type of problem, you may use one of a number recommended designs – usually for an isolated subpart of a user interface design). In this way the use of user interface patterns is closer to the original architectural use than to the conceptual modelling and systems design use [8] – which has proved to be more successful than the architectural use(!) For our need in a user interface modelling language, general conceptual modelling patterns turn out to be more useful than concrete user interface patterns.

### 4 The proposed modelling approach

Our proposed solution – which has some similarities to work by Trætteberg [25] – is to use abstract patterns-based compound (or composite) user interface components as building blocks in user interface modelling languages and tools. The core concept in this is *modelling patterns*. As a modelling pattern usually involves a number of objects, a user interface supporting a modelling pattern must be a *composition* of different user interface components (each being simple or composite). As the user interface supporting the modelling language shall work on a number of user interface platforms, the compound (or composite) user interface components must be at a certain *abstraction level*.

This approach as it is just described is banal in the sense banal was described in the previous section. To solve this, transformation rules from the abstract, compound components to a concrete implementation on the different deployment platforms must be part of the modelling framework. It is essential to stress that these transformation
rules describe how the *modelling patterns* are to be realized on various platforms. This means that the transformation rules must be instantiated with the same concrete objects or classes that the patterns are instantiated with.

To utilize the potential of this modelling approach, it must include a number of different mapping to concrete representations for each abstract compound user interface component on each platform, both based on preferences, desired user interface style, modalities, etc. Fig. 1 shows how the different main parts of the modelling approach are connected – expressed using a Unified Modelling Language (UML [7]) class model.

![Fig. 1. Connections between the concepts in the modelling approach](image)

To illustrate how the modelling approach may work, we look at a simple example. A typical modelling pattern that should be supported in a modelling language is the *Composite pattern* [8]. Fig. 2 shows this pattern. The pattern consists of a generalized *Component* that may either be a *Composite* component of a *Leaf* component. The Composite component may have a set of children, which are Components – i.e. either Composite or Leaf components. The *children* aggregation is recursive in an arbitrary number of levels.

![Fig. 2. The Composite Pattern](image)

This modelling pattern may be used e.g. for modelling a file system (see Fig. 3), a mail system (Fig. 4), and a department structure in a large organization (Fig. 5).
Fig. 3. File system instantiation of the Composite pattern

Fig. 4. Mail system instantiation of the Composite pattern

Fig. 5. Department structure instantiation of the Composite pattern

The user interface model for each of the pattern instances is the instantiation of the pattern, choices of which platforms the pattern shall be implemented, and also which of the available mappings that are to be used on each platform. Before we look at how each instantiated patterns will be transformed to a running user interface, we present examples of how the mapping rules could be for the given pattern.

For traditional GUI presentation one mapping could be to present the Composites and the Composite children in tree view to the left and to present all children of a node selected in the tree in a list view to the right. Details about the Leaf children are
presented in a separate window (e.g. using a forms based presentation) when the user double-clicks on Leaf items in the list view. Fig. 6 shows this mapping scheme.

![Fig. 6. First mapping scheme for GUI presentation](image)

An alternative mapping for GUI presentation could be to present the Composites and the Composite children in tree view to the left. On the top right, only Leaf children of a node selected in the tree are presented in a list view. On the bottom right, details for the node selected in the top right list view are presented (e.g. using a forms based presentation). Fig. 7 shows this mapping scheme.

![Fig. 7. Second mapping scheme for GUI presentation](image)

On a PDA platform one mapping could be to present the Composites and the Composite children in a tree view (occupying the whole screen), while the Leaf children are presented in a list view (also occupying the whole screen). Selecting an item in the list view opens this item in a new view (also occupying the whole screen). Fig. 8 shows this mapping scheme.
Fig. 8. First mapping scheme for PDA presentation

An alternative mapping for the PDA platform could be to present all Components on one level of the tree structure in a list view (occupying the whole screen). Navigating down in the tree structure is done by selecting a Composite item in the list view (which causes all its children to be shown in the same list view). Navigating up in the tree structure is done with a dedicated drop-down list-box showing all ancestors of the items in the list view. Selecting a Leaf item in the list view opens this item in a new view (also occupying the whole screen). Fig. 9 shows this mapping scheme.

Fig. 9. Second mapping scheme for PDA presentation

Given these (and of course a number of other) mappings on these (and possibly other) platforms, each of the instantiated model patterns may now be mapped to a desired number of arbitrary mappings (often only one per platform) on the desired platforms (mostly more than one).
In this way the file system instantiation of the model pattern may be realized using the first mapping on the GUI platform and the second mapping on the PDA platform. Similar, the mail system instantiation may be realized using the second mapping on both platforms, while the organization model instantiation may be realized using the second mapping on the GUI platform and could be available in both mapping on the PDA platform – based on user preferences.

Fig. 10 shows how the file system implementation may look on the GUI platform, while Fig. 11 shows it on the PDA platform (details views are omitted). Fig. 12 shows how the organization model implementation may look on the GUI platform, while Fig. 13 and Fig. 14 show the two alternative implementations on the PDA platform. The detail view for the chosen Employee in Fig. 14 is omitted as it is identical to the corresponding view in Fig. 13 (the rightmost screen dump).
Fig. 11. PDA mapping of File System instantiation (detail view omitted) – right view shows navigation to ancestors

Fig. 12. GUI mapping of Organizational model instantiation
In the description and examples above, we have focused on how the abstract compound user interface components are composed and the structure of various mappings to running user interfaces. It is also important to mention that part of a mapping description is also the generic functionality in the implemented user interface. By this we mean e.g. how selection and double click on elements work, drag and drop support, other direct manipulation features, functionality on table headings (sorting, width adjustment, sequence changes, etc.).
The instantiation of a model pattern also includes determining necessary visual and functional details like icons to use (e.g. in a tree view), column headings, sorting of lists, menus, toolbars, layout style for detailed presentations, mapping from data types of attributes to presentation styles, actions for element etc. These issues should as much as possible be part of the pattern instantiation, not of the mapping process (and thus be specified on a conceptual level) – but in some cases it may be necessary to give some details of this kind specifically for a given mapping on a given platform.

Modelling languages and tools following the approach described in this section also need to include mechanisms for how the abstract composite components (i.e. the pattern instances) may be coupled to build user interfaces consisting of more than one pattern instance. Preferably, the coupling mechanism should also be based on modelling patterns, and ideally it should be recursive, where the top level is the dialog structure of the application.

5 Discussion

The goal of the approach presented in the previous section is that by using pattern-based abstract composite user interface components with connected mapping mechanisms it is possible to have a common specification of a UI that work across platforms with significant differences – without having to have major amounts of extra specification for the variant on each platform. This solves the main problem with model-based user interface languages and tools today as outlined earlier in this paper. And an extra benefit is that the approach also solves the two other problems outlined.

As both the number of abstract, compound components (i.e. supported modelling patterns), the number of mappings for each pattern, and the number of implementation platforms are limited, it is possible to optimise the mappings with regard to usability and exploiting special features on each platform. In this way, the approach solves the usability problem encountered in user interfaces generated with many model-based languages and tools today – which was the second problem outlined.

As showed in the examples in the previous section, the presented approach may be used to implement more advanced user interfaces than simple forms-based ones. It is also feasible to use the approach to specify and implement e.g. a map-based user interface with direct manipulation, by coupling elements from the pattern instantiation to behavioural rules in the map-based mapping mechanisms. This shows that the approach also solves the third problem outlined.

In the previous section, it was mentioned that it is possible to apply more than one mapping to a given platform for a pattern instance – and base the actual presentation on user preferences. A natural extension of this is to use the same scheme to realize an adaptation mechanism.

To make the proposed modelling approach successful, it is of critical importance to find the right set of abstract patterns-based compound user interface components. It is also very important to define a good set of mappings for each pattern to the chosen set of implementation platforms – but this is much easier. Choosing the appropriate set of platforms is more of a market than a technical issue.

As mentioned, the main challenge is to find the right set of abstract components. It must be a limited set to make the modelling language comprehensible and to limit the amount of work needed to define all appropriate mappings. Yet the set must be
sufficient comprehensive to render it possible to use the modelling language to
specify an arbitrary user interface. It is of limited help to be able to specify and
implement the wrong user interface very efficiently on a vast variety of platforms.

A way of making this choice less critical is to make the modelling language
expandable - both regarding new modelling patterns/abstract components and how
they are mapped to implementation platforms.

6 Conclusions

If we consider a probable future development, we foresee that a combination of
increased bandwidth available for wireless communication, increased requirements on
security, a huge multitude of mobile devices, a fairly large variety of client platforms,
and enhanced maturity of new modalities will increase the needs for developing
applications and services that must be able to run on a number of quite different
platforms. This will cause an increased need for languages and tools that support
cross-platform user interface development.

For model-based user interface development tools to meet these needs, different
approaches than the ones used today are needed. Basing user interface modelling on
simple (yet abstract) concepts may work quite well across platforms that are quite
similar, but when the user interface models are to work across platforms with
significant differences (regarding screen size, available modalities, and/or user
interface components) this scheme falls apart. The result in many modelling
approaches is that the abstract models must be supplemented by additional platform
specific models describing how the user interface are to be tailored to the various
platforms. Even worse, these platform specific models tend to dominate the
specification with regard to the specification volume.

To solve this, and a number of other problems with model-based user interface
development languages and tools, we have proposed a modelling approach using
abstract composite user interface components based on conceptual modelling patterns.
Supplementing these pattern-based, abstract, compound user interface building blocks
with a number of well defined, usability-optimised mappings to a set of
implementation platform (as part of the modelling scheme) it is possible to have
models that are both conceptual and powerful. These models will work as
specifications towards platforms with significant differences without having to
supplement the abstract models with a lot of platform-specific models.

It is our hope that this paper will serve as an inspiration for the user interface
modelling community when determining directions for new languages and tools for
model-based user interface development.

Acknowledgement

The paper is partly based on results from the MOGOP project, which is funded by the
Norwegian Research Council and Genera A/S.
References

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995


H. Trætteberg: Model based design patterns, Position paper for CHI2000 workshop on user interface design patterns. Available at http://www.idi.ntnu.no/~hal/publications/design-patterns/CHI00-position.pdf


J. Vanderdonckt and A. Puerta: Computer-Aided Design of User Interfaces II - Proceedings of the Third International Conference on Computer-Aided Design of User Interfaces, 1999


Multiple User Interfaces: Towards a Task-Driven and Patterns-Oriented Design Model

Ahmed Seffah¹ and Peter Forbrig²

¹ Human-Centered Software Engineering Group, Department of Computer Science, Concordia University
1455 de Maisonneuve Blvd. West, Montreal, Quebec, Canada H3G 1M8
E-Mail: seffah@cs.concordia.ca

² Software Engineering Group, Department of Computer Science, University of Rostock
Albert-Einstein-Str. 21, 18051 Rostock, Germany
E-Mail: pforbrig@informatik.uni-rostock.de

Abstract. The convergence of the Internet, mobile telephony, and handheld technologies has led to the emergence of new kinds of internet-based interactive systems (IBIS). Such systems can allow a single or a group of users to interact with the server-side services and information using different kinds of computers and devices including palmtops, mobile telephones, etc. In this technological context, a Multiple User Interface (MUI) refers to an interactive system that provides both multiple views of the information and coordinates the services provided to a user. The information and services may reside on a single server, or may be distributed among several independent and heterogeneous systems. The desired views are made available on different platforms – operating systems, user interface toolkits and on a large array of devices. Each view should take into account the specific capabilities and constraints of the platform while maintaining (promoting) the cross-platform usability of the MUI. This paper begins by describing a set of constraints and characteristics intrinsic to multiple user interfaces, and then by examining the impacts of these constraints on the specification, design and validation processes. Then, it describes the research opportunities in important topics relevant to MUI development and usability including cross-platform usability, adaptation, task model-based and pattern-oriented design.

1. Introduction

Multiple user interfaces are proliferating in a variety of fields, such as cooperative engineering, e-commerce transaction, on-site equipment maintenance, remote software deployment and support, contingency management and assistance, as well as distance education and telemedicine. For example, a civil engineer can use a PDA for gathering information when inspecting a new building. Then, a mobile telephone can be used for adding comments and faxing or uploading all information gathered to the headquarter office. Finally, the same engineer or any other employee can use an office laptop to analyze the data and prepare a report. During this workflow, the engineer interacts with the same information and service using different variations of a single
interface. These variations can support different "look and feel" and to a certain extent, different interaction styles. This different "look and feel" should take into account the constraints of each device, while maintaining cross-platform consistency. McGreere [11] summarized the usage of a MUI as follows:

“One can imagine having multiple interfaces for a new version of an application; for example, MSWord 2000 could include the MSWord 97 interface. By allowing users to continue to work in the old interface while also accessing the new interface, they would be able to transition at a self-directed pace. Similarly, multiple interfaces might be used to provide a competitor’s interface in the hopes of attracting new customers; for example, MSWord could offer the full interface of a word processor such as Word Perfect (with single button access to switch between the two), in order to support users gradually transitioning to the Microsoft product.”

However, even if the runtime infrastructure and the related development environment for developing the MUI on each platform is already available or will be in the near future, the following are the major design, specification and validation questions that should be addressed both by academia and industry researchers:

- Should we strive for uniformity in the services offered, dialogue styles and presentation formats, or should we adapt the interfaces and services to the constraints and capabilities of each device and/or each context of use?
- When designing multiple user interfaces, what is the best way to take the constraints related to each type of device into account while assuring the maintainability and cross-platform consistency of this interface mosaic?
- How can we make it possible for users to customize a multiple user interface device they are using in terms of accessibility to resources, tools, services, etc? When it is done from one device, how can such customization be reflected on all other devices that the user can use?
- How can we implement and validate N interfaces for M devices without writing N*M programs, training an army of developers in L languages and UI toolkits, and maintaining N*M architectural model for describing the same UI?

This paper is an attempt to clarify these questions. It is in three parts: a characterization of the essence of MUI; a brief summary of early results obtained during a workshop we organized [17], and a guided tour of fertile research topics awaiting investigation.

2. Multiple User Interface – Definition and Characterization

We introduced the concept of MUI during the HCI-IHM workshop [17]. The term multiple user interface is being used by others [11] and [20]. Intuitively, a MUI:

- Allows users to interact with the server-side services and information using different interaction/UI styles. For example, our requirement engineer uses pen and gestures for a PDA, function keys or single characters for a mobile phone and direct manipulation for desktop.
- Allows an individual to achieve a sequence of interrelated tasks using different devices. For example, a requirement engineer can use a mobile telephone on the road to confirm an appointment, use his desktop to email information about the
interview, use his PDA to gather information about the user needs when he is interviewing stakeholders and finally use his laptop/desktop computer at to synthesize the information and write the requirement report.

- Presents features that behave the same across platforms, even though each platform/device has its specific look and feel.
- Feels like a variation of a single and cross platform interface for different devices with the same capabilities.

Fig. 1 shows a multiple user interface to the same Internet financial management system. This interface consists of three views for a desktop, a PDA with a keyboard and mobile phone. From the user perspective, these three different interfaces should be the same. However, this is not realistic because the capabilities and constraints imposed by each platform. The multiple user interface here can be seen as compromise between customized and platform-dependant UI. The effort required to keep all interfaces consistent and to maintain them, as the functionality of the underlying financial system is expanding, increases linearly with the number of interfaces.

![Fig. 1. An example of a MUI](image)

More formally, a Multiple User Interfaces (MUI) refers to an interactive system that provides both multiple views of a model and co-ordinates the user actions gathered from different devices/computers. The model may reside in a single information repository, or may be distributed among independent systems. Each view can be seen as a complete user interface for a specific platform (hardware, operating system, user interface toolkit) because all devices are able to present all features of the system. However, all these mosaic interfaces form a unique and single multiple user interface. The interaction style and the displayed information/feedback can vary from one to another platform.

2.1 Multiple User Interface Styles

There are, at least, three user interface styles that a MUI may implement:

- GUI Style or WIMP interfaces.
  This style, the most popular and dominant, employs four fundamental elements: windows, icons, menus, and pointers.

- A Web browser-based user interface (WUI).
  The Web was originally conceived as a hypertextual information space; but the development of increasingly sophisticated client and server sides’ technologies has fostered its use as a remote software interface. A WUI is generally a mixture
of markup (in say HTML, XML syntax), style sheets, scripts (in say
ECMAScript, VBScript) as well as embedded objects such as Java applets, plug-
ins. An application using WUI style information is typically displayed in
a single GUI window called a browser, though multiple browser windows can be
used by an application to display information. The browser provides basic
navigation. Different browsers for small devices and mobile phone are being
developed. Such browsers are able to display a customized version of a WUI.
The Yahoo Stock Manager is an example of a WUI that has a customized
version for mobile phones. The Web clipping architecture and WAP frameworks
provides basic services for dynamically generating customized HTML
documents that can be displayed on mobile phones
- Handheld user interface (HUI).

There are two major classes of PDAs in use today – those using a true GUI style
of appearance and behavior and those that use a GUI or WUI subset. Both
classes of UI employ a gesture-based interaction using a stylus and/or a touch
tablet. Even if it is not yet clear what style of UI will dominate handheld, the
use GUI and WUI models should be reevaluated because of the lack of screen
space and the low bandwidth.

We expect that in the near future designers will be asked to combine these three
different styles. Fig. 2 shows an example of a MUI that uses GUI and WUI styles.
GuruNet is a pop-up application to retrieve reference information (dictionary,
thesaurus and encyclopedia) and real-time information (e.g. news, sports, weather or
stock quotes) across the Internet from inside any application. GuruNet offers a Web
browser-based user interface and an optimized GUI version for PDA and mobile
phone. Dealing with different UI styles complicates both the development and the
validation of the MUI. Making a tradeoff between these different styles, when they
come into conflict, is not an easy task.

Fig. 2. GuruNet: an example of a MUI implementing different UI styles (GUI for PDA, WUI
for desktop, WUI for Mobile phone)
2.2 Characteristics and Constraints

The following are the fundamental intrinsic characteristics of a multiple user interface:

- **Abstraction**
  All information/services should be the same across devices of the same or larger category, even if not all information/services is shown or needed for all platforms. For example, a product listing may include only the best-selling items on a small narrowly device with the rest relegated to a secondary "more products" page. For an office desktop, the product includes all the selling items. We can use a different text color to highlight the selling items.

- **Cross-Platforms Consistency**
  A MUI can have different look and feel while maintaining the same behavior over different platforms. For example, all user preferences must be preserved across the interfaces. For example, if the end-user uses one access mechanism today and another one tomorrow, then the changes made in one user interface are reflected in the other.

- **Uniformity**
  A MUI should offer the support of the same functionality and feedback even if certain special features or variations are eliminated in some platforms [14]. For example, an airline reservation system may make choosing a flight and buying the ticket in two separate steps. This separation should be preserved in all versions instead of having the simplified version unify choosing and buying into a single step.

- **User Awareness of Trade-Off**
  It would be OK to have an advanced version including additional features (such as specifying a seating preference) that were not in the simplified version. Missing these features is a trade-off that the user would make in return for the benefits of being able to use the system under various limited circumstances.

- **Conformity to Defaults Standards**
  It is not necessary for all features to be available in all access mechanisms. For example, a PDA interface may eliminate images or it may show them in black-and-white. Similarly, text may be abbreviated on a small display, though it should be possible to retrieve the full text through a standardized command.

These characteristics and constraints are not artifacts of current development technologies, but are intrinsic to the MUI concept. Together, they characterize a MUI and complicate its development. They also justify the need to rethink the existing approaches to usability engineering.

3. Relevant Background Works

Remarkably, though focused research on multiple views and multi-devices interaction can be traced to the early 1980s. There are relatively few examples of successful implementations [9]. Perhaps the main cause of this poor success rate, are the difficulties of integrating the overwhelming number of technological,
psychological, and sociological factors that affect MUI usability into a single unified design.

In the evolution of user interface ranging from the hardware to the concept of a multi user interface [9], the fifth stage (1990s to present) is the most relevant for our discussion. A multi-user interface supports groups of devices and people cooperating through the computer medium. A MUI can be considered as a multi-user interface. Our single user in the context of a MUI is what a group is for multi-user interfaces. He or she is asynchronously collaborating with himself. Even if a user is physically the same person, he can have different characteristics while working with different devices. For example, a mobile user is continuously in rush, impatient and he cannot wait [14]; he needs immediate, quick, short and concise feedback. The same user, when he is in his office, can wait a few seconds for more details and explanations.

Another important relevant work to MUI is the concept of context-sensitive or oriented user interface. This area is still an active research domain and many models are emerging. The concept of a plastic user interface [19] or moderator [20] are two promising models. In a recent essay, Winograd also compared different architectures for adapting a user interface to the context of use [21]. As we characterized it in the previous section, a MUI is a context-sensitive user interface. This does not mean that a MUI should adapt itself magically at runtime to the context of use and in particular to the platform capabilities and constraints. It can be adaptive or adaptable. As we will discuss it in the next section, the adaptation can be done by the end-user or developer during the specification, design, and development, or before or after the deployment.

Compound document is also a useful technology that can support the development and integration of the different views that a MUI offers. A compound document framework can act as a container in which a continuous stream of various kinds of data (components) can be placed [15]. In certain extent, a compound document is an organized collection of user interfaces that form a single integrated perceptual MUI. Each form of content has associated controls that are used to modify the content in place. During the last decade, different frameworks have been developed such as Andrew, OLE, Apple OpenDoc, Active X and Sun Java Beans. Compound document frameworks are important for the development of a MUI for different reasons. It allows the different parts of a MUI to co-exist closely. For example, it makes data from one to another be "active", unlike Cut and Paste. It also eliminates the need for an application to have a viewer for all kinds of data; we just need to invoke the right functionality and/or editor. Views for small devices don't have to implement redundant functions. For example, there is no need for Microsoft Word to implement a drawing program; views can share a charting program. Compound document frameworks can also support asynchronous collaboration between the different views and computers.

4. Fertile Topics for Exploration

We now turn to the discussion of promising research topics in MUI design, development and usability. By its very narrative nature, this section of the paper is
highly speculative and will raise far more questions than it answers. Further, this is a selective list of topics; it is certainly not intended to be exhaustive. Rather, our goal is to give the reader a tantalizing glimpse of the rich problem space defined by MUI.

4.1 Vertical versus Horizontal Usability

Vertical usability refers to the usability requirement specific to each platform while horizontal usability is concerned with the cross-platform usability issues of a MUI. What kind of tool can be used to support these two dimensions of usability?

Many system manufacturers have issued design guidelines to assist designers in developing usable applications. These guidelines can be categorized according to whether they prescribe a design model, i.e. “do this” or whether they discourage a particular implementation, i.e. “don’t do this”. Palm Inc. has put forth design guidelines to address navigation issues, widget selection, and use of specialized input mechanisms such as handwriting recognition. Microsoft Corporation has also published usability guidelines to assist developers with programming applications targeted at the Pocket Windows platform. For example Microsoft recommends giving “the user immediate and tangible feedback during interaction with an application. Appropriate feedback includes acknowledging a request, pointing out an error, or tracking the progress of an operation.” They also suggest “although auditory feedback can be useful for attracting a user’s attention, it should be used sparingly.” Such feedback is either broadly specified or too simplistic. Compounding this problem is the fact that in many cases it would appear that the usage of different guidelines could create many inconsistencies. Some guidelines can come into conflict and making a tradeoff not an easy task for MUI developers. The Java “look-and-feel” developed by Sun aims to be a cross-platform guideline that can fix the limitation of platform-dependant guidelines. However, this guideline does not take into account the particularities of each specific device, and in particular the platform constraints and capabilities.

To be consistent across platforms, an application does not hard-code its UI components for a particular look and feel. For example, the PL&F (Pluggable Look and Feel) is the portion of a Swing component that deals with its appearance (or look), is distinguished from its event-handling mechanism (its feel). When you run a Swing program, it can set its own default look by simply calling a UIManager method named setLookAndFeel.

4.2 Adaptation Strategies

Adaptation refers to the ability to tailor and optimize an interface according to the context in which it is used. Adaptation requires a MUI to sense changes about the context of use, make inferences about the cause of these changes, and then reacts appropriately. There are two visions of adaptation.

- **Adapting to technology variety**
  Technology variety implies supporting a broad range of hardware, software, and network access. The first challenge in adaptation is to deal with the pace of
technology change and the variety of equipment that users employ. The stabilizing forces of standard hardware, operating systems, network protocols, file formats, and user interfaces are undermined by the rapid pace of technological change. This variety results also in computing devices, in particular mobile phones that exhibit drastically different capabilities. For example personal digital assistants (PDA) have use a pen based input mechanism and have average screen sizes in the range of 3 inches. On the other hand, the typical PC uses a full sized keyboard, a mouse and has an average screen size of 17 inches. Coping with such drastic variations implies much more than mere layout changes. Pen based input mechanism are magnitudes slower than traditional keyboard and thus are inappropriate for applications such as word processing that require intensive user input. Similarly the small screens available on many PDAs only provide coarse graphic capabilities and thus would be ill suited for photo editing applications.

- **Adapting to context of use diversity**

A second challenge to broadening participation is the diversity of users [10]. Accommodating users with different skills, knowledge, age, gender, disabilities, disabling conditions (mobility, sunlight, noise), literacy, culture, income, etc., is the source of a further complication. For example, walking down the street, one user may use his mobile telephone’s Internet browser to lookup a stock quote. However, it is highly unlikely that this same user review the latest changes made to a document using the same device. Rather, it would seem more logical and definitely more practical to use a full size computer for this task. It would therefore seem that the context of use is determined by a combination of internal and external factors. The internal factors primarily relate to the user’s attention while performing a task. In some cases, the user may be entirely focused while at other times he may be distracted by other concurrent tasks. An example of this latter point is when a user driving a car, operates a PDA to reference a telephone number. External factors are determined to a large extent by the device’s physical characteristics. It is not possible to make use of a traditional PC as one walks down the street. The same is not true for a mobile telephone. The challenge to the system architect is thus to match the design of a particular device’s user interface with the set of constraints imposed by the corresponding context of use.

A fundamental question is when a multiple user interface can be or should be tailored as a single and unique interface. The range of strategies for adaptation is delimited by two extremes. Interface adaptation can happen at the factory, that is, developers produce several versions of an application tailored according to different criteria. Tailoring can also be done at the user’s side, for instance, by system administrators or experienced users. In the other extreme, individual users might tailor the interfaces themselves, or the interface could adapt on its own by analyzing the user’s context of use. As a major conclusion of the workshop we organized [18], we consider the adaptation of a MUI should investigated at the different steps of the entire design, deployment and exploitation lifecycle:

- **User customization during usage**
Tailoring operations are the entire responsibility of the user. While this laissez-faire approach avoids the need for system support, it lacks a central arbitrator to resolve incompatible and inconsistent preferences between devices. The arbitrator should have the ability to make global changes (cross-platform changes) based on local adaptations. This makes MUI more difficult to write, and fails to amortize the development cost of support for adaptation.

- **Automatic adaptation at runtime**
  The idea to write one UI implementation that magically adapts itself, at runtime, to any device or platform, is not a realistic approach. The drawback of this strategy is that there may be situations where adaptation performed by the system is inadequate or even counterproductive.

- **Just-in-time customization during development or deployment**
  Developers can use one high level language to implement an abstract and device independent user interface model. Then, they can use a renderer to generate the code for a specific platform. The user interface markup language (UIML) aims to support such approach.

- **Customization during design and specification**
  This needs to use an appropriate design methodology and a multi-platform terminology to properly build a task model of a MUI. This model may be expressed in one or different notations. Tailoring operations can happen at the abstract interface specification where the dialogue gets modified, for example to shortcut certain steps, to rearrange the order for performing steps, etc.

### 4.2.1 Model-Based Adaptation

A task model describes the essential tasks that the user performs using a user interface. A typical task model is a hierarchical tree with different subtrees of the tree indicating the different tasks that the user can perform. Task models are a very convenient specification of the way problems can be solved. They are able to specify different problem domains on different abstractions. Our early investigations show us that in the case of a multiple user interfaces, we should make a distinction between four kinds:

- General task model for the problem domain
- General task model for software support
- Device dependent task model
- Environment dependent task model

The general task model for the problem domain is the result of a very accurate analysis of the problem domain. It describes how a problem can be tackled in general. All relevant activities and their temporal relations are described. Such a model can be considered as the representation of the knowledge of an expert. The state of the art is captured within this model.

The general task model for software support contains activities that have to be executed by humans by principle or because of financial reasons. It contains all activities that have to be executed by a software system or in an interactive way. In some sense this model is the first design decision. Later on the behavior of the developed software has to be consistent to the specified behavior of this task model.
The capabilities of a device are the most restrictive once in the context of use of a software system. This is the reason why the relation between task models and devices is especially stressed. As we already mentioned there are approaches to transform whole applications from one platform to another one without looking at the tasks that will be supported. But sometimes it is wise to look at the tasks first and to decide which tasks can be supported in an optimal way by a special device. This information is captured in the device dependent task model.

The environment dependent task model is the most specific one. It is based on several design decisions in previous models and describes computer-supported tasks for a special device. This model describes the behavior of a system based on the available tools, resources and the abilities of the user. It can be interpreted statically (influences are fixed during design time) or dynamically (influences are evaluated during run time).

There are different possible scenarios for the usage of task models in a client/server environment, which is the case of Web applications with a MUI. Often such models are used implicitly only. The implementation is done in a manual way. But it is possible to generate software based on task models. The idea for the modularization of task models was described in [8]. It is based on the usage of process algebra and allows the specification of temporal specifications by equations [6].

![Ontology of a model-based approach](image)

**Fig. 3.** Ontology of a model-based approach

The kernel model describes the general approach of solving a problem with all flexibility. Constraints describing actual restrictions (available tools, user disabilities, etc.) are used to adapt the model to a specific situation. This adaptation can be done statically during generation or dynamically at run time.

If the resources on the client side are large enough then the interpretation of a task model can be done there. If there is a lack of resources the interpreter should run on the server side. In this case the client is informed about all tasks that can be performed in the next step. One of these tasks is selected and executed. The server is informed about this selection and computes a set of tasks that can be executed after this actual task.

In a client server environment there is the question whether the adaptation has to be performed on the client or on the server side. We will have a look at this problem for the task models at these various levels.
4.2.2 Models and Mappings

The most abstract model is the general task model for the problem domain. It is more or less for analysis reasons only. A usable model for a computer is the general task model for software support only. Allocating tasks to humans or machines is the first design decision. It results in the general task model for software support. This mapping can be done by attaching information to the nodes. This development step is omitted in this discussion. It is very similar to the transformation from the general task model for software support to the device dependent task model.

Tasks are attached to devices (more or less stereotypes or roles of devices only). This mapping specifies which type of device supports, which task. The restrictions are due to the input/output features of devices. They are not because of computational abilities. If the computational power is not strong enough then the computation can be performed on the server side. But because of small displays and keyboards it makes no sense to perform tasks with a lot of data transfer.

It is assumed that the stereotypes of the devices can be put into a "relation. This relation expresses the input-output capabilities of a device (e.g. Personal computer PC > Palmtop PT > Mobile phone MP). If the stereotype of a device is attached to a node this task and all nodes of the sub-trees can be performed on a corresponding machine. If a stereotype S is attached to a node then all devices belonging to a stereotype T with T > S can be used to perform the corresponding task as well. We assume that real execution of a task is performed in the leaves only. Let us have a look at an example.

The whole task model can be performed on a personal computer. If only a palmtop is available it makes no sense to perform tasks related N11, N12, N21, N23 and N31. You need the full screen of your computer. With a mobile phone at hand it makes only sense to perform N221, N222, N321, N322 and N323. These tasks can be performed of course using a personal computer or a palmtop as well. The model does not specify that you need a mobile phone it specifies that you at least need such a device.

This general device dependent task model allows an extraction of the specific task model for a palmtop and a mobile phone (see Fig. 5, Fig. 6). The question arises whether all mappings of tasks to devices are possible. This is not the case because temporal relations of the model have to be considered. If the successful performance of a task not attached to a device is a precondition of a task attached to this device the specification has an error of course. If task N1 is in the temporal relation of a sequence to task N2 the model for the palm top (Fig.5) can't be correct. Such kinds of mistakes can be detected by looking at the constraints of the model.
4.2.3 Task Models and Client Server Architecture

The task model can be interpreted on the client or on the server side. It makes sense to consider it as specification of a workflow. In this case the execution of a task model starts with the instantiation of the model. This instance plays the role of a repository or mediator. It controls the workflow in a CSCW application. One person co-operates via different devices.

The question arises whether one can continue in executing a task on a different device. Is it e.g. possible to continue writing an email on a mobile phone? It could make sense if the body of the mail is already written and only the address is missed. In this case the address could be added at that moment the mail has to be delivered. This can be initiated via a mobile phone.

Two scenarios are possible representing the extreme situations.

1. The task models are interpreted on the PC and on the mobile phone. The server is only informed about the state of the execution. It plays the role of a mirror or proxy.
2. The task model is interpreted on the server and the clients are used as a kind of display only.

For case 1 the server works as a mirror of the work of the clients. If a new device is switched on a view of the relevant tasks is downloaded to the device. This means that the state of the execution of the tasks is stored on the new device. If states are changed on the client the server is informed and vice versa. It depends on the temporal relations whether tasks can be performed or not. States of subtrees that are not supported by the device cannot be changed by this client.

In case 2 the server controls the execution of tasks. It has to consider which tasks are supported by which device and has to send the corresponding information to the device.

Two mixtures of both scenarios are also possible.

All approaches can be combined with the idea of specifying specific features of a device by separate models. This idea is discussed in [12].

4.3 Pattern-Oriented Design

Another perspective that can facilitate the development and validation of MUI is the concept of pattern-oriented designs. In the user interfaces design community, there have been vigorous discussions on usability patterns and pattern languages worldwide.
since 1997, and there are many groups devoting time to the development of patterns and patterns languages for different kinds of user interfaces. Amongst the heterogeneous collections of usability patterns, “Common Ground”, “Experience”, “Brighton” and “Amsterdam” play a major role in this field and have significant influence [19], [16].

For example, the Web Convenient Toolbar pattern provides direct access to frequently used pages or services. A typical Web convenient toolbar includes navigation controls for What’s New, Search, Contact Us, Home Page, Site Map, and so on. Fig. 7 shows two design solutions and implementations of this pattern for a Web browser and PDA platforms. For PDA, it is implemented as a combo box using Wireless Markup Language (WML). For a Web browser, it is implemented as a toolbar using HTML and JavaScript or a Java applet. The implementations should take into account the context of use including platform constraints and capabilities. This is why the combo box implementation is suitable for the PDA.

![Fig. 7. The Convenient Toolbar Pattern on a Web Browser and a PDA](image)

Patterns should not be considered just as an alternative design tool to guidelines. In the context of MUI development, we perceive them as a high-level tool for compiling experience gained through end-user feedback as well as for transferring, by means of software tools, the knowledge from usability experts to software engineers unfamiliar with usability principles. For instance, CASE tools have long been available to assist software developers to integrate the many aspects of Web application prototyping. However, the majority of them do not provide any mechanism for explicitly ensuring the usability of the developed application. In addition, it is not clear how usability knowledge should be integrated with existing development tools in order to maximize the benefits of usability patterns. Tidwell pointed out that the latest technology and design tools can help the designer to build much more creative Web applications than in the past. However, there is still room for improvement in the design process and in usability issues [19].

Given the wide variety of applications, usability pattern implementations should exist in various formats. An editor, should provide advice to pattern users in terms of selecting the suitable implementations for their context. Selection rules should be embedded in the tool.
Furthermore, rather than using different programming languages for coding the different implementations, we are investigating XML as a unified and device-independent language for implementing patterns. By using XML-compliant implementations, patterns can be translated into scripts for script-based environments like HTML authoring tools, beans for Java GUI builders like VisualAge, and pluggable objects like Java applets and ActiveX components. Generating a specific implementation from an XML-based description is now possible because of the availability of XML-based scripting languages. Among them, the User Interface Markup Language, UIML (http://www.uiml.org) is a potential candidate [1, 22]. UIML descriptions of a user interface can be rendered in HTML, Java and WML. Tools like the IBM-Automatic code generator [4] from design patterns encourage us to investigate the generation of code from XML-based pattern implementations as well as for automating the development of patterns-oriented designs. The UPade editor we are developing aims to support such design approach [22].

5. Conclusion

In the current technological context, the concept of Multiple User Interfaces (MUI) is essential. As outlined in this paper, an effective MUI design must consider both cognitive and social factors in addition to the usual technological constraints. Architectures that neglect these matters cannot effectively cater to the requirements of the different users. Unfortunately, adoption of a MUI application is contingent on the acceptance of all the stakeholders.

Research has put forth several approaches to assist developers in creating effective MUI designs. Generally, these methods work well for regular software development and have thus been adapted for the case of multiple user interfaces. However, this approach usually results in tools that do not capture the full complexity of the task. Pattern hierarchies seem to be an exception to this finding. Whereas an individual pattern provides a solution to a specific problem, organized as a hierarchy, patterns guide the developer through the entire architectural design. In this way, they enforce consistency among the various views and breakdown complex decisions into smaller more comprehensible steps.

Acknowledgements

This paper is the result of two years of discussion and brainstorming with many people including the participants of the workshop on multiple user interfaces we organized. We are grateful to Fabio Paterno, Gerome Canals, Paul Ashutosh, Franck Tarpin, Jean Vanderdonckt, François Oger, Valsilos Zarikas, Antonio da Silva Filho and Homa Javahery.

References

Foundations of Cognitive Support: 
Toward Abstract Patterns of Usefulness

Andrew Walenstein
Department of Computer Science, University of Victoria, Victoria, B.C., Canada
walenstein@csr.uvic.ca

Abstract. Computer tools for cognitively challenging activities are considered useful, to a great extent, because of the support that they provide for human thinking and problem solving. To analyze, specify, and design cognitive support, a suitable analytic framework is required. Theories of "distributed cognition" have been offered as potentially suitable frameworks, but they have generally failed to plainly articulate comprehensive theories of cognitive support. This paper seeks to clarify the intellectual foundations for studying and designing cognitive support, and aims to put them in a form suitable for design. A framework called RODS is described as a type of minimal, lightweight intellectual toolkit. Its main aim is to allow analysts to think in high-level cognition-support terms rather than be overwhelmed by task- and technology-specific implementation details. Framing usefulness in terms of cognitive support makes it possible to define abstract patterns of what makes tools "good". Implications are drawn for how the framework may be used for the design of tools in cognitively challenging work domains.

1 Introduction

A universal and critical design goal for tools is to make them useful. In the realm of physical labour, critical aspects of usefulness are understood in terms of mechanical advantage. For example a lever is understood to be useful primarily due to its ability to reduce the force needed to raise a mass. For cognitive work domains such as software debugging, financial forecasting, and writing composition, a key aspect of a tool's usefulness relates to how it improves cognition. For instance, we should expect that useful software visualization systems will make program comprehension easier, faster, or better in some way. Thus in highly cognitive work domains, usefulness is closely associated with the provision of cognitive support—i.e., with the ways in which cognition is helped, aided, or otherwise assisted by artifacts. Tools for cognitive work domains may fail to be usable in many ways (e.g., they may be hard to learn), but they cannot fail to provide cognitive support.

In order to systematically analyze, specify, and design cognitive support in tools, the fundamental principles of cognitive support must be known and effectively wielded. This is not easily done. Many common shortcomings of HCI
theories and guidelines are by now well known (see e.g., Carroll [6]). Four inade-
quacies are of particular interest in this paper. First, existing theories may be too
costly to learn and apply [4]. Second, they can be too task- or technology-specific.
This greatly hinders the specification of reusable and science-grounded patterns
of successful design [25]. Third, the theories can fail to adequately address use-
fulness in terms of cognitive support. Frequently the objective adopted for HCI
theories is to identify usability problems, instead of identifying the forms of cog-
nitive support which render a tool worthwhile using in the first place. Fourth,
the theories may be ineffective for generating design advice. The theories must
work in the absence of a prototype solution if they are to be especially useful in
design; they must identify steps one might take to add cognitive support.

These four shortcomings currently hamper the application of distributed cog-
nition (DC) theories. DC has been portrayed as a promising general theoretical
framework for analyzing cognitive work domains [10]. Several researchers have
recently adopted DC as an umbrella approach for HCI research and design (e.g.,
Wright et al. [30], Hollan et al. [10], Rogers et al. [20]). The key tenet in DC
is that cognition is not localized to a single individual, but is instead a process
distributed amongst various artifacts and humans. But knowing how any cogni-
tive system works is only a prelude to changing it by introducing new tools. It is
reasonable to expect that theories of cognition (even distributed cognition) will
primarily explain or predict how existing cognitive systems work. In contrast, a
theory of cognitive support is a theory of intervention: it tells one how cognitive
systems are improved via the introduction of artifacts. It is important to be able
to clearly articulate the principles underlying these interventions in a way that is
technology- and task-independent, and such that learning and application costs
are minimized.

This paper outlines an attempt to satisfy these goals by proposing a high-
level cognitive support theory. At its heart is a general cognitive support analysis
framework called RODS. RODS integrates and adapts existing theories from DC
and elsewhere. It is essentially a high-level and general theory of cognitive sup-
port together with an overarching framework for analysis. The central resource
within RODS is a list of four support principles identifying distinct classes of
cognitive support. These four classes of support give RODS its name.

RODS is proposed as a resource for designers to use throughout the design
and development process, but it is expected to be especially helpful during design
envisionment. Our long range goal is to facilitate various phases of design, but
our current focus is on relatively informal design reasoning such as might occur
during design brainstorming sessions. Our aims are thus similar to those of the
cognitive dimensions (CDs) work which “raise the level of discourse” [9, pg. 132]
of designers. In particular, we wish to specify theory-derived patterns of cognitive
support which are task- and technology-independent. These would allow design-
ers to avoid “death by detail” [9, pg. 131] and yet still proceed towards essential
design insights. If we are successful, we expect such “broad-brush” theories of
cognitive support to have broad implications for HCI development.
The rest of the paper is structured as follows. First, requirements for generating a suitable theoretical framework for cognitive support are outlined in Sec. 2. Six desirable traits are extracted from this analysis. Section 3 describes RODS and shows how it embodies these six desirable traits. Section 4 briefly overviews how RODS has been used to view various tool implementations as instances of patterns of cognitive support. Related work is overviewed in Sec. 5, and implications and conclusions are drawn in Sec. 6.

2 Leveraging Mechanical Support Theory

Identifying fundamental principles is a critical undertaking for any field of research. Fundamental principles frequently provide deep and general insights that are as important as the fine details. This fact was noted by Newell and Simon [16] in their 1975 ACM Turing Award lecture. They noted that the essential characteristics of a discipline can often be stated in short, general sentences. They highlighted, in particular, the importance of the cell doctrine in biology, the theory of plate tectonics in geology, and the germ theory of disease. These are all gross qualitative theories which are critical for understanding a domain. They tie together, relate, and organize multitudes of facts. The fact that they are not specific theories capable of generating precise predictions does not diminish their importance. High-level, generalized truths can be enormously valuable in understanding a broad range of phenomena at a high level.

The arguments raised by Newell and Simon likely apply to cognitive support theories. For designers in HCI, there is, in fact, a particular reason for needing a high-level, qualitative theory of cognitive support: cost of application. There exists a "cost gulf" [4] which can prevent otherwise helpful theories from being applied. High-level, qualitative theories are usually "lightweight" and widely applicable. Consequently, they can be expected to provide the analyst with the most benefit for the least investment in theory learning and application. Colloquially speaking, high-level, qualitative theories may yield the best "bang for the buck" for the analyst. Thus, articulating such an overarching qualitative theory for cognitive support should be given a high priority—ahead of, say, specific theories pertaining to one form or implementation of cognitive support.

Where will such a general, qualitative theory come from? Distributed cognition (DC) is one possible source. DC theory contains many insights into the ways in which tools affect and support thinking. It identifies key principles underlying DC such as the fact that artifacts are key resources used to represent and propagate knowledge. Our concern here is particularly on generalizable principles for how artifacts support cognition. Although DC is a prime candidate to be able to describe such principles, they have yet to be clearly and comprehensively elucidated. What are principles of cognitive support? How are these foundations best highlighted in a designer-oriented framework?

One way to approach these questions is to look to previous successful frameworks for inspiration. Here we shall turn to mechanical support theory for in-
sights. Compared to cognitive work, the realm of physical work is very well understood. We now understand a great deal about how to assist physical work.

Consider our understanding of levers. A lever is a simple machine—an “atomic” machine of sorts. It can be defined in structural terms as a movable pivot rotating about a fixed fulcrum. The total amount of work done is not reduced when using a lever. Instead, the lever is merely a force-amplification device, meaning that it reduces the force needed to move a load. However the assistance is not completely free: overheads are introduced by the fact that the lever must be depressed a greater distance than the mass needs to be raised. These overheads are perfectly acceptable when the leverage is needed.

There is much to admire about our relatively mature understanding of mechanical advantage. We know that the key principle for a lever’s usefulness is mechanical advantage. The concept of mechanical advantage can be stated simply as a way of trading off distance traveled for a reduction in required force. It is a general concept which is defined in abstract terms, and is thus removed from the particulars of its implementation (materials, sizes, etc.) or its use (lifting people, water, planets, etc.). Furthermore, the fundamental concept has a succinct and memorable name: “leverage”. Catalogues of various simple machines have been described (inclined plane, pulley, etc.). These, in combination, form a type of “language” for building more complicated physical labour saving devices [18]. With such a language we can decompose complex machines into their component machines, and compose new complicated machines out of simpler ones.

Our scientific understanding of the principles of mechanical support is now well-established. Designers of new machines can rely on these theories as justification for their design decisions without having to reestablish their veridicality. Even without complicated materials models and physics equations, the gross, qualitative theories are valuable. The basic concepts of simple machines are easy to learn and readily applied without deep analysis or scientific knowledge. It does not take a PhD in physics to apply the concepts during real design.

It would be highly desirable to be able to tell a similar tale for understanding cognitive support at the broadest levels. We can use our understanding of mechanical support as an intuition pump for deriving desiderata for understanding cognitive support. In particular, we can expect it to be helpful to have the following:

1. **Core theory**: this would explain the basic idea behind cognitive support. It would be the analogue to concept of mechanical support.
2. **Small vocabulary of advantages**: this would describe “atomic” principles of cognitive advantage. Ideally, this would identify orthogonal principles that can discussed independently. These principles, in turn, would generate equivalence classes identifying tools implementing a common type of cognitive support. These cognitive support classes would be analogues of the classes of simple machine (lever, pulley, etc.).
3. **Composition language**: this would explain how the various primitive types of support can be composed. It would be an analogue of mechanical composition (e.g., attaching a pulley to a lever).
4. **Mnemonic, evocative names:** the names of the cognitive support principles would ideally have helpful names that are easy to understand and remember. Ideally the names would index into deep expert knowledge concerning related issues.

5. **Abstract, generalizable description level:** cognitive support would be ideally defined at an abstract, functional level which is removed from particulars of the tool or its uses.

6. **Analysis framework:** this framework would serve as a foundation for applying the cognitive support principles during analysis. It would allow the analyst to decompose complicated tools into their component types of cognitive support. It would also allow the designer to do the reverse, i.e., to compose simple supports into more complicated ones.

In the future, as DC and HCI mature, we may expect to someday build detailed models of cognitive support and perhaps even be able to quantitatively predict cognitive benefits. But for now, it would be very helpful merely if the above six desiderata could be addressed in some adequate way such that the basic forms of cognitive support can be loosely analyzed. Ideally, such cognitive support analyses could be done without requiring an advanced degree in cognitive science. The following section describes RODS, an attempt to provide such a framework.

### 3 The RODS Analogue to Mechanical Support

RODS is a high-level, qualitative theory of cognitive support. The name “RODS” comes from the four main classes of cognitive support it identifies: task Reduction, algorithmic Optimization, Distribution, and Substitution. The framework is described in more detail below. It is introduced by arguing how it addresses the six desiderata from the previous section.

#### 3.1 Core Theory of Cognitive Advantage

The central tenet of DC theory is that cognition is not a process localized to an individual human mind, but one that is spread out amongst possibly many humans and artifacts [11]. Various artifacts, including computers, can therefore be viewed as parts of a single cognitive system. Critically, DC argues that a cognitive system will operate better or worse depending upon whether the appropriate external artifacts are available, and depending upon how they are designed. This insight is memorably noted by Norman who said “it is things that make us smart” [17].

Since DC generally explains cognition in terms of computation, the explanation of cognitive assistance must surely also be computational in nature. The essential argument is that a computation that utilizes a computational advantage may be substituted for another equivalent one which does not. By “computational advantage” we mean some way of improving the computation according to some measure (speed, memory use, etc.). Cognitive support can therefore be understood entirely in computational terms: support is the provision of computational
advantage. Newly introduced artifacts reengineer the overall computations involved in a DC system. Thus designing cognitive support means reengineering computational systems such that the system’s cognition is improved.

From these considerations, a general, qualitative theories of cognitive support can be stated as follows: The cognitive support provided an artifact is the computational advantage that it provides.

3.2 Small Vocabulary of Advantages

A staggering variety of cognitive artifacts are used by humans. Software developers, for instance, use a wide assortment of diagrams, compilers, analyzers, visualizers, editors, and so on. Yet it would be surprising if each variation in cognitive artifact would require a wholly different explanation. A more plausible situation is that is that some relatively small set of principles are in combination sufficient to account for the many varieties of cognitive support. Variations and combinations of these “atomic” principles might be seen to generate the enormous design space of cognitive artifacts.

Given that cognitive support is considered computational reengineering, the atomic support principles will be associated with distinct principles of computational advantage. The trick, of course, is settling on a suitable set of principles. Ideally, these would identify orthogonal computational advantages. We have found it useful to settle upon just four commonly understood computational principles. Each of these induce a class of support types; these support types are termed “task reduction”, “algorithm optimization”, “distribution” and “substitution”. These support classes are listed in Figure 1, and described below.

**Task Reduction.** It is sometimes possible to eliminate work that is unnecessary. For example a pathologically designed programmer’s editor might insist on having the developer re-read every line of code in a program before each and every edit she makes (e.g., by forcing a line-by-line scroll through the program). In most (all?) circumstances this would be a waste of time and effort. Computationally speaking, the problem is that there are unnecessary computations being performed; from an HCI point of view, the task can be reduced by eliminating unnecessary steps. Removing unproductive work will decrease the amount of cognitive work done and thus should influence performance.

This form of task and performance “enhancement” is quite obvious, but it is included in RODS because in some cases it is helpful to reduce real task demands. For instance, it may be possible to require only “good enough” answers from users. Moreover, it is critical to include task reduction in RODS so that one cannot confuse any of the other support types with a simple reduction in the work done. Thus the remaining principles will all insist on maintaining some form of equivalence in work.

**Algorithmic optimization.** Algorithmic optimization relies on the fact that differences in encoding or procedure in can create differences in performance
Task Reduction

- **Cmp Principle**: some functions are easier to compute
- **Substitution Type**: substitute simpler tasks for more complicated ones
  - Example (cmt): removing redundant or unused computations
  - Example (HCl): eliminating unnecessary steps
- **Design Principle**: remove unnecessary work; relax task demands

Algorithmic Optimization

- **Cmp Principle**: functionally identical algorithms differ in efficiency
- **Substitution Type**: substitute equivalent methods, ADTs, or encodings
  - Example (cmp): changing to doubly-linked list; switching sorting algorithm
  - Example (HCl): switching to Roman numerals
- **Design Principle**: optimize cognitive processes for task & infrastructure

Distribution

- **Cmp Principle**: distribution adds memory or computing resources
- **Substitution Type**: substitute external resources for internal ones
  - Example (cmt): caching memory to a hard drive; client-server architecture
  - Example (HCl): writing down a shopping list; automating constraint checking
- **Design Principle**: distribute (i.e., redistribute or offload) data or processing

Specialization

- **Cmp Principle**: specialized routines or processors can be more efficient
- **Substitution Type**: substitute specialized processors for more general ones
  - Example (cmt): use a FPU or accelerated graphics card
  - Example (HCl): enable visual search to substitute for "manual" search
- **Design Principle**: change representation to make use of specialized hardware

---

**Fig. 1.** Summary of RODS cognitive support classes.

without changing the essential outcome—i.e., without changing the “function” being computed [13]. For succinctness, we shall gloss over the many possible variations in terminology (e.g., “data structure” instead of “encoding”, “algorithm” instead of “procedure”, etc.). Normally, differences in encoding and procedure go hand-in-hand. For instance, for two different encodings of some data, different processes are normally needed to compute equivalent functions. Because of the intimate relationship between procedure and encoding (e.g., see Rumelhart *et al.* [21]) they are both considered to be variants of a single principle. The term “algorithmic” is adopted because it is a term used in analogous works in computing theory (e.g., see Aho *et al.* [1]). Where the meaning is clear, the term “algorithmic” will be dropped.

The principle identifies the class of cognitive support that works by *optimizing* the algorithm when given some particular task and computing infrastructure. This use is consonant with optimization in computing science, which normally
fixes the function being computed and the underlying computational infrastructure [2]. Performance can be optimized according to a variety of measures (e.g., speed, memory usage, etc.). In changing data encoding, the information content is presumed to be unchanged [13]. In computing science, a familiar example is the change from singly-linked lists to doubly-linked lists. Such a change can make various list operations simpler and quicker to perform.

In cognitive support terms, optimization normally means the introduction of artifact designs that reduce cognitive burdens on users (memory use, processing, etc.) by changing the encodings that are used. In other words, it frequently involves re-representation [31]. One instance of optimization is the difference made to arithmetic tasks when switching from Roman numerals to Arabic numerals or vice versa [17]. For example, Roman numbers are easier to add because the algorithm involved in adding Arabic numbers is complicated (involving symbol substitution).

**Distribution.** In the ordinary sense of the term, computational systems are called “distributed” when they have multiple, loosely-connected distinguishable computing resources. Distributing computations amongst multiple computing resources can have several types of performance advantages. Having multiple processing elements do the computation means that the work can be divided, thereby reducing the work done by each processor. This can also speed up the execution because frequently the work can be done in parallel. Distribution can also mean that computations with resource requirements exceeding the capabilities of one limited processor might still be performed if the excess load can be taken up by other elements. In computing science, a familiar example of distributed computing is a client-server architecture. One important reason for moving to client-server architectures is that the burdens on the client are greatly reduced.

In its application to HCI, distribution of computation can be interpreted in cognitivist terms [30]. Thus, instead of “data”, one can speak of “representations”, “mental state”, or “knowledge”. Instead of “processing”, one can speak of “reasoning”, “inferencing”, or “thinking”. For example, using an external memory [17, 23] distributes knowledge. Distributing processing analogously means either having the artifact perform or embody the processing, or having a user process symbols externally. An example of the former is a type-checking compiler. Type-checking compilers externalize the test of constraints on a program [5]. An example of the latter is the manipulation of a slide rule to compute a mathematical function [11].

**Substitution.** The principle underlying substitution is the fact that computing facilities can be adapted specially to restricted sets of tasks. The specialization means that they can be made more efficient. In computing terms, it means they compute fewer functions or operate over a restricted input domain. For instance it is common to have specialized processing hardware such as a floating-point unit (FPU), digital signal processor, vector processor, or graphics accelerator.
Unlike the CPU (which is a general, reprogrammable processor), such special computing hardware computes only a restricted set of functions and has limited programmability. The analogue in cognitive terms is the existence of efficient but specialized mental capabilities. These may either be “hardware” (built-in) or various forms of optimized “software” (learned or over-learned skills) [29]. A classic example is perceptual operators [7]. These are fast, effort-free, and execute at least partially in parallel [19]. They stand in contrast to deliberate reasoning which is slow, serial, and effortful.

The principle of specialization creates a category of cognitive support that operates by allowing more specialized processing to substitute for more general, deliberate reasoning. This substitution is a staple of the visualization literature. The essential quality of many accounts of visualization efficacy is that specialized perceptual mechanisms substitute for more complicated inferences (see Larkin et al. [13], Casner [7]). One standard example is the use of a line chart to enable visual search to substitute for deliberate search [7].

3.3 Compositional Language

A single cognitive artifact can be associated with multiple cognitive supports. For instance, when knowledge is distributed from the head and onto an artifact, perceptual substitutions might be enabled for operations over that knowledge. This implies that many of the varieties of more complicated types of support may be reducible to compositions of computational substitutions identified by RODS. Although this cannot be exhaustively demonstrated, we can illustrate the compositionality of RODS principles by deconstructing composite instances of support according to their constituent support types. Here we consider here Larkin’s analysis [12] of display-based problem solving (DBPS).

Larkin [12] invented the term “display-based problem solving” to describe a form of problem solving that makes extensive use of external displays. It is clear that to Larkin [12] there are three essential qualities of display-based problem solving: (1) that all or almost all of the relevant problem solving state can be read from the display, (2) that because of the nature of external displays, perceptual inferences can be used in places where otherwise more taxing logical inferences would need to be made, and (3) that little planning or deliberation is required for any of the steps—the solver employs very local control. Using this conception, DBPS involves a combination of distribution and substitution. The distribution involves distributing the current state of the problem. Substitution is involved because the solver is able to use perceptual skills to make inferences concerning problem implications, goal selection, or problem constraints. Substitution is also involved because learned rules are cued by rapid recognition of current system state, eliminating the need to deliberately reason about what operations to perform next. Once DBPS is deconstructed in this manner, a variety of different variations on DBPS can be entertained.
3.4 Mnemonic, Evocative Names

RODS is couched in terms familiar to most computer scientists. This does not guarantee that the terms will be more understandable, memorable, or more evocative of important implications. However the odds are that, unlike their cognitive science analogues, the computing terms will be at least meaningful to the average computer science student. Furthermore the terms may index into deep computer science knowledge of computing optimization. This may allow non-specialists to reason by analogy about various forms of cognitive support.

3.5 Abstract, Generalizable Description Level

RODS defines four categories of cognitive support and their underlying principles in cognitivist terms (knowledge, inferencing, etc.) and computational terms (memory, processing, etc.). Both of these are description levels are abstract and independent of their implementations. In particular, cognitive support categories are defined without reference to a task or even a particular cognitive model. For instance, distribution is defined without referring to the specifics about what are being redistributed. Plans, constraints, goals, and processing history might be distributed (e.g., see Wright et al. [30]). This means RODS can be used on whatever issues of cognition the analyst considers important (goals, social roles, etc.). In addition, the computational principles referenced are abstracted away from their implementation. For instance, any number of artifacts may play the role of an external memory (white boards, computers, even other people [23]). Thus RODS is independent of both task and technology: how they are used depend on the analyst’s application context.

3.6 Analysis Framework

The “output” of a cognitive support theory is an explanation of cognitive benefits provided by artifacts. The analysis of benefit is necessarily comparative: the benefits of an artifact may be understood only in comparison to what is implied by its absence, substitution, or modification. One way of thinking about this comparison is to suggest that there is a continuum of different levels of support that ranges from the completely unsupported (entirely mental) to the completely automated (no human thinking involved). In between are cases where cognition is spread between humans and artifacts. In practice, both extreme ends of the spectrum will be unattainable for interesting tasks. Still, it is instructive to imagine what, in principle, the unachievable extremes of the spectrum would entail. In particular, starting at the completely unsupported end of the spectrum will allow us to use RODS to define a structured space of cognitive support types.

In order for the entirely mental end of the continuum to hold up under close inspection, all of the user’s problem, evolving solution, and mental state information would need to be held internally; all of the processing of such information would also need to be done internally. Distribution begins to change that picture. As distribution of cognition is increased, the locus of cognition expands
away from the individual mind and is dispersed. As various parts of the computa-
tion are distributed, substitution and optimization changes can be added. That is, once data of any sort is distributed, processing can be distributed, substitutions can be enabled, and optimizations can be performed. For instance, Sharples noted that writing things down unlocks other types of support:

So long as ideas, plans, and drafts are locked inside a writer’s head, then modifying and developing them will overload the writer’s short-term memory. By putting them down on paper (or some other suitable medium) the writer is able to explore different ways of structuring the content and to apply systematic transformations... [24, p. 135]

The above observations suggest one conceivable method for generating the space of all possible cognitive supports for some task (conceptually, at least). The generation is effectively by recursive application of RODS transformations: start with the set of all possible distributions of data and, one-by-one, apply each type of cognitive support to generate new compositions. For instance in Sharples’ writing example, one could begin by enumerating the possibilities of distributing ideas, plans, drafts, goals, and so on. Then one might consider various ways of processing these externally, substituting efficient perceptual operators for more complicated ones, and of making changes to encodings and methods.

The procedure described above also suggests a way of improving design methods. Although an exhaustive generation of the design space would be impossible for real tasks, a restricted analysis might be both feasible and helpful. One would need: (1) a particular cognitive task analysis for how an unaided mind would perform a task (i.e., an analysis of the of data and processing required), (2) a list of known perceptual operators that might substitute for deliberate reasoning, and (3) a catalogue of common principles for re-representing data to optimize cognitive processing. Then #1 could be stepped through for ways of distributing data and processing, and then ways of applying #2 and #3 could be envisioned. This sort of analysis has proven feasible even for non-trivial tools such as software engineering tools [28].

In the case of the above sort of analysis, the process described generates a restricted space of design options. This might be used to make design envisionment more systematic. Alternatively, design options might conceivably be analyzed automatically (much as, for instance, how Casner automated perceptual substitution design [7]). The design space analysis might also eventually lead to improved designer resources. For instance, design heuristics and cookbooks might be produced which codify successful implementations of each known type of cognitive support.

4 Toward Mining Patterns and Theory

A philosophy guiding our development of RODS was that it will be fruitful to crystallize design knowledge by simultaneously considering our rich wealth of successful design within the context of our existing theories of cognitive support.
Our belief is that a good place to begin looking for patterns of good design is in the intersection between craft wisdom and scientific knowledge.

Towards this goal we have been applying RODS to extracting design insights from the world of software engineering tools. While doing this we have been building and refining RODS itself. Essentially we have tried to examine successful tools from the field while looking towards DC theory to justify the patterns we have found [27]. Our results are very preliminary, but we have found that RODS is helpful for giving the “gist” of an enormous variety of different types of support. At the same time, we have found that classifying a tool idea as an instance of cognitive support tends to pull in related literature that may be used to justify or refine the idea.

An excerpt of the results so far hints at the type and breadth of analysis. Table 1 lists several exemplars of tool features, a categorization of these as support implementations, and a list of literature that might be linked to the idea based on its association with the support type. Work has been started on using cognitive models to generate a finer classification system for cognitive support types [27].

5 Related Work

This work builds upon and synthesizes select aspects of many prior works from HCI, DC and cognitive science. These include works by Hutchins [11], Zhang and Norman [31,17], Wright et al. [30], and Rasmussen [19]. The main point of departure from prior works is the emphasis on clearly articulating a small, comprehensive, and orthogonal set of computational principles underlying the different support classes.

There exist several attempts at synthesizing a generalized, abstract understanding of cognitive support and related cognitive design issues. Neuwirth et al. [15] produced an analysis of how artifacts can assist in several cognitive tasks in hypertext authoring. Scaife et al. [22] try also to provide a synthetic account of the cognitive benefits of graphical representations. Both of the above analyses consider several instances of the cognitive support types identified by

<table>
<thead>
<tr>
<th>SUPPORT (data)</th>
<th>SAMPLE LITERATURE</th>
<th>EXEMPLARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang &amp; Norman [31]</td>
<td>Wright et al. [30]</td>
<td>code inspection checklists, diagramming constraints, undo history, link visitation history</td>
</tr>
<tr>
<td></td>
<td></td>
<td>type checking, compiler error list generation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPORT (processing)</th>
<th>SAMPLE LITERATURE</th>
<th>EXEMPLARS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang &amp; Norman [31]</td>
<td>Scaife &amp; Rogers [22]</td>
<td>type checking, compiler error list generation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>browser link colouring (visual search), algorithm animations, call graph visualizations</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPORT (substitution)</th>
<th>SAMPLE LITERATURE</th>
<th>EXEMPLARS</th>
</tr>
</thead>
</table>

Table 1. Examples of the types of RODS analyses explored.
RODS. Many other works touch on a smaller or more restricted collection of cognitive support types. These include frameworks and summaries by Tweedie [26] and Narayanan et al. [14].

Another collection of related works consider generalizable, high-level cognition-related design issues. The most prominent of these is probably the “cognitive dimensions” framework [8, 3] (CDs). The CDs are intended to represent orthogonal dimensions of usability tradeoffs in notational systems [8]. The CDs framework itself seems essentially orthogonal to RODS. Each class of cognitive support identified by RODS generates design considerations for notational systems. Thus the CDs may be a very helpful resource for identifying design tradeoffs while reasoning about cognitive support.

A third stream of related work is the proposal by Sutcliffe et al. [25] for producing a reusable federation of psychological claims together with their theoretical justifications. The most obvious surface difference between their work and RODS is that their claims libraries are proposed to be built up through a taxonomy-driven claims abstraction and collection process, whereas RODS is borne directly from prior theories. Nevertheless their approach aligns well with RODS. Notably, it might be possible for each of the cognitive support classes identified by RODS to be considered as abstract, reusable claim in some future extension of the work of Sutcliffe et al.

6 Conclusions and Implications

DC is a theoretical framework which appears to have significant promise for application in HCI. One need which has to date been inadequately met is for a clear articulation of a general framework for analyzing cognitive support. RODS takes steps towards providing such a framework. The framework defines primitive support classes and outlines a general method for generating a restricted design space consisting of various compositions of cognitive support types (subject to some particular analysis of a cognitive task). This capability has far reaching implications for the specification, design, and evaluation of interactive systems.

One set of implications stem from the succinct but inclusive definition of cognitive support. With such a definition it is conceivable to try to specify at design time the cognitive support desired or required and then work systematically during implementation to satisfy these requirements. For instance, a usability engineer might specify that a certain amount of memory offloading must be provided. Tests might be run periodically to ensure that usability targets (i.e., “usefulness targets”) are being met. We have performed a preliminary exploration of this possibility in the context of a study of observing and measuring plan and planning distribution in commercial software development environments [27].

A second set of implications stem from the fact that cognitive support identifies ways of actually improving cognition. Thus “good” design moves are defined and refined (made concrete). Refining “good” design moves can be expected to be helpful [30]. It may help systematize design space exploration and thus help eliminate designer oversights, thereby reducing the dependence on early usabil-
ity testing. Our analysis of a reverse engineering tool [28] (using a variation of RODS) lends evidence to this supposition. Our analysis showed that applications of cognitive support theory (based on RODS) could anticipate design changes made by the tool’s designers only after receiving user feedback.

In conclusion, we must emphasize that the importance of RODS may be in terms of a high-level, qualitative theory for understanding an entire field. If the analogy to mechanical support is apt, RODS may make it possible for HCI designers to forgo talking about usability blunders at the implementation level, and begin to speak directly in terms of usefulness at the cognitive level.

References

User Interface Design Patterns for Interactive Modeling in Demography and Biostatistics

Sergiy Boyko\textsuperscript{1,2}, Peter Forbrig\textsuperscript{2} and Anatoli Yashin\textsuperscript{1}

\textsuperscript{1} Max Planck Institute for Demographic Research, Doberaner Str. 114, D-18057, Rostock, Germany
\{beiko, yashin\}@demogr.mpg.de
\textsuperscript{2} University of Rostock, Department of Computer Science, Albert-Einstein Str. 21, D-18051, Germany
\textsuperscript{2} peter.forbrig@informatik.uni-rostock.de

Abstract. The paper is focused on the interface design patterns for interactive modeling and it is an effort to systematically describe the usage of UID patterns in this area. Main objectives were to develop the UID patterns to increase the usability of the software for numerical computing and to make the process of numerical simulation highly interactive.

On the basis of these UID patterns authors developed reusable software components in several programming languages: Java, Python, C++ and Matlab. The patterns described in the paper were verified in the development of practical software tools for demographers and biostatisticians, but can be readily applicable to other domains of numerical computing.

1 Introduction

Computers were conceived to perform numerical calculations. More then 60 years have passed since the moment of invention of the first digital computer\textsuperscript{1}. Nowadays scientists working in the area of numerical computing use modern software and languages that gradually replace old-fashioned Fortran, error prone C, or hard to master C++.

But how much HCI has changed in that area during the last 10 years? Has numerical modeling experienced the revolutionary changes that took place in the design of GUI driven software during that period? The answer is twofold. On the one hand, it is positive due to a wide spread of integrated modeling environments like Matlab, Matematica, and Mathcad with rich and powerful libraries of numerical procedures and extensive visualization facilities.

On the other hand, Fortran, C and C++ still predominate in the numerical computing area, they have a rich set of useful generic-precision intrinsic functions. There is a vast body of existing code written in these programming languages.

\textsuperscript{1} In 1939, St"ubitz and S.B. Williams built the Complex Number Calculator, the world's first electrical digital computer. Its brain consisted of 450 telephone relays and 10 crossbar switches and it could find the quotient of two eight-place complex numbers in about 30 seconds.
languages. Unfortunately, even with the help of modern modeling environments scientific programmers mostly create code operating in the batch mode, neglecting the possibility to create GUI for their programs and leverage the power of interactive modeling.

Let’s introduce some terminology. We will refer to the *modeling* as to the act of the formal representing of a system, biological object, effect, etc. We will focus on the mathematical *models* which is a set of assertions about properties of the object.

We will refer to the *design* to represent the act of defining an object. And finally, we will refer to the *simulations* as to results of executing of the *models*, or put it another way, as to output the model produces, which reflects the modeled experimental data.

This paper describes user interface design (UID) patterns for interactive modeling, i.e. the patterns for the communication with models, rather than UID patterns used by the software for numerical simulations. Our objectives were to develop UID patterns to increase the usability of the software for numerical computing and to make the process of numerical simulation highly interactive.

These patterns were verified in the development of practical software tools for demographers and biostatisticians. However, all patterns described in the paper are readily applicable to other domains of numerical computing.

## 2 Two Levels of Interaction

There are two levels of user interactions in numerical computing; the first level is the interaction with the modeling software, and the second level, on which this paper is focused – is the interaction with models. Let’s consider the following example that clarifies this point. Let’s assume that the user (scientist) simulates the model, described by a set of ordinary differential equations. The typical scenario could be the following:

1. Open the Modeling Environment
2. Load the model, stored in a file
3. Select the differential equation solver
4. Set the solver parameters
5. *Set model parameters*
6. Save the model
7. Run the simulation
8. *See the results*
9. *Make modification of the model*
10. Repeat steps 2-10

Items in italic correspond to the second level of user interaction. On steps 1-3 the user interacts with the modeling environment, he uses menus and dialogs to

---

1. In the simplest case it is a source code editor
navigate the directory tree and to open a file in which the model is stored. Then he selects a solver suitable for the given model, and sets its parameters.

When the user changes model parameters he is starting to interact directly with the model (step 5).

Then again, the user interacts with the software saving the model and running the simulation (steps 6, 7). On step 8, the user interacts with the model observing its outputs and if necessary makes additional modifications of the model, for instance, changing a relation between model inputs (step 9).

However, such an interaction with the model is not highly interactive. Modifications of the model are frequently performed simply by editing source files. Changes of model inputs may be performed only between the simulations [6].

To understand user’s goals and expectations about functionality of interactive modeling tools we need to develop a more comprehensive specification, based on task-oriented analyses of user interaction with the system [3, 4, 7].

3 Data-Centric Modeling

Because of the central role of experimental data for the mathematical modeling in demography and biostatistics, the following HCI pattern may be used as a generalization. We refer to this pattern as the “Data-centric Modeling Circle” to stress the importance of the experimental data.

The circle may be formally divided into several phases, each of which requires specific activities of the user.

Preliminary Analysis This phase usually includes extensive work with the raw data stored in a database. During this phase the user performs more or less standard statistical procedures. He transforms the data to a form that emphasizes those effects, which are of most interest. On this stage the data are frequently represented visually as charts, diagrams, etc.

Mathematical formalization of the model This is the most difficult part of the process. Almost everything happens in the investigator’s head. The scientists combines his knowledge about phenomena. He formulates a theory consistent with the new data and tests new hypothesis about the mechanisms of observed phenomena.

Simulation and testing As soon as the model has been designed, time to learn the solutions is coming. On this phase the model should be transformed to a form suitable for computer representation using the modeling software of choice.

On this phase the user interacts with the model, studies its flexibility and suitability for the description of the experimental data. The user, most probably, makes numerous changes in the model structure. Thus, one of the very important criteria is how fast these changes may be incorporated into the model, which in turn depends on the tools used for modeling.
**Evaluation** On this phase the user performs the evaluation of the model and compares simulation results with the experimental data described by the model.

4 UID Patterns for Analysis, Visualization and Simulation

Thus, user interaction with the model and data may be separated into three categories: analysis, visualization, and simulation. We generalized user interaction scenarios [7] based on a real-life episodes and developed the UID patterns for interactive modeling and reusable software components [6, 2] that were verified in several interactive modeling tools used to solve real-life problems in the fields of demography and biostatistics.

The main objectives of this work were:

1. Increase the usability of modeling tools
2. Prevent users from possible errors
3. Let the user make use of his expertise and creativity without having to spend time on the programming
4. Decrease time of the data-centric modeling circle
5. Develop reusable software components for interactive analysis of data, interactive visualization of data and interactive simulation of data.

The following sections provide more detailed specifications of user tasks and the UID patterns that model them. Each section consists of several subsections. Objective subsection provides a short description of the problem. Example subsection illustrates the problem using real-world problem, usually from demographic domain. Solution subsection explains the UID pattern that is used to solve the problem. Discussion subsection provides the explanation of usability principles on which the given UID pattern is based. And finally, Example of usage demonstrates the application of developed UID patterns to solve real-world problems.

5 Two-panel selector

5.1 Objectives

The user needs to select a subset of data. This selection is made on the basis of user expertise. Formalization of the selection procedure is complex, or impossible. User wants to work with different subsets, changes should be easily accomplished.

5.2 Example

After the Chernobyl accident numerous measurements of thyroid activities have been performed [1, 5]. Some of these measurements are of very low quality. The user needs to select trajectories suitable for evaluation of average thyroid doses based on these measurements.
5.3 Solution
To solve this problem, the Two-Panel selector has been developed. The Two-Panel selector (Fig.1(a), 1(b)) is a generalization of the pattern widely used in user interface design. Perhaps, the most well-known example is Norton Commander utility. The user has two panels and may select files or directories on one panel and copy or move them to another.

(a) Normal mode

(b) Left panel is zoomed, grid and information tags are shown

Fig. 1. Graphical Two-Panel Selector Pattern

The Two-Panel selector was mainly inspired by this example. We used two graphical panels: one panel to show the whole data set and another panel to show selected data. The left panel is used to show all data. Selections are activated by the mouse click. Selected trajectories immediately appear on the right panel. To make selected trajectories visually distinctive, they are displayed using lines of double width, and their nodes are highlighted. If the user made a mistake, it is possible to reselect a trajectory clicking on it in the right panel. To make
the selection of data more convenient, the user may zoom a rectangular area on the left panel. The right panel remains unzoomed to simplify a navigation and selection since in this way a user has a close-up and a general view of the data. Additionally, each trajectory has an associated information tag that may contain arbitrary information to specify the selected trajectory. The bottom panel on Fig.1(b) demonstrates a zoomed panel with information tags turned on.

5.4 Discussion

The Two-Panel selector greatly improves the usability and may be applicable to the design of systems for interactive analysis of data, when the user has to select subsets of data. Sometimes this selection is made on the basis of user knowledge and the development of selection algorithms may be an extremely complex task, because rules on which the formalization could be made are hard to articulate. Usage of this pattern significantly increases the interactivity, since the user can easily manipulate the data. Thus, time needed for data analysis may be substantially minimized. The two-panel selector also greatly increases the system feedback.

5.5 Example of Usage

The Two-Panel selector was firstly used in the software for the evaluation of thyroid activity for the population of Ukraine irradiated to the aftermath of Chernobyl accident [1, 5].

6 Interactive 2D User Input

6.1 Objectives

On the stage of verification the user wants to test how the model behaves when input parameters have specific shapes. The user wants to make these changes with the mouse, and to observe these changes immediately. The system must restrict the user behavior to prevent him from possible errors.

6.2 Examples

Simulation of the stress in the human and experimental animal populations. This pattern also finds its usage for interactive determination of initial guesses for non-parametric and semi-parametric optimization problems [10].
6.3 Solution

To solve this problem we developed the 2D Interactive Input Pattern. According to the objectives this pattern may be either restrictive or non-restrictive. The restrictions may have different forms and are based on mathematical requirements. The most interesting application of this pattern is the optimization of non-parametric or semi-parametric models. One of the most complex tasks is determination of the initial guess. Instead of guessing and trying different inputs coded in functional or tabular forms, the user may simply draw an initial guess by the mouse so that it will be close to the values of the empirical data.

![Image of 2D Interactive Input Pattern]

**Fig. 2.** Restricted and Non-Restricted 2D Input Patterns

Fig. 2 demonstrates a dialog of the software for numerical optimization of non-parametric discrete heterogeneity model used in the analysis of stress experimental data for nematode worms [9]. The graphical panels are different forms of the restricted 2D Input Pattern.

The left panel allows the user to change values of two parameters, restriction is imposed in the form $p_1 + p_2 \leq 1$, $p_1 \leq 1$, $p_2 \leq 1$. Graphically it means that parameters may have only those values that are below the diagonal of the square (shown on the panel). Current values of parameters are shown as a cross on the left panel and their exact values are displayed in the text fields.

The right panel demonstrates another form of the restricted user input. Thin lines on the right panel are used to give initial guesses for the optimization procedure. The thick line is the resulting modeled curve approximating the empirical data plotted as diamonds. The user may arbitrarily change the shape of thin lines.
with the mouse. The only restriction is that these curves must monotonically decrease. Obviously, this interactive procedure greatly simplifies the process of the initial guess finding. Being non-interactive, the initial guess determination would take a reasonable amount of time and could be a problem of its own.

Another interesting example of the restricted 2D Interactive Input is shown on Fig. 3. In the given case a restriction is the shape of the signal. The user may change signal shape only so that the signal is a combination of rectangular steps (meanders).

![Fig. 3. The Restricted 2D Input pattern](image)

The user may change the shape of the signal by dragging different segments of the curve by the mouse. Additionally, some frequently performed actions may be activated by double-clicks. For instance, a double-click on the left horizontal segment positions the right horizontal segment on the same level as the left one; a double-click on the central horizontal segment flattens the signal so that it becomes the straight line. Corresponding values may also be changed using edit text fields.

### 6.4 Discussion

Usage of the 2D Interactive Input highly increases the interactivity level of the simulation. The user changes model parameters with the mouse, and may provide complex shapes of input without coding. This significantly improves the flexibility. Additionally, the user immediately observes the input in a visual form that eliminates possible errors and increase the feedback of the system. The user is prevented from input errors when the restricted form of the Interactive Input Pattern is used.
Another interesting aspect of the 2D Interactive Input is simulations in real time. Using this pattern it is possible to change values of model parameters in the course of simulations [6].

6.5 Example of usage
The Interactive Input Patterns were used in the software for non-parametric estimates of discrete heterogeneous model [9].

7 Multiple Observer with States
7.1 Objective
The user wants to accumulate visual displays of heterogeneous data.

7.2 Example
The user works with the demographic data corresponding to different countries and different ages. He wants to have separate displays of data for each country grouped by genders.

7.3 Solution
To simplify the visualization of heterogeneous information the UID pattern that we called Multiple Observer with States can be used. The Multiple Observer consists of several displays, each of which can be in several states: active state, when it is ready to visualize the next set of data, and inactive state. Besides, the display may react differently on user actions: displayed data may be accumulated or each next data set may replace the previous one.

States may be changed by the user. For instance, the user may click on the display to make it active, or the display may be data-aware, when it knows the data it should visualize.

7.4 Discussion
Usage of Multiple Observer with States allows the user to easily accumulate displays of heterogeneous data, this pattern improves the performance time and user's satisfaction. The main usability principle on which this pattern is based is immediate feedback. It also minimizes user actions. Usage of the data-aware Multiple Observer prevents the user from possible errors and increases safety.

7.5 Example of Usage
The Multiple Observer with States is used in the interactive modeling tool for the simulation of age-dependent dynamics of longevity in the human population [10].
8 Summary

This paper is an effort to systematically describe the usage of GUI for interactive numerical computing in terms of user interface design patterns.

One of the most important objectives that we pursued was to develop UID patterns to increase the usability of the software and make numerical modeling interactive. Usability of the software can be measured by usage indicators: learnability, memorability, speed of performance, error rate, satisfaction and task completion [8]. To become a UID pattern the pattern should improve all user indicators. All presented UID patterns seems to satisfy these criteria.

Another aspect of this work was to demonstrate that GUI may significantly simplify the process of numerical modeling. The efforts spent on the development of user interface are often rewarding, because they decrease the time of Data-Centric Modeling circle.

At the same time the development of graphical user interfaces requires additional skills, knowledge and time. Every so often the developer and the user of the modeling software is the same person. That is why the development of user interface should be based on high-level software components [6, 2] that do not require substantial efforts on learning and usage.

With the software components, the developer can encapsulate pieces of models, and in order to use these components the developer does not have to know their internal details.

Frequently UID patterns may be mapped to the reusable software components or a combination of them. All described patterns were implemented as reusable software modules using several programming languages, namely C++, Java, Python and Matlab. We used well proved design patterns, such as MVC, State, Chain of Responsibility, etc [4].

Developed components provide clean and straightforward external interfaces for the developer and can be easily combined together with standard widget components to create complete interactive modeling tools. This has been demonstrated on numerous examples of real-world applications from the domain of demography and biostatistics [1, 5, 9, 10].

References

for the population of Ukraine, exposed in the result of Chernobyl accident. Kiev, publishing of Ukrainian Ministry of Emergency Situations and Chernobyl, 1997
User Interface Conceptual Patterns

Pedro J. Molina$^1$, Santiago Meliá$^1$, and Oscar Pastor$^2$

$^1$ CARE Technologies S.A.
Pda. Madrigueres, 44.
03700 Denia, Alicante, Spain
{pjmolina|smelia}@care-t.com

$^2$ Information System and Computation Dept.
Technical University of Valencia
Valencia, Spain
opastor@dsic.upv.es

Abstract. User Interface Patterns are not sufficiently explored at the Conceptual phase. Work in area of User Interface patterns is predominantly done at design phase but not enough work is dedicated to analysis patterns. This paper shows different examples of abstract user interface patterns and explores the impact of such patterns in the software life-cycle. Conceptual User Interface Patterns can be used for direct specification of device independent interfaces that can be refined using UI design patterns, or moreover, used to automatically obtain prototypes of the user interface specified in several devices.

1 Introduction

Patterns have proven their utility in different fields of appliance. Design patterns [6] or architectural patterns [2] are well known uses of successful patterns in computing. Patterns provide, in a given field, a common interlingua (or lingua franca [4]). At the same time, patterns document problems and its correspondent best solutions. Pattern languages are excellent tools for expressing the concepts involved in a given domain. Therefore, they constitute a valuable way to express distilled experience from real life.

In the User Interface field, most works deal with design patterns [12,14]. However, in this paper we advocate for starting employing patterns in early phases (requirements and analysis) and propagating them in later phases (design and implementation). Covering in this way, a pattern oriented development over the whole life-cycle following a similar approach as proposed in PSA[7].

In the last four years, we have been exploring first at the Technical University of Valencia and later at CARE Technologies the field of User Interface Specification based on patterns. As a result of such research, we have discovered a collection of patterns useful to deal with the specification of UI of information systems. The discovered patterns are useful to describe abstract user interfaces, guiding the design phase and providing automatic code generation strategies for fast prototyping. Such patterns are the elemental concepts described in Just/UI [9] (a model for abstract UI specification). A CASE tool implements the
model assistance in the specifications and we have also developed translators to automatically transform abstract specifications to implementations.

This work presents a way for applying patterns in the Conceptual phase in the area of User Interfaces. Section 2 introduces UI Conceptual Patterns and provides examples of them. Section 3 describes the role of the patterns in the specification process. Section 4 introduces a refining approach. Sections 5 and 6 cover design and implementation phases, respectively. Section 7 provides examples of code generation from conceptual patterns. Then, related work is presented. Eventually, conclusions and references are given.

2 UI Conceptual Patterns

Patterns are widely used in the User Interfaces area. Some works like Trætteberg [12], and van Welie [14] provide collections of patterns to deal with problems in the design and implementation of user interfaces. However, we are going to focus on the conceptual phase.

Following the approach of Fowler [5], we propose to apply the concept of pattern at the conceptual phase. Moreover, we follow a pattern-oriented approach in each phase of the software development as it is used in PSA[7].

At the analysis phase, while developing the Conceptual Modelling, we can detect and document valuable patterns. Such patterns appear in the domain where users and analysts talk about the requirements of the system to be built. Particularly, applied to the User Interface context, we can typify the User Interface Conceptual Patterns such as those patterns related to user interface topics that appear in the requirements gathering at the conceptual phase.

Conceptual patterns are clearly focused on what (the problem space) and not focused in how (the solution space). Accordingly, they expressly discard design and implementation issues. The rationale is to prevent taking any design or implementation decision at the analysis phase. Such topics will be covered after at the correspondent design and implementation phases.

The pattern language developed (Fig. 1) is oriented to detect common necessities or requirements from users. The patterns help analysts indicating what information must be gathered and how it must be organized. In this sense, they provide a common language between users and analysts to describe the requirements in terms of patterns. At this phase, each pattern supplies the accumulated experience of previous UI analysis. Furthermore, such patterns imprint structural, static and behavioural restrictions that must be taken into account in the following phases (design, implementation, and maintenance).

In the classical software life-cycle, it is better to detect errors in early phases than in the later ones. Patterns, like errors, provide more advantages if they are detected and applied in early phases.

Finally, the use of User Interface Conceptual Patterns provide the same advantages as patterns, but applied at the Conceptual phase: documented analysis concepts, provide a common language for analysis, and easy reuse of analysis concepts.
2.1 Examples of UI Conceptual Patterns

To illustrate the idea of UI Conceptual Patterns we are going to describe three examples using a classical tabular form. For each pattern we will provide a name, an also known as (other names), the problem to be resolved, the context (express the conditions needed for pattern application), the forces (different opposing requirements), the solution (a core solution, not details), the restrictions, an example, the rationale and other related patterns.
**Filter Pattern** A filter is a condition for searching objects. In information systems, users need very frequently specific searching tools. In the conceptual phase, analysts must capture such requirement.

<table>
<thead>
<tr>
<th>Name</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Also known as</td>
<td>Query</td>
</tr>
<tr>
<td>Problem</td>
<td>The user needs to browse and search objects belonging to a large set.</td>
</tr>
<tr>
<td>Context</td>
<td>In information systems is a very frequent task to search for objects. Powerful search mechanisms are needed to help the user.</td>
</tr>
<tr>
<td>Forces</td>
<td>The number of objects in the set may hinder the searching process. A complex query interface can be hard to understand for not experienced users.</td>
</tr>
<tr>
<td>Solution</td>
<td>Provides a mechanism to query the objects satisfying certain conditions. The analyst can express it in a OQL-like syntax with variables, letting the user introduce data in such variables in run time.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Objects to be searched must be comparable (in other words, objects have a common type to be comparable).</td>
</tr>
<tr>
<td>Example</td>
<td>Web searching engines, library searching facilities, etc.</td>
</tr>
<tr>
<td>Rationale</td>
<td>Provides a mechanism to reduce the complexity. The user can incrementally narrow the searching scope.</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>Order Criterium, Display Set, Population observation.</td>
</tr>
</tbody>
</table>

**Master/Detail Pattern** This pattern is typical for business applications. Head information and depicted data in a composite presentation increases the usability with respect to a solution with two single presentations.

<table>
<thead>
<tr>
<th>Name</th>
<th>Master/Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Also known as</td>
<td>Master/Slave, Director/Details</td>
</tr>
<tr>
<td>Problem</td>
<td>The user needs to work with different sets of information units linked by a relationship. The main information unit (the master) determines the slave information units (details).</td>
</tr>
<tr>
<td>Context</td>
<td>Applications normally contain scenarios with several aggregated objects. The user needs to interact with all of them.</td>
</tr>
<tr>
<td>Forces</td>
<td>Layout may discourage a master/detail presentation. Less important information can be accessed throughout navigation. On the other hand, Master/Detail maintains the information synchronized and jointly.</td>
</tr>
<tr>
<td>Solution</td>
<td>Provides a unique composed scenario presenting master and detail data at the same time. Synchronizes data in the details when master data changes.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Details are synchronized with master information units.</td>
</tr>
<tr>
<td>Example</td>
<td>Invoice/Order is a typical case of master/detail.</td>
</tr>
<tr>
<td>Rationale</td>
<td>The joint presentation of two or more information units allows the user to work in a unique scenario capable of performing several tasks and, at the same time, the scenario maintains the details synchronized with respect to the master component.</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>Object observation, Population observation.</td>
</tr>
</tbody>
</table>
Defined Selection Pattern If the analyst can detect a closed set of values for a given range (domain), the use of this pattern allows to reduce user errors, avoiding the introduction of invalid values.

<table>
<thead>
<tr>
<th>Name</th>
<th>Defined Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Also known as</td>
<td>Closed selection</td>
</tr>
<tr>
<td>Problem</td>
<td>The user needs to select one or more items from a list of elements.</td>
</tr>
<tr>
<td>Context</td>
<td>The selection set is closed and known in advance.</td>
</tr>
<tr>
<td>Forces</td>
<td>The number of elements could be high. The selection key could be hidden to the user. Localized aliases should be shown instead of keys.</td>
</tr>
<tr>
<td>Solution</td>
<td>Provides a table to gather the codes and aliases for each item. Ask for the default values and the minimum and maximum number of items selectable.</td>
</tr>
<tr>
<td>Restrictions</td>
<td>Items must be known in advance.</td>
</tr>
<tr>
<td>Example</td>
<td>Marital status or sex are classical examples of defined selection in UIs. Defined selection is frequently implemented as combo-boxes or radio-buttons.</td>
</tr>
<tr>
<td>Rationale</td>
<td>The user requirements can be gathered using a simple tabular form with an item per row. The number of items may encourage different implementations to maintain the usability. This pattern can prevent user-errors: it only provides to the user the valid values.</td>
</tr>
</tbody>
</table>

| Related Patterns | Introduction.                                                                      |

Fig. 1 shows the user relationships among the patterns in the pattern language developed. The complete collection of patterns can be consulted in [8, 9].

3 Using patterns at the analysis phase

The conceptual pattern language can be a key concept in a model for abstract user interface specification. As described in [9] we have developed a Model-based User Interface Development Environment based in conceptual UI patterns. Patterns are employed as bricks and blocks (created, composed, and/or reused) to build the UI specification. Such specifications maintain the good properties of the conceptual patterns:

- Target platform independence: No design or implementation decisions have been taken in the specification.
- Describe the problem in terms of well-documented concepts.
- Reuse work form previous cases.

A CASE³ tool is implemented supporting the gathering of complete object-oriented conceptual specifications comprising: system logic and abstract user interfaces. Such tool is being used in industrial developments with success.

³ SOSY Modeler is the industrial CASE tool developed at CARE Technologies S.A.
The capture of each pattern in the modeler tool is driven by a form template. The analyst can fill the template with the appropriate values to instantiate the pattern. Fig. 2 shows an example of template for gathering the *filter pattern*. In Fig. 2, fields labeled as a-d: collect a name (a), an alias (b), a help message(c), and observations(d), respectively. The pattern is instantiated by filling its template. Once instantiated (created), it can be applied to a specific context, composed with others, reused, or destroyed. Finally, the composition of patterns following the structure showed in Fig. 1 constitutes the complete UI specification of a system.

The modeler tool supports the validation of the specification. Each pattern contains a template that must be filled. When a compulsory field is empty the validation process can detect such errors in order to obtain a complete specification. Furthermore, there are rules for correct composition of patterns and type checking⁴. The validation process also checks these rules, thus providing a verification of correctness. The validation process provides a list of errors and warnings. Once the specification is finished and produces zero errors in the validation process, the specification is ready for the next phase.

⁴ The validation process is out of the scope of this paper. The number of validation rules is very high, so it is not described here for brevity.
4 Refining the specification

Once the analysis specification is built and validated, the next phase is the design. At this point, designers must decide about configuration, distribution, and other topics. After that, programmers have to take additional decisions about implementation.

Following an approach similar to [7], it is a good idea to employ patterns in different phases. For example, design patterns [14,12] could be applied at design phase to maintain a pattern-oriented development.

Moreover, we propose to refine patterns in a series of successive steps from analysis to design and from design to implementation. Such refining allows adapting the solution to a specific implementation environment. The initial specification can be refined in these phases to add such decisions for a given design and then for a given implementation. Once again, this working method is an excellent way to maintain the system documented.

4.1 The $\Delta$ (Delta) Effect

Some questions arise due to the refinement-based approach: What implications imposes a decision taken at the conceptual phase? How do UI conceptual patterns applied at the analysis phase influence the subsequent phases?

As stated before, a pattern can supply semantic, behavioural and structural restrictions. Such restrictions have a conical scope or $\Delta$ (delta) effect (Fig. 3): the decision has implications from its definition to refined parts in later phases. These parts are constrained by the pattern application decision taken at upper phases. The question marks in Fig. 3 represent questions made in the Requirements phase. Such questions with its correspondent answer can instantiate conceptual
patterns in the analysis phase. Fig. 3 shows the scope of such decisions as Δ-form triangles where parts of the subsequent phases are constrained or guided by such decisions.

In this sense, decisions taken in upper phases (conceptual patterns selected at analysis phase) can guide, or help to resolve decisions at the design phase where design patterns must be selected. A wizard can help to select patterns in the design phase guided from the information captured in previous phases. For example, a wizard could suggest the reification of the Conceptual Pattern Master/Detail [9] at the design phase with the design pattern Container Navigation [12, 14] and Navigation Spaces [14].

5 Design Approaches

Decisions taken at the analysis phase have implications and constrain the possible alternatives in the following phases. Nevertheless, to perform the design phase and implementation, there are different approaches. This section describes how conceptual patterns are useful in order to make good design choices. The most important design approaches are: Manual Design, Semiautomatic Design, and Automatic Design.

5.1 Manual Design

At the moment, O O methodologies require additional tuning in the design phase. This refinement must be provided by a designer. Designers base their decisions in knowledge and acquired experience. In this case, conceptual patterns could help the designer to constrain the population of the possible design patterns to choose from. Therefore, conceptual patterns provide experience and simplify the selection of valid alternatives for design.

5.2 Semiautomatic Design

Design is not done manually in this case. On the other hand, it is assisted by design model-based tools or by means of wizards, which guide designers to obtain designs and implementations. The implementation could be obtained in an automatic way by means of code generation. Bell [1] defines a kind of generators named translative in which, analysis domain is accompanied with design templates. It allows to get a set of architectural and design patterns for a certain architecture, and it guides the designer to choose the most adapted templates for the system. The main advantage is that it allows the intersection between a set of M domain analyses with a set of A architectures, obtaining therefore the cartesian product $M \times A$ of different possible sets of implementations. In this approach, mapping decisions between the conceptual and design patterns are not made in an automatic manner. With respect to our approach, the designer could select the templates in the design phase which are compliant to the patterns specified in the analysis phase. In such a task, this phase can be guided by an assistant like in SEGUIA [13, 15].
5.3 Automatic Design

Nevertheless, we discuss in Just-UI [9] the mapping that converts from the conceptual patterns to design patterns using different transformation techniques. It normally takes a number of decisions based on design experience, speeding up and simplifying the design and implementation process. This approach is currently been developed at the CARE-T firm, using the OO-Method methodology [10] to automatically obtaining not only the UI but also system logic and persistence. The fundamental advantage is that it only requires collecting information in the requirements and analysis phases, thus obtaining in a very early phase tangible results using automatic code generation. It also eases an early development costs estimation since it allows Function Points counting [11] using these conceptual patterns. Therefore, the final productivity obtained with this approach is better with respect to the previous ones.

<table>
<thead>
<tr>
<th></th>
<th>Manual Design</th>
<th>Semiautomatic Design</th>
<th>Automatic Design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design Effort</strong></td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Design Choices</strong></td>
<td>Many</td>
<td>Many</td>
<td>Few</td>
</tr>
<tr>
<td><strong>Error prone</strong></td>
<td>Yes (high)</td>
<td>Yes (moderate)</td>
<td>No</td>
</tr>
<tr>
<td><strong>Mappings from</strong></td>
<td>It depends on</td>
<td>It depends on</td>
<td>Always</td>
</tr>
<tr>
<td><strong>Conceptual</strong></td>
<td>designer</td>
<td>designer,</td>
<td></td>
</tr>
<tr>
<td><strong>Patterns</strong></td>
<td></td>
<td>but assisted</td>
<td></td>
</tr>
<tr>
<td><strong>Prototyping speed</strong></td>
<td>Slow</td>
<td>Medium</td>
<td>Quick</td>
</tr>
<tr>
<td><strong>Customization</strong></td>
<td>High</td>
<td>Medium</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Flability</strong></td>
<td>It depends on</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>designer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Design Approaches.

Table 1 summarizes the main characteristics of the three approaches presented for design.

Automatic design is our preferred choice. However, pure automatic translation could not be applicable or not desirable for each system, especially, when the system is out of the scope of the translator (out of the context it is based in) or an optimization must be introduced. Each system could require additional tuning that may imply changes in the automatically generated artifact. In such cases, manual design will be necessarily complemented with one of the previous approaches.

6 Deriving implementations

Implementation phase can be also performed following different implementation approaches: Manual Programming, Code Reuse, and Code Generation. Usually, mixed approaches of these three ones are also employed.
6.1 Manual Programming

This is the most traditional but, at the same time, the most error prone and slowest approach. Here, programmers manually write part or all of the application code. A big problem appears when it is necessary to develop a system as quick as possible but the implementation is built from the scratch. Furthermore, manual programming introduces errors in this phase so it is important to complete it with exhaustive debugging and testing tasks.

6.2 Code Reuse

On the other hand, we can use software reuse for speeding up the development. The reused artifacts are obtained from previous developments and must have been highly tested. There are different code reuse artifacts like components, frameworks or idioms. In the user interface field, CIOs (Concrete Interface Objects [13]) or widgets, libraries, ActiveX and JavaBeans components are the natural artifacts to be reused. Nevertheless, for any software reuse technique to be effective, it must be easier to reuse the artifacts than developing the software from scratch. Moreover, in order to reuse an artifact, you must know what it does, where it is, and if it has been tested. Therefore, it is crucial the use of repositories to increase the reuse in development teams.

6.3 Code Generation

Translators and code generation techniques can be applied to transform analysis and design patterns to an executable program in a given platform, architecture and language. Of course, in order to derive the implementation, coding decisions must be taken automatically. Therefore, a code translator must be parameterized to support variations in the implementation produced in order to change decisions about configuration, components to employ or code topics.

In our approach we have adopted a mixed solution. On one hand, we have used code generation in domain dependent software. On the other hand, we have reused frameworks in domain independent software. We have built translators for the most used platforms at CARE Technologies. There is a desktop solution implemented in Visual Basic. The Web platform is covered with ColdFusion and JSP technology. And the Java platform is implemented using JFC/Swing.

7 Examples

We provide a couple of examples to show the differences and commonalities of two different implementations for the same system. The developed system manages a golf tournament. Both examples are completely generated and do not contain any manual changes.

Fig. 4 shows an example of implementation for the Windows platform produced automatically using a translator for the Visual Basic 6.0 language. In this
Fig. 4. Example of an implementation for Windows.

Fig. 5. Example of an implementation for Web.
figure, a *Master/Detail pattern* can be observed (Scorecards play the role of master and Result plays the role of detail). Note that in the master component, the scorecard number 120 is selected. The detail shows 18 results for such scorecard number 120. In the master component, a *filter* (contained in the tab labeled: ‘By Player’) is applied to constrain the population of scorecards. In the detail component there are two toolbars. The right vertical bar is used to offer *actions* while the bottom horizontal one is used for additional *navigation*.

On the other hand, Fig. 5 shows an implementation of the same scenario in a Web environment using JSP, HTML, and JavaScript technologies. The implementation is also obtained by means of automatic translation from the system specification. The *filter* provides the same functionality as in the previous counterpart. Here the scenario only shows the scorecards. A limitation in the target platform may discourage the *Master/Detail* presentation, providing a different but equivalent solution: users can *navigate* to Results using the link labeled ‘Result’ in the navigation bar. *Actions* over objects are accessible from the services bar.

8 Related Work

Martin Fowler [5] has explored the world of analysis patterns and provides lots of examples in his brilliant book. Fowler’s patterns are focused on conceptual modelling and the abstraction level is closer to the patterns here presented. However, Fowler does not cover User Interface topics in his book.

On the other hand, van Welle [14] and Trætteberg [12] have developed user interface design patterns collections also focusing in usability issues and Web design. Such collections contain experience accumulated from designers and usability experts. These knowledge sources provides valuable experience and should be employed in the design phase. Accordingly, design patterns collections and languages can also be used as possible reification of Conceptual User Interface Patterns.

Coldewey and Krüger [3] proposed a framework based on a pattern language for developing form-based UIs. They also provide two conceptual patterns: *General Action* and *Dialog Category*. Granlund and Lafrenière [7] have develop the Pattern Supported Approach to cover the User Interface Design Process. In their approach they advocate for using patterns at different phases as medium for collecting experience and document the system. We absolutely agree with them, but we also intent to use patterns for supporting code generation.

Vanderdonckt describes SEGUIA [13,15] as a tool developed for the TRIDENT environment that provides a semi-automatic generation assisted by the analyst. Implemented as an expert system, SEGUIA assists analysts to transform a TRIDENT specification to source code making questions and suggesting responses to the designer.

Finally, in an excellent article, Bell [1] reviews the Object-Oriented Model-Based Code Generation technologies. He reviews specification approaches, tools
support and generation strategies. Such approaches are heavily oriented to develop the system logic and not aboard the problem of User Interfaces.

9 Conclusions

We have presented the kind of patterns we have been using for user interface developing. Such patterns have proven to be useful:

1. At the analysis phase: Analysts can employ this kind of patterns as a tool for documenting the user interface requirements. A tool based on such patterns helped to build UI specifications for information systems. The patterns were implemented as constructor concepts in a conceptual modeler tool.

2. At the design phase: The specification based in such conceptual patterns can be refined applying design patterns as described in [14] and [12]. Moreover, a design tool can suggest a reduced choice of mappings from each conceptual pattern to design patterns.

3. At the implementation phase: The design specification can be automatically transformed to source code to build parts or the complete user interface.

4. During the entire life-cycle: The documented specification based in conceptual patterns can be used to exchange ideas or knowledge among the members of the development team. It constitutes a lingua franca as stated by Erickson [4]. Therefore, they increase the effectiveness of the communication among developers.

Furthermore, the refining of UI conceptual patterns for different platforms has allowed us to provide a systematic approach for obtaining automatic implementations directly from the analysis models. Thus, it constitutes an excellent method for rapid prototyping and for increasing the productivity of software development.

The approach described is appropriate for resolving the problems presented in ubiquitous computing in a natural way: the development team must invest its effort in the abstract specification, minimizing the manual refinements for a given platform and tuning translators for each desired platform or device. Note that such tuning is done only once (the first project for such a device), the next time, the tuning can be reused. The principle followed is:

"Build one specification, translate it to N implementations".

The work is not over, it is just starting. Patterns should be distilled, reviewed and updated. In the next months we expect to discover new patterns and continuously enrich the pattern language that we employ [8,9]. At the same time, we think that public review is the best method to obtain valuable feedback from the research community to improve the pattern languages.
References

Monitoring Human Faces from Multi-View Image Sequences

Nailja Luth
Fraunhofer Institute for Computer Graphics, Division Rostock
Joachim-Jungius-Str. 11
18059 Rostock
nluth@rostock.igd.fhg.de
http://www.rostock.igd.fhg.de/

Abstract. The paper proposes a vision-based monitoring of human faces based on the automatic pose-invariant face detection and generation of detailed parametric descriptions of dynamic changes of face mimics. The monitoring approach involves the extraction and the classification of face features. The approach is designed especially for the monitoring of faces which captured directly during the Human Interface Interaction. The use of multi-view images provides sufficient information for the automatic head pose estimation and face mimic changes. The results of the automatic facial image analysis is described by Face Mimic Graph. The Face Mimic Graph provides the quantitative and qualitative information of face changes in conjunction with the time axis.

1 Introduction

The monitoring of human face mimics could be very useful on different applications, e.g. teleconferencing, Human-Computer-Interfaces (HCI), virtual avatars. Especially in HCI, the results of the face expression recognition in direct conjunction with the working environment of the user can lead to a derivation of emotional states of a person. The ability to recognize and to understand facial expression automatically may facilitate communication. Computer Vision community is mainly interested in the tracking of human face and in the recognition of facial expressions. Most automatic expression analysis systems attempt to recognize a small set of prototypic expressions, such as happiness, anger, surprise and fear. In our daily real communication with other persons we seldom deal with isolated face expressions but more often our evaluation of emotional expression is based on the total observable behaviour in a given situation.
1.1 State of the Automatic Face Expression Recognition

The difficulty and complexity of the automatic face expression recognition are reasoned by a variety of facial expressions across human population and context-dependent variety even for the same person [1]. The automatic face expression recognition needs to solve following tasks: detection and location of faces in a scene, facial feature extraction (FFE) and face expression classification (FEC). In this paper we address the task of FFE and classification of face feature changes.

Compared to research works for face recognition, there is a relatively small amount of work on FFE and FEC. However in the last two years the number of publications is permanently growing. The proceedings [2] of a recent conference on automatic face and gesture recognition presents many papers on face detection. There are mainly two approaches: holistic template-matching systems and geometric feature-based systems. In holistic systems, a template can be a pixel image or a feature vector obtained after processing the face image as a hole. In the latter, principal component analysis and multiplayer Neural Networks are extensively used to obtain a low-dimensional representation. In geometric feature-based systems, major face components and/or feature points are detected in the images. Different methods were developed to automatically recognize facial expressions. In [3] a system uses optical flow within local face regions for recognition. These methods are relatively insensitive to subtle motion because information about small deviations is lost when their flow pattern is removed or thresholds are imposed. As a result, the recognition ability and accuracy of the systems may be reduced. A 3D-face model was used to fit it into an image. Initial adjustment between the 3D-Model and surface images was performed automatically using View-based and Modular Eigenspace methods.

The distances between feature points and the relative sizes of the major face components are computed to form a feature vector. The feature points can also form a geometric graph representation of the faces. Feature-based techniques are usually computationally more expensive than template-based techniques, but are more robust to variation in scale, size, head orientation and location of the face in an image. All works can be divided in two groups: work with static images and with image sequences. Most methods in the first group are based on principal component analysis [4, 5]. In [6] two types of facial features will be extracted: geometrical positions of fiducial points on a face and a set of multi-scale and multi-orientation Gabor wavelet coefficients. An architecture based on a two-layer perceptron for the face expression recognition was developed.

1.2 Facial Action Coding System

Over the past decade many research works on face expression recognition are based on the special coding system FACS: Facial Action Coding System, developed by P. Ekman & W. Friesen [7]. FACS has been used extensively also in facial animation to help animators interpret and construct realistic facial expressions. FACS
is an anatomically based coding system that enables discrimination between closely related expressions. FACS deals only with what is a clearly visible phenomena. FACS is concerned only about the description of facial motions.

In reality, humans are capable of producing thousands of expressions varying in complexity and meaning that are not fully captured with a limited number of expressions and emotion categories. Therefore the systems in [8, 9] are aimed to recognize both subtle feature motion and complex facial expressions. This approach allows to capture the full range of facial expressions. In [8] the approach recognizes upper face expressions in the forehead and brow regions. Initially the feature points will be marked manually. The further automatic tracking of their movements across an image sequence is performed by using special pyramidal optical flow algorithm. The recognition of upper face AUs was developed by applying of Hidden Markov Models (HMM). In [9] the approach was extended to lower face AUs. The further development of the system for automatic face analysis (AFA) [10] involves additionally the extraction of furrows.

2 Monitoring Approach Description

The key features of our approach are shown in Figure 1. It must operate in future under a variety of conditions, such a varying of illumination, backgrounds, it must be able to handle non-frontal facial images of both males and females of different ages and races. We designed the approach for tracking of facial mimic changes based on automatic analysis of image sequences. Every image frame from the sequence will be automatically normalised and decomposed into face feature. We used the definition of Action Units (AU) of Facial Action Coding System FACS [7] for encoding the movements of face features (FFs). To minimize the user assistance we developed special image analysis methods for automatic detection and extraction of FFs from an image. The classification of some FFs was performed using Hidden Markov Models. The results of the face decomposition will be transformed into a face mimic graph (FMG). The visualization of face mimic changes that are recorded in the Face Mimic Graph and evaluation of the FMG are represented in Chapter 6.

In the first version we integrated the dual uncalibrated camera system for capture of face expressions. The simultaneous frontal and profile images of a face mimic deliver a sufficient information concerning the movements on a face.

The functionalities in rough overview are the following:

- registration and estimation of significant 2D-changes in nearly multi-view images
- automatic face feature extraction and tracking
- creation of a dynamic parametric description of the face mimic graph
- visualisation and evaluation of the face mimics dynamics using the Face Mimic Graph
3 Facial Image Analysis

Facial Image Analysis involves both image processing and analysis tools. First, to detect faces in image sequences. Second, to track the detected faces over consecutive frames and select the frames that will be best for the classification. The final step is the facial mimic detection on the extracted faces.

3.1 Image Frame Normalization

The first component of the Facial Image Analysis is the image sequence capture. The facial expressions of some persons “on-demand” were recorded as training sequences. The persons sat in front of the first camera, in profile of the second camera. Each facial expression began with neutral face expression. The image sequence is recorded with 20 frames. In the first version every fifth frame is selected as a relevant intermediate one. The captured face feature moving was enough to track their displacement positions. Every intermediate frame represents both the face feature moving and the head moving, that is mostly typical for face expression of an individual. Exact estimation of the head movement from the captured image sequence will be very difficult. When out-of-plane rotation of the head is small, an affine transformation on images can align images so that face position, size and orientation are kept relatively constant across persons, and these factors do not interfere significantly with feature extraction. To estimate the parameters of the affine transformation we used the position of three stable facial points. In our experiments three points were sufficient in most cases. The estimated transformation matrix is used to normalise the image frame relatively to the previous frame and is involved in the face mimic graph. The meaning of the normalisation is illustrated in comparing of both subtraction results (Fig. 2). The lower right difference image in Fig. 2 shows only relevant displacements that are generated by face mimic changes. On the contrary, the upper right image illustrates difference image that includes face changes as well as
head moving. The estimation of real face feature moving can not be done exactly without the consideration of the head moving.

Fig. 2. Example of image normalisation. The upper two grey level images show original neutral and smile images without normalisation; the third upper image represents the result of subtraction. The lower right image, as a subtraction of normalised images, shows the real moving of face features from neutral to smile state

3.2 Face Feature Detection

Automatic detection of changes in a face mimic begins with the detection of the face position in an image and division of the face image into face features, like brows, eyes, lips, and furrows (Fig. 3). Because of face asymmetry we consider structural and expression’s asymmetries as important characteristics of faces.

Fig. 3. Some examples of face features

The detection and localisation of face features is based on the detection of changes. The main steps of the process are described below.
1. **Extraction of change regions.** The change regions are face feature changes that are appearing during the face mimic changes. The extraction method is based on the subtraction of two first images and then the segmentation and further filtering of the result difference image using closing operator (Fig. 4).

![Fig. 4. Detection of change regions. The right image represents the result of segmentation and closing of the difference image between two left successive images.](image)

2. **Finding of face features search regions.** Firstly, all extracted change regions will be transformed into one-pixel-regions, using the skeletonisation method. The generated skeleton lines (Fig. 5, left image) characterize the change regions more efficiently. According the general parametric face model (Fig. 5, right image) can be now all search regions for feature candidates defined and consequently processed.

![Fig. 5. Skeleton image and the general face feature model.](image)

3. **Analysis of every search region** with the aim to extract the chin form, left and right face contour and mouth lines (Fig. 6). Extracted skeleton lines are considered as candidates for face regions.
Fig. 6 Extracted skeleton lines as candidates for chin, eyes and mouth search regions.

4. **The geometry models** of face features will be generated under consideration of the hypothetic suppositions about their positions and the positions of extracted skeleton lines (Fig. 7).

![Diagram of face features](image)

**Fig. 7** Parametric geometry form of lips and eyes boundaries

The points $P_i$ represent an approximated polylines of lips and eyes countours. They are called feature points and their displacements are important for the classification of face feature changes.

5. **The algorithm for the search of feature points** will be illustrated by example for finding of feature point $P_1$ of the left lip corner. We adapted the main idea of the method in [11] to our application. A trapeze window with short basic line on the left side. A mean value of grey levels $\text{mean}_{\text{grey}}$ and minimal value $\text{min}_{\text{grey}}$ will be calculated for every column. In case of
the number of minimal column will be calculated. The left and right corners of lips and eyes are the darkest points in their environment. The value for threshold is smaller than 0.6. This estimation was done on the basis of calculated diagrams of many experiments (Fig. 8).

![Graph](image)

**Fig. 8** Statistical quotation between the grey level of the lip or eye corner and direct neighbourhood

The results of lip, eye and brow corner finding are represented in Fig. 9. The left and right corners of lips and eyes are the darkest points in their environment. The value for threshold is smaller than 0.6.

![Images](image)

**Fig. 9** Results of the detection of mouth and eye & brow corners

6. **The search of other feature points** P2, P3, P5, P6, P7 (Fig. 7) will be detected by using the edge detection. We applied the Deriche’s gradient operator modified by Lanser [12] (Fig. 10).
7. **The tracking of feature points** in next image will be done using correspondence analysis.

For instance, a detected chin, mouth, brow or eye contours build potential search regions. After the detection of these regions, a more refined model for the face is required in order to determine which of the detected regions correspond to valid faces. The use of the eye detection algorithm in conjunction with the head detection module improves the accuracy of the head model and discards regions corresponding to back views or other regions that do not correspond to a face. The results of the eye detection algorithm are used to estimate the face pose and to determine the image containing the most frontal pose among a sequence of images.

8. **The detection of furrows** is based on the gradient filters. The results of detected furrows are represented in Fig. 11. We consider furrows in forehead, eyes, between brows and cheek regions.

![Fig. 10 Example of the edge detection for lip contours](image)

![Fig. 11 Detected furrows](image)

### 4 Facial Action Units Classification

The changes of feature points for every face region could be well coded by FACS [7]. FACS breaks down the face into upper and lower face actions and further subdivides facial actions into small units called Action Units (AUs). Each AU
represents an individual muscle action, or an action of muscle group, into a single recognizable facial posture. In total they classify 66 AUs that in combination could generate defined and gradable facial expressions. Example AUs are the inner brow raiser, the outer brow raiser and the lid tightener. Each AU is a minimal action that cannot be divided into smaller actions.

The extracted feature positions will represented by feature vector sequence. The classification of feature vectors that we are proposing involves the use of Hidden Markov Model (HMM). The “states” of the HMMs include the movements of feature points of brows, eyes or lips. First the feature vectors will be transformed into a sequence of a finite set of symbols using the quantisation function. Our quantisation function is developed using the clustering analysis. As an example, we describe the sequences of feature vectors (FV) for brow regions. The FV with length 12 \( v=(v_i), i = 1,...,12 \) will be quantised into 16 clusters: \( 1,...,K, K = 16 \).

Every symbol sequence corresponds to an Action Unit, which is modeled by HMM. HMM describes the statistical behaviour of a process that generates time series having certain statistical characteristics.

A discrete HMM has \( N \) states \( \{s_1, s_2, ..., s_N\} \) and \( M \) observation symbols \( O_1, O_2, ..., O_M \).

We used a 4-state HMM with left-to-right-topology for the classification of brows motions, AU1, AU2, AU4 (Fig. 12).

**Fig. 12** HMM-toplogy for AU1, AU2, AU4

For every AUs of brow motions was a separate AU-HMM modeled.

The classification procedure is following: the observable chain of states of a face feature and will be classified using all AU-HMMs for this face feature. The maximum of the probability defines the corresponding AU (Fig. 13).

**Fig. 13** Classification procedure
5 Face Mimic Graph

Despite some restriction of FACS we decided to use it because of the general character of coding of face expressions. The essential restriction of FACS are following:

- Real facial motion is almost never completely localized. However Facial Action Units in FACS are purely local spatial patterns. Therefore the detecting of an unique set of action units for a specific expression is not guaranteed.
- The description of AU is qualitative. The subtle motions of eyes, lips or brows can not be described quantitative.

We designed a Face Mimic Graph - FMG, that allows to describe more flexible and dynamical the motions of face feature and to avoid some FACS limitations.

The FMG (Fig. 14) involves both permanent face feature - lips, eyes, brows, face contour - and transient feature – furrows.

![Fig. 14 Face Mimic Graph](image)

The geometric description of every face feature consists of the position of feature points and parameterised curves of feature contours. In the next table “Changes” all the displacements of feature points will be notated for every frame. In case of eyes, lips and brows the AU-classification result will be written. The combination of facial AU for every frame could be used in future for the classification of face mimic.

6 Monitoring of facial mimics

The results of the face changes monitoring is registered in the Face Mimic Graph. It involves all information that describes the face feature changes, head moving and
results of Action Unit classification. In the first version we developed only the classification of some face feature changes. These registrations of face feature changes are the first step towards the classification of the face mimics. The classification of main important face mimics are our next future tasks. Under main mimics we mean a-priori defined mimics states clusters that are relevant for specific applications, such as positive or negative emotional states. In the current state, the visualisation of the Face Mimic Graph is limited by a histogram of changes dynamic. In Fig. 15 is an example histogram on image sequence monitoring over a short time. The person changed his face mimics “on-demand”, the captured image sequences was automatically analysed.

**Fig. 15** Histogram of face change dynamic over 3,5 minutes

The corresponding images to a chosen time point can be visualised directly by setting of cursor, e.g. the cursor position is labelled by black color in figure 15. The corresponding images are shown on the right side.

7 Conclusions

In the first version main functionality of the automatic face monitoring is developed. The special methods of the detection of face features and the estimation of their changes are developed. The results of automatic tracking of feature changes and their classification are described in the Face Mimic Graph. Additionally, the 3D-modelling of a face provides the 3D-visualisation of the face and the estimation of 3D-changes of face features. There are some significant restrictions in the first version:

- only one face must be represented in a image
- stable background without quick movements behind the observed head
• manual interaction for definition of stable points and sometimes for editing of found coincident feature points
• using HMMs means the training for all Action Units. Alternative classification methods are required in future.

The important future extension of the approach lays in the classification of important face mimics, that describe positive expressions, like smile, as well as non-positive ones, like sadness or unhappiness. Additionally we plan to integrate more alternative methods especially for automatic analysis and classification.

References

8. Lien, J., Kanade T., Cohn, J., Li Ch-Ch: Automated Facial Expression Recognition Based on FACS Action Units. Proceedings of FG’98, April 14-16, 1998 Nara Japan
Improving Mouse Navigation -
A Walk Through the “Hilly Screen Landscape”

David Ahlström, Martin Hitz and Gerhard Leitner

University of Klagenfurt, Department of Informatics-Systems,
Universitätstraße 65-67, A-9020 Klagenfurt, Austria
{david.ahlstrom, martin.hitz, gerhard.leitner}@uni-klu.ac.at

Abstract. During computer interaction much time is spent navigating the graphical user interface to find and invoke functions through interface controls. If this navigation process could be optimised, users would spend less time searching for and navigating to interface controls. This paper presents a walk through an ongoing research project aimed at developing and assessing a navigation support module for mouse based interaction, which enhances standard screen pointer behaviour with position context sensitive functionality - creating a “hilly screen landscape”. The main hypothesis of this work is that a context sensitive screen pointer prevents navigation to and selection of erroneous and inappropriate interface controls, decreases pointing and selection times and contributes to increased overall usability of the application. A description of the navigation support module and hypothetical situations where such a module could prove to be useful are provided together with major implementation and evaluation issues of the project.

1 Introduction

The complexity of a software’s functionality is normally reflected in its GUI (Graphical User Interface). Small and simple software products often have simple and easy to understand GUIs. As the complexity of the functionality grows, the GUI becomes more and more complex and for the user harder to understand and to operate. In complex GUIs of modern software with several menus, buttons and other interaction controls, it is hard for the user to know where to navigate, how and when to invoke functions. Moreover, increasing use of mobile computers equipped with pointing devices with questionable ergonomic properties (such as track points, track balls and touch pads) in inappropriate working environments (like planes, trains, or cars) significantly complicates mouse based navigation for most users. Hence, there is an increasing need to explore new methods to support the user in the GUI based interaction dialogue.

Since much of the time spent during interaction is invested in the actual navigation process (visual search of the GUI to localise the appropriate interface control, followed by moving the screen pointer to the control in order to manipulate the control), this seems to be one promising point of attack to tackle the problem of complex GUIs in order to make them easier to operate.
In this paper we present an ongoing research project which aims to optimise mouse based interaction and navigation. We intend to implement and evaluate a navigation support module which overlays GUIs with simulated gravitation fields, making the screen pointer position context sensitive. The navigation support module defines peaks and valleys within the GUI and thus forms what we call a "hilly screen landscape". Contrary to ordinary GUIs (with totally flat screen landscapes), a GUI enhanced with such a navigation support module exhibits changes in effort required to navigate within certain areas of the screen. Mouse motions within a valley results in faster screen pointer motions, allowing easy and smooth navigation. Whereas moving the screen pointer alongside a valley or "downhill" is easy, moving the mouse "uphill" results in delayed reactions of the screen pointer, making it harder for the user to navigate in that direction.

We hypothesize that in applications enhanced with such a navigation support module and the resulting hilly screen landscape

- navigation will be easier and less error prone compared to applications with standard GUIs,
- cursor targeting and selection task times will be reduced, and
- that the module contributes to an increased overall usability of the application.

The following section points out relevant related work. In Section 3, we present the planned functionality of the navigation support module along with some concrete user situations where the module might be useful. Section 4 describes major implementation issues and Section 5 emphasizes on the planned assessment and evaluation process. The paper concludes by briefly describing the project outline, highlighting important project steps, and sketching the current status of the project.

2 Related Work

Accurate navigation and screen pointer positioning are particular problematic with isometric pointing devices [13] and other pointing devices with similar questionable ergonomic properties [1]. The main remedy approach for navigation problematic has been to introduce new hardware devices.

Previous research done to support the navigation process can roughly be divided in two categories, research and development of new haptic input devices which enhance interaction by tactile and force feedback, and research concentrated on navigation and object manipulation within virtual reality environments. In the first category we find work with the FEELit (Immersion Corporation [8]), a mouse physically attached to a limited workspace and equipped with motors to steer the hand of the user. The produced forces where found to improve steering and targeting tasks [6]. Similar results are reported in [2], where a multi-modal mouse with both force feedback (produced by creating a drag between an electromagnet in the mouse and the iron mouse pad) and tactile feedback (produced by a solenoid-driven pin stimulating the index finger) was
used. In [4], Campbell et al. describe a vibrating isometric joystick and stress the importance of synchronization between tactile and visual feedback.

The usage of haptic devices such as the PHANToM (SensAble Technologies Inc. [15]) for object manipulation and navigation in virtual reality environments has been evaluated [3]; research was done also into enhancing the traditional desktop system with the same device [12]. Oakley et al. [14] augment buttons and scrollbars with different haptic effects using the PHANToM. They show that mechanisms which automatically push the screen pointer towards the widget’s centre can be helpful in targeting tasks, thus supporting “clear” hits and prohibiting accidental slip offs. Considering size and complexity of the device, the usability in mobile and desktop computing is questionable.

Little work has been done to experiment with standard input devices such as the mouse and its derivatives for mobile computers, found in most modern computing environments.

One common drawback of the above mentioned projects is that they all are device centred. For the user to enjoy the extra functionality and the extra feedback, an often clumsy and expensive device has to be purchased and installed. A new way to operate the computer has to be learned and trained. Furthermore, most of the devices are only applicable and usable in the context of special software applications.

Thus, it seems promising to investigate the possibilities of enhancing GUI based navigation in the context of standard hardware equipment and everyday software applications.

3 Description of the Hilly Screen Landscape

We use the metaphor of a hilly landscape to facilitate the understanding of the navigation support module’s functionality. In nature, a landscape is a region of land with distinguishing characteristics and features. If one of its characteristics is that of differences in altitude, the landscape is said to be hilly or mountainous.

As a runner in an orienteering competition has to navigate from control station to control station in the landscape, the computer user has to navigate through a screen landscape from widget to widget in order to complete his tasks. Whereas the athlete might have to navigate throughout a hilly landscape, where the control stations are placed on the top of hills, on plateaus and in valleys, the computer user always navigates in a totally flat landscape. On his map, the athlete sees the changes in altitude and he can also notice the changes while running. The effort required to run uphill is greater than the effort required to cover a downhill distance. For the computer user navigating in the screen landscape, the effort required to cover a certain distance is equal anywhere in the landscape.

Our navigation support module transforms the flat screen landscape into a hilly screen landscape. As the orienteering competitor, the computer user can now notice changes in altitude during navigation in the screen landscape. A
graphical representation of the screen landscape indicates the locations of hills and valleys on the screen in order to provide further and visual support.

By careful and intelligent placement of peaks and valleys, the resulting hilly landscape can reflect the intended and most optimal navigation paths for the current screen. The screen pointer behaviour is made context sensitive, depending on screen position and widgets at the position. Less frequently used or even dangerous areas, such as delete buttons etc., are placed on top of hills, to keep them out of the natural navigation path. Many screen areas are of no or little interest, containing no interaction elements, and navigation should be focused to the relevant areas.

3.1 Concrete Usage Situations

Fig. 1 depicts a typical data entry form which contains a number of input fields and buttons, often found in e.g. Web applications. The curved line (starting at the top of the form) symbolises a characteristic navigation path through the form. The user navigates from field to field, finally arriving at the lower right hand corner at the submit button.

![Image of a data entry form](image)

**Fig. 1.** A typical data entry form with marked characteristic navigation path.
The interesting parts of the form are the areas that contain interaction elements. Empty parts or parts with only text are of low relevance for the navigation process. To support navigation to and within the interesting areas, we put them at low altitude in the hilly screen landscape, making it easy for the user to move the screen pointer to and within these areas. We want to discourage navigation in the direction of the bottom left hand side of the form where the “dangerous” buttons labelled “Exit” and “Delete” are located. In order to make it harder for the user to reach these buttons, we place them upon a hill. Fig. 2 depicts the hilly screen landscape for the data entry form in Fig. 1.

![Fig. 2. Graphical representation of the hilly screen landscape for data entry form in Figure 1.](image)

The screen shot in Fig. 3 shows a frequently occurring situation. During a slide show presentation, the main interaction task for the presenter is to step between the slides. The presenter might be using an optical mouse operated with a track ball (as often found in auditoriums) to navigate the GUI and control the computer. Having no stable support for the hand movements, the task of selecting and pressing the back and forward buttons in the bottom part of the screen will most certainly be a difficult task. Using the navigation support module, the bottom part of the screen could be placed in a valley, thus supporting the navigation towards this interaction relevant area. A user that is aware of the landscape, having a mental picture of it and its functionality, can deliberately take advantage of it. With minimal mouse movements he/she can start the motion towards a valley then letting the screen pointer slide down in the valley, towards the target. But a user does not have to be aware of the landscape to take advantage of its functionality, since the screen pointer automatically moves in the “right” direction (i.e., downwards).

In several applications, the user is confronted with very small control elements (e.g., buttons for scrolling up and down in text documents and handles for selection and manipulation of items in drawing application) which require precise and accurate screen pointer positioning. By placing small targets within dimples in the screen landscape the target areas are virtually made larger. As the screen pointer reaches the vicinity of the target, it slides towards the tar-
get center, ensuring a hit. The acquiring act, precisely positioning of the screen pointer, and then determine that a successful acquisition has taken place can put great burdens on low vision users [7] and motor disabled users [16], as well as on non-disabled users. "Gravity fields" around targets might reduce this burden.

Fig. 3. Example situation where the hilly screen landscape could be useful. By placing the back and forward buttons (bottom) in a valley, navigation is supported towards this relevant area.

3.2 Additional Functionality

Beside the main functionality of the navigation support module, i.e. creating a hilly screen landscape which provides support during navigation, some additional functionality seems to be appropriate and/or required in order to maximize the usefulness, such as

- a visual representation of the hilly screen landscape (such as a map with contour lines or a 3D graphic), which can be activated and deactivated by the user,
- activation and deactivation of the module itself and
- mechanisms that let the user modify and reshape the landscape according to desire and needs.
4 Implementation Issues

The functionality of the hilly screen landscape and the concrete user situations described in previous sections reveal some constraints and issues we have to take into account during the specification and implementation of the navigation support module (called “HSL” in what follows). The most prominent issues concern

- ways to manipulate the screen pointer behaviour (techniques to implement the “gravitation” fields which realise the slopes within the hilly screen landscape),
- the choice of implementation platform and language,
- the level of implementation (device driver level vs. application level),
- portability and operating system independence and
- the possible integration of a 3D environment for visual representation of the hilly screen landscape.

From a broad architectural perspective, the HSL will be injected between the GUI oriented parts of the operating system layer and the application layer, respectively (Fig. 4). The HSL control module can be conceived of as an extension of the respective operating system services, taking into account any screen landscape definitions provided and controlling screen pointer movement accordingly. It offers an API to the application layer, which in turn needs to be augmented by a HSL definition module. The main function provided by the API is the screen landscape definition: The HSL definition module informs the HSL control module about the topography of each affected window employed by the application. The HSL control module will then modify the screen pointer movement characteristics whenever the respective window gains the interaction focus. In addition, the API of the HSL control module encompasses auxiliary functions to control diverse HSL states like enabling/disabling gravity effects, displaying/hiding the landscape representation, etc.

![Conceptual view of a standard architecture (a) and conceptual view of a HSL enabled architecture (b)](image)

**Fig. 4.** Conceptual view of a standard architecture (a) and conceptual view of a HSL enabled architecture (b)
We have divided the actual implementation process of the navigation support module into roughly five phases, where each phase is followed by a usability evaluation phase.

1. **Motion manipulation** - The focus of phase one is set on the actual behaviour of the screen pointer. Experimental development is used to identify possible techniques for manipulating the screen cursor motion behaviour under different operating systems and programming languages. This experimental phase is followed by an evaluation process which has as its primary aim to identify the most suitable technique(s) for further implementations, as well as to collect basic and valuable user attitudes to the idea of a navigation support module.

2. **Characteristic application** - Based on the results of the first implementation and evaluation phase, the product of implementation phase two is a small application-like (e.g. a data entry form as such in figure 1) prototype where user navigation tasks can be examined within a realistic setting.

3. **Additional functionality** - The third implementation phase concentrates on additional features of the navigation support module, i.e. functions for runtime modification and visualization of the hilly screen landscape, to be integrated into the prototype from the previous phase.

4. **Example application** - Phase four of the implementation process results in a full scale test application, with all functionality integrated, which allows us to evaluate the implications of the navigation support module on all levels, ranging from execution of user sub-tasks to implications of overall application usability.

5. **Plug-and-play** - The last phase is devoted to implementation of a plug and play functionality, where we explore possible solutions to make the navigation support module executable with arbitrary applications.

For a brief report on the current state of development, we refer the reader to Section 6.

### 5 Usability Evaluation Issues

In order to verify or falsify our hypotheses put forward in Section 1 we plan to conduct a series of usability tests. At least one usability test will follow each implementation phase. In the first usability tests we will focus on the screen pointer behaviour in order to identify the most suitable technique(s) to manipulate the screen pointer behaviour. As we proceed through implementation phase two and three, the basic functionality is secured and we can evaluate the navigation support module in combination with a “real-life-like” application.

Throughout the whole assessment process, we will run the tests using several types of input devices (e.g. mouse, track point, track ball, touch pad) to find out what devices are most successfully combined with the navigation support module.

The following two subsections describe the usability tests in some more detail.
5.1 Early Usability Tests

Users will be confronted with the functionality of the navigation support module without performing a concrete task. They will be asked to navigate around as they like to for a few minutes. Parameters such as eye movements on the screen and facial expressions will be recorded. Additionally, the subjects will be asked to verbalize their thoughts about the “device” (i.e. the behaviour of the screen pointer), what they think is the difference to other “devices” they are familiar with and what their subjective impression about the “device” is. Different data will be collected in order to find out whether and how subjects are acting and reacting in a subjective as well as in an objectively measurable way on the feedback given from the different input devices and screen pointer manipulation techniques.

Next, the subjects will be asked to perform a small dummy task to collect also data of a real application task. The subjects should be able to complete the task due to the previous habituation phase. Objective measures such as task completion time, error rate, error correction possibilities can be recorded and will serve as an additional criterion for the usability of the different navigation techniques.

Results of the first step should be that we can decide what screen pointer manipulation technique is best for implementation and what features are “musts” and “must nots”.

5.2 Later Usability Tests

In later stages of usability testing we will run usability tests of “real-life-like” applications. This could be for instance a comparison of two user-groups using the same application or Web-site. The control group will be confronted with conventional flat screen landscapes, the test group will work with the hilly landscape functionality. The control and test groups again are to be divided in subgroups on the basis of at least two criteria: expert vs. standard user and standard device (mouse) vs. special device (track point, track ball and touch pad). Several parameters will be recorded, such as objective measurements like task completion time, error rates etc., as well as subjective measurements like verbal and non-verbal expressions of the test subjects. Additionally, the observation methods mentioned in the previous subsection, e.g. eye-tracking, will be applied too.

To provide a realistic situation, some auxiliary tasks are to be considered, e.g. installing the device drivers or plug-ins or activation and disabling of the functionality.

These later usability tests will also evaluate the implemented additional functionalities, visualization and modification of the screen landscape and how they influence user behaviour and performance.

Finally, statistical analyses will be used to identify overall and special advantages or disadvantages of the navigation support module.
6 Ongoing Work

In the initial project stage, which can be described as a preliminary investigation and analysis stage, we focused on the desired screen pointer behaviour and motion characteristics. The goal was to find suitable ways to manipulate the standard screen pointer behaviour and to select the most appropriate programming language and platform for implementation. To keep the implementation of the module as portable and flexible as possible, we aimed to find programming solutions for the screen pointer manipulation at the highest possible level, thus avoiding having to work at device driver level. After some exploration and experimental implementations, Java [9] with the Java 3D API [10] turned out to be the most prominent candidate. This alternative meets all requirements extracted from the relevant implementation issues listed at the beginning of Section 4.

At the time of writing, we are working on the implementation of three different screen pointer manipulation techniques which we call sliding, affixing, and friction. Using the sliding technique, within a gravitation field (i.e. slope) the screen pointer automatically moves in one direction (representing the down hill direction) forcing the user to “work” in the opposite direction to go uphill, or just let the screen pointer “slide” downhill. The affixing technique lets the screen pointer stay still in a slope as long as the input device does not produce any motion events. The gravitation force is first activated when the user starts to move the screen pointer in one or the other direction. The realisation of the friction technique is based on manipulation of the control-display (C-D) gain setting as described in [11]. A low setting represents the uphill direction and a high setting the downhill direction.

The next project step will be the specification of the navigation support module. The description of the navigation support module is rather (and easier) made by the emotions, reactions and behaviour it should awaken and triggered during usage, than by concrete functions. Therefore, the specification and description will be based on two parts, one “emotion” oriented and one function oriented part, as recommended in [5]. We suspect that this division will prove to be helpful when designing and developing such an ease-of-use oriented artifact as the navigation support module. A challenging specification subtask is the modelling of the navigation support module. UML will be used for modelling purposes.

Parallel with the implementation of the navigation support module we will work on the development of the usability process and typical methods. In the first phase we have to consider fundamental research, such as cognitive psychology to evaluate for example whether and how our model is influencing common interaction patterns, familiar action-feedback loops, etc. Perceptual, cognitive and motor aspects will be taken into account. This is important because our module touches different sensing channels in a new way. In distinction to e.g. devices with tactile feedback, our module does not give tactile feedback but aims to simulate motoric effort (going up a hill) by visual feedback. In the case of isometric joysticks, e.g. track points, the required extra pressure needed to go uphill results in a kind of force feedback.

237
In the later phases more or less standard usability methods such as expert reviews and usability test with users will be performed to prove the costs and benefits of our module. On the basis of a framework of standard methods we probably have to expand the available methods due to the specific questions mentioned above. Development of e.g. special software test tools could be necessary and will be considered in the project plan.

References
5. Cooper, A (1999): The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How To Restore The Sanity, Indianapolis, Indiana, SAMS.

238
Designing User Interaction for Face Tracking Applications

David Chatting, Jeremy Thorne

Content and Coding Lab, BTexact Technologies, Adastral Park, Ipswich, UK
{david.chatting,jeremy.thorne}@bt.com

Abstract. Face tracking could potentially become a powerful new technology in the Interaction Designer’s arsenal, providing new modes of access to computing and communications. However, such interfaces could be misused in ways that confuse and worse misrepresent users, leading to poor interactions. Careful considerations of the human face, facial expression and the characteristics of tracking systems are required to ensure responsible design. This paper begins to explore this necessary field of study highlighting the need for future experiments. We conclude that due to the imperfect nature of tracking systems, feedback is immensely important and we consider methods of providing this.

1 Introduction

Apparatus for tracking the facial expressions of humans have existed for some years [1][2]. Traditionally these have been captured using markers placed on the face, however using computer vision techniques markerless systems are becoming viable [3][4]. The inconvenience and expense of marker based systems limit their use to specialist applications with large budgets, the markers are quite obtrusive and inhibit performance. Markerless systems enable many to have access to this technology using only an inexpensive camera, in a variety of locations and situations. There are a wide variety of applications including: mobile videophones, animation, broadcast television, accessibility aids and affective computing. Using such technology there is the scope for creating extremely natural computer interfaces and communication systems. However, we present some properties of such applications and argue that these systems require special attention.

Our current work is in the development of a markerless face tracking system for the Prometheus project [5] a three-year collaborative LINK project under the Broadcast Technology Programme funded by the UK DTI and EPSRC. The project includes markerless face and body tracking, actor and clothing model animation, scene construction and three-dimensional display technologies. It is seeking to build a virtual production chain for 3D television to encapsulate these technologies.

The Prometheus face tracker uses a single camera to capture the actor’s performance in real-time, using a tracking algorithm derived from [6] and described by [7] from which it derives the head’s orientation and the facial expression. This can be encoded as an MPEG-4 stream [8] and be used to animate a photo-realistic head.
The thoughts presented in this paper are the result of our own experience using markerless face trackers over the past three years and the application of research from other areas to this new field. As yet we have not verified our hypotheses by experiment. Our hope is to open up this area for further study.

2 Characteristics of Face Tracking Systems

It is important to understand the nature of face tracking systems in order to design responsible user interfaces.

Figure 1 shows the minimal set of components required for a face tracking system: the human face, expression measuring apparatus generating an abstract representation of that face and an interpretation stage, where that representation is manipulated. Arising from these components any face tracking system will exhibit the combination of their characteristics, regardless of its form. We also consider there to be two broad system classes; Facial Human-Computer Interaction (FHCI) and Human-Human Computer Mediated Facial Communication (CMFC). This section considers the general characteristics and those of the two classes.

2.1 Characteristics of a Minimal Face Tracking System

The human face is an extremely mobile and flexible device. There are enough muscles in the face to distort the face into over 7,000 unique facial actions [9]. The face can move fast too: the smallest perceptible expression exists for as little as one twenty fifth of a second [10]. However the features and their placement on the human face are universal and in addition Darwin [11] claimed that there are universal expressions that are recognised cross-culturally.

The expression measuring apparatus transforms the motion of the face into some abstract representation of the expression. There will inevitably be some optimal operating conditions e.g. the actor is within the field view of the camera, the lighting is correct, and some constraints on the range of facial motion detectable.

We will only consider here computer-based systems where the representation of expression will exist as a digital sequence. The sampling process required to convert continuous facial motion to a discrete digital sequence will necessarily introduce quantisation noise and have a limited frequency response.

The effects of digital representation are also dependent on the specific representation we use for expression. There are many existing schemes including:
FACS (Facial Action Coding System) [12], MPEG-4 FAP (Facial Animation Parameter) [8], AMA (Abstract Muscle Action) [13] and FAML (Facial Animation Markup Language) [14]. Any conceivable scheme is an abstraction of reality and as such will be more or less suitable for some applications. For instance, systems implementing the original FACS scheme can not represent asymmetric expressions, but FACS can provide a high-level description suitable for machine classification of emotion [12]. The MPEG-4 FAP coding gives a lower level representation defining the displacement of known control points on the face along significant axes. Associated displacements along other axes must be interpreted and cannot be represented explicitly in the coding. Since MPEG-4 does not prescribe how the interpretation should proceed, expression reconstruction will vary between implementations.

Inevitably there is delay between the expression being posed and its complete interpretation. There will be a trade-off between accuracy and system performance, which will be made according to the application; be it real-time or off-line.

Given the mobility of the face, the issues of representation, and the current state of the art, it seems likely that any tracking system will remain ‘blind’ to some classes of expression in the foreseeable future. The tracking equipment and representation of expression should be chosen to match the requirements of the ultimate application.

2.2 Characteristics of Facial Human-Computer Interaction (FHCI)

We consider computer interfaces driven by human face tracking as belonging to a class of interaction that Nielsen calls “Non-command User Interfaces” [15]. These interfaces are those which do not rely on formal commands and syntax, such as command lines and WIMP (windows, icons, menus and pointer) paradigms. Examples of such are gesture recognition systems [16], eye tracking [17] and ubiquitous computing scenarios [26].

Eye tracking systems are cited as an example of a non-command style interface and many of the issues surrounding eye tracking are also common to face tracking. For example, care must be taken with the interpretation of eye tracking data, since the movement of the eye is far more complex than the viewer is consciously aware. The eye makes frequent saccades and even when fixated there is significant jitter. Also the distinction between exploratory and intentional eye fixations must be made to avoid the ‘Midas touch’ effect where everything the user looks at is selected, for instance every folder is automatically opened even when the user is merely reading its name [17].

Similarly with facial expression we are generally not directly aware of our current expression and many of the movements we make are not necessarily intentional. It is impossible to disengage the face, it is ‘always on’ even when distracted by something else. Jacob [17] suggests the use of a ‘clutch’, so pressing a key or making some other form of input to release the face.

While the eyes have a clear mapping as a pointing mechanism, the face is much more flexible and the high level concepts expressed are less easily mapped to computing tasks. More “natural” input modes does not necessarily lead to more intuitive interfaces.
2.3 Characteristics of Human-Human Computer Mediated Facial Communication (CMFC)

Human-Human Computer Mediated Communication is concerned with enabling real-time communication between two or more people via computers and networks.

We seek face to face communication as it facilitates non-verbal communication, which helps us gauge the attitude of the other party with more confidence. The generation of facial expressions is in part at a subconscious level, it is difficult to have conscious control of the expression on the face all the time. For instance, few people can artificially pose a convincing smile, as they are unaware of the spontaneous timings that occur. Our true emotions are hard to suppress and professional actors are considered skilful, so we tend to trust the facial cues we see [18].

However, where facial expression is represented digitally, it can be duplicated, stored and modified; as with any other binary sequence. We can imagine a face tracking system where dishonest faces are caught by the system and replaced with amiable expressions. It becomes possible to manipulate consciously something that was largely out of control and digital facial expression no longer naturally reflects our true emotions. There is clearly much scope for deceit. We should therefore question whether there is any basis for trust in this scenario. Will this devalue this means of communication in virtual environments? [18]

Donath [18] discusses many implications of using representations of a human face in the design of computer-mediated human interactions; including the perception of identity, social identity and expression.

In certain circumstances it might become desirable to hide our facial identity or only selectively reveal different attributes to different people [19] due to the large number of social assumptions inferred from the appearance of a face.

3 Feedback in Face Tracking Applications

From our previous analysis face tracking, applications necessarily have limited capabilities. The user should be informed of these by a responsible application, allowing them to adjust their behaviour with predictable and so learnable results. We introduce here three feedback mechanisms through which the user can become aware of an application’s capabilities: Reviewable, Reflective and Indirect.

3.1 Reviewable (Non-real-time feedback)

An application is Reviewable if it allows the user to review how the machine has interpreted the face, before it is committed. This essentially allows the “undo” function with which we are familiar in the desktop paradigm. Making the actions of the user reversible inspires confidence and promotes the exploration of the systems. Examples of this include animation applications where face tracking data can be iteratively reviewed and tweaked by the artist until the desired character manipulation is achieved.

However, there are some classes of face tracking applications where this review is impossible. For instance in real-time telecommunication, where the machine must interpret the face and reconstruct a rendering of it at the far end instantaneously.
3.2 Reflective (Real-time feedback)
An application is Reflective if it presents to the user a real-time view of the machine’s interpretation of their face. This should allow the user to learn the cause and effect relationships and the degree of expression that stimulates large or small responses. Most face tracking systems today incorporate some augmented self-view indicating the computer’s interpretation [3][4].

However, there are some applications where feedback in this way is difficult. Consider a machine where a display (optical, aural or haptic) can not be physically incorporated into the device due to the small size, or other operating constraints.

3.3 Indirect
In everyday social situations we can not see our own facial expression, only gauge them indirectly via the response of others. Applications may be designed using a similar feedback approach.

For example consider Figure 2 which illustrates a communication system. Participant 1 only sees a reconstruction of Participant 2’s face and vice-versa; hence each is aware of the other’s reactions. However as we have previously discussed, any system will have capabilities and limitations, being blind to some faces and misinterpreting others. There are two stages of interpretation in this loop, each potentially having a different set of characteristics, both of which can manipulate the reconstructed face in unintended ways. This may result in misunderstanding and confusions, especially where each participant is unaware of the limitations of the underlying system. With necessarily imperfect interpretation stages this style of feedback may prove be hazardous.

4 Face Tracking Applications
We present here a set of scenario applications and discuss their properties and design challenges with reference to the application classes and possible feedback mechanisms that could be employed.

4.1 Mobile Videophone (CMFC)
The problem of reliable translation of facial expression is of particular importance in telecommunications applications; for instance a mobile videophone arrangement
where the participants’ faces are tracked and drive virtual masks visible to the other. The most obvious advantage of tracking and virtual masks over coding the video is the reduced bandwidth requirement for high level descriptions of facial motion. In this scenario the expression must be interpreted in real-time and sent immediately to the receiver, so this can not be made a reviewable system. We would discourage designing an indirect system for the reasons discussed. However it may also be difficult to implement a reflective system using video due to small screen sizes for handheld devices, and alternative output modes should be investigated.

Here the system has an obligation not to misrepresent. Given this situation there must be overwhelming evidence before a “dangerous face” is pulled. A dangerous face is one that expresses an extreme emotion, for example an angry snarl or a suggestive wink; these may differ between cultures. Human translators of conversation face a similar problem, although in most circumstances they are able to verify the speaker’s intention. For these reasons it would seem likely that such systems would tend to be used for relatively passive exchanges.

4.2 Animation (CMFC)
In this scenario the goal is to create a stream of data that can be applied to the head of an animated character to recreate a believable sequence of facial expressions, emotions and lip synchronisation. The quality of the data is of the greatest importance; in terms of frame-rate, range and smoothness of expressions. Thus the priority of the tracking application is to capture the raw data with as much precision as possible, so that post processing to determine facial motion has the greatest chance of success. That is to say capturing uncompressed frames of video from high quality cameras at a high frame rate, that can be analysed by the computer at its leisure after capture has completed. Gleicher discusses the challenges of using vision techniques to drive animation in [20].

We feel a degree of real-time feedback is important to demonstrate to the user that the video input is acceptable. The system should also be reviewable to enable the user to determine and correct (by re-recording or manual editing) any misinterpretations made in the processing.

4.3 Broadcast Television (CMFC)
Using face tracking to code video or drive 3D facial models as part of a virtual production chain such as Prometheus [5] provides advantages for broadcast (e.g. news or chat shows) in that the output can be scaled more intelligently for different transport networks and end devices.

As with animation, quality is the main priority, and a delay while processing occurs is acceptable. However if the broadcast is “live” then there is no opportunity to review the computer’s interpretation before broadcast. The real-time reflective interface must show sufficient detail to allow the producer to make quick decisions whether to trust the output of the computer. Particularly with news broadcasts it is again important not to pull dangerous faces with extreme emotion that might put a different slant on the content. Hence a fairly emotionless, cautious approach is required in the tracking.
4.4 Accessibility Aids (FHCI)
Non-intrusive facial tracking provides obvious advantages for users who find traditional input devices – keyboard and mouse – difficult or inconvenient to use. Jacob [17] suggests that for quadriplegics or for users whose hands are occupied (e.g. pilots) eye or face tracking interfaces provide significant benefit, even if they perform only minimally well because the users have no other available method of input.

If face tracking is to be used for controlling user interfaces, the tracking algorithm must run in real-time but not absorb a major share of computational resources, since other tasks must be able to run concurrently [21].

This scenario provides some inherent feedback since the user can see how the interface has responded to their actions through menu option choosing, or navigation through virtual worlds. However, because of the potentially dangerous consequences of some of these actions, the interface should also be at least partly reviewable with the options to confirm actions or undo mistakes.

Existing techniques have focused on gross head movements or eye tracking, but as techniques and computing power improves the ability to detect and process more subtle gestures and expressions will become possible. This will lead to richer interaction and an increased bandwidth flow into the computer [22]. However the gestures must be natural to avoid fatigue through repeated actions.

4.5 Affective Computing (FHCI)
Affective computing is concerned with responding appropriately and sensitively to user’s emotions. For example MIT [23] have developed a CD player that plays music based on the listener’s current mood and listening preferences. There are applications of affective face tracking in ubiquitous computing environments, where the aim is to make many computers available throughout the physical environment, while being effectively invisible to the user.

Emotions and mood changes are generally gradual and the computer will only become ‘aware’ of a user’s mood over time. Hence continuous real-time feedback is inappropriate, as is occupation of a large portion of the available display space. A small visual token (e.g. icon, light, flag) or ambient change may be more useful. Existing research has already addressed classification of emotion in faces [12][24] and real-time systems that combine this with face tracking are feasible in the near term.

5 Discussion
Having presented our views on the classes of face tracking applications, some mechanisms for providing feedback and some sample applications we now discuss some of the specific issues that arise for interaction design.

5.1 Manifested Interfaces
We observe that the interaction may be constrained and not fully natural. Consequently face tracking applications must expose the system’s capabilities and make them apparent to the user so as to allow their actions and the reaction to be
predictable and learnable. The interface mustn’t suggest too much or too little functionality.

It has been suggested that “Face-to-Face Implies no Interface” [25]. Ideally this would be the case, but practically given the imperfect nature of the tracking and representation, we consider a manifested interface to be essential. We must prevent failures in the system being responsible for escalating failures in the communication. In their description of Ubiquitous Computing, Mynatt and Nguyen [26] note that, “…systems rely on implicit sensing that is naturally ambiguous or error-prone, it is up to the designer to help users comprehend the, sometimes variable, limitations of the system.”

In dealing with error, there are two ways in which the tracking can fail: the system can lose track of the subject (system failure), or it can misinterpret a facial expression (false reading). In the first, the computer is aware of its failure and alerting the user can be straightforward. In the second, the computer makes a mistake on a single frame, but doesn’t realise it. In addition the user may also not spot that the feedback interpretation is incorrect. That is, we suspect that if in general the tracking is perceived to be correct, quick aberrations will go unnoticed. However in CMFC systems, the recipient may notice the mistake, particularly if the expression posed was “dangerous”. Therefore if the system is to allow dangerous faces and not cautiously filter them out, then feedback must specifically alert the user to their creation.

In situations where a reflective interface is appropriate a ‘Magic Mirroring’ metaphor provides a useful tool. The user can see themselves on the screen as they face it; it appears like a mirror which augments the view with computer generated markers. For this metaphor to be apparent the screen and camera must be collocated and the image must be flipped horizontally in order to maintain the illusion. This is the interface currently employed in the Prometheus tracker.

The choice of markers is important. Some existing systems [3] overlay the face with spots that reflect the image features being tracked. The augmented face appears as if physical markers had been stuck to the skin. This interface seems appropriate if the captured data is used to reconstruct the same feature motions on a remote model of the face, however no clue is given as to the computer’s understanding of the expression. For example in lip tracking enough information could be inferred from the points around the lips to correctly reconstruct the face, but the unseen classification as happy or sad could be incorrect. Here, more iconic markers may provide an advantage by giving the user a pictorial representation of the computer’s understanding – for example a symbol depicting creased lines overlaid on the brow to represent the detection of a frown. It is possible to use to give the user feedback pre and post interpretation. For example displaying of both the feature detection and the 3D puppet.

If a true reflective interface is not appropriate – because the display is occupied by something else, or there is no display - subtle feedback is still possible through the use of colour, sound or other modes.

5.2 Interface Paradigms
As we have demonstrated a number of application scenarios in which we feel it is desirable to have face tracking based interfaces. In these circumstances the expression
capture equipment (for instance camera) may be the primary or sole input. The traditional interface paradigms of pull down menus and windows have been designed for mouse and keyboard operations and a new set of primitives may be needed for face tracking based interfaces. The use of face tracking, or any other “natural” communications mode, does not necessarily result in intuitive interaction and requires careful design, as we discussed in relation to the problems of eye tracking.

5.3 Initialisation
Attention must be given to the design of the initialisation stage in which the face is first acquired. The delay and impositions of this stage must be endured by the user each time the interface is used. Here the face cannot be augmented or labeled as the system is not yet aware of it. Practically since the computer is still searching for the face a greater amount of processing is taking place that will inevitably lead to lower frame rates and an increased latency in the feedback. In a more ubiquitous scenario where the user is not directly aware of the interface, the initialisation becomes hidden and this acquisition stage becomes even more complex, as there can be no dialogue between human and machine.

5.4 Expression Representation
The internal representation for the interpreted facial expression should be chosen to match the target application. For instances where reconstruction or playback of the face is required low-level prescriptive representations are good since this reduces the requirements on the player to understand or interpolate the data. If understanding is required then progressively higher level representations and the tools to convert between these interpretations are crucial. High level representations are also key to reducing bandwidth requirements.

5.5 Trust in CMFC Systems
Trust is a fundamental notion a communications system, that the message sent matches the message received. In a CMFC system we have considered the dangers of the system pulling “dangerous” faces unintended by the user and the consequences for the discourse. Additionally we highlighted the responsibility of the system to have overwhelming evidence before such a face is sent. Additionally, where the sampling rate of the face acquisition is below that of the speed at which an expression can pass over the face we observe that brief expressions can become unobserved. Ekman finds “micro-expressions” of approximately 25th of a second to often disclose dishonest behaviour, where normal “macro-expressions” often last between half and a few seconds [10]. In such a system the recipient could not perceive these micro-expressions and would not be given the opportunity to judge the other as dishonest. However, Ekman showed visual lie detecting to be a difficult task at which many of us are poor.

All these factors mean that a CMFC user should be prepared for the representation of the other to deviate from reality. A responsible application should endeavour to match the users’ perception of the systems trustworthiness, with reality. It is our hypothesis that too much trust will be attributed to the system, leading to
communication difficulties. Assuming that to be true, we would need to introduce an element of mistrust in the user’s mind.

There are many ways in which we build trust relationships and there has been work to extend these notions to computer interfaces [27][28]. We are considering promoting a little mistrust in the interface, which requires that many of these principles are actively broken. In a face interface there are a number of ways in which we might change the user’s perception of trust, mainly through the appearance of the head, it’s animation and the voice. One approach may be to deliberately make the face unrealistic in ways that prompts the user to reassess it, perhaps inserting strange (although not deliberately disturbing) animation. The “uncanny valley” phenomena has been noted by Mori [29], where the more realistic the model becomes the less forgiving we are of small deviations, whereas we would be more tolerant if the model had a cartoon quality. This phenomenon may be of use in designing for mistrust. Further studies are required to investigate these ideas further.

6 Conclusion

Face tracking technology is available now. With advances in markerless techniques there are many situations in which it can be valuably applied. We do not believe that these “natural” interfaces inherently provide natural and intuitive interaction, considering there to be serious design challenges that must be addressed in responsible systems. We have sought to identify a number of these challenges and further work is required to test our hypotheses.

We have considered how the limitations of these systems necessitate feedback, to provide the user with a view of the machine’s understanding and interpretation of the scene, aiding learning of its capabilities. We have discussed three possible feedback mechanisms; Reflective, Reviewable and Indirect. We consider that a Reflective or Reviewable interface should be implemented whenever possible. Additionally, we have described a set of scenarios where face tracking may be applied with reference to these concepts.

Given the human face as an extremely subtle communications mode, we have highlighted some means in which it can be distorted through a FHCl and considered how we might design such applications to promote a little mistrust to prevent communication problems.

Acknowledgements

We are grateful to Anna Haywood, Daniel Ballin, the members of Content and Coding Lab and the Future Content Group of BTexact Technologies for their advice and comments.
References

3. Eyematic Interfaces: FaceStation — http://www.eyematic.com
AWUSA – A Tool for Automated Website Usability Analysis

Thomas Tiedtke, Christian Märtin, Norbert Gerth

Augsburg University of Applied Sciences
Department of Computer Science, Baumgartnerstrasse 16
D-86161 Augsburg
{tiedtke, maertin, gerth}@informatik.fh-augsburg.de
http://www.fh-augsburg.de/~tiedtke/awusa

Abstract. This paper presents a framework for automated website usability evaluation that combines different techniques for data-gathering and analysis. The AWUSA approach is based on the combination of information architecture, automated usability evaluation and web mining techniques. Its key components were implemented as an analysis tool.

1 Introduction

Good usability is a key quality of any business website. It is needed to clearly articulate the business goals of a visited site and it enables users to accomplish their tasks more effectively by choosing the right actions and navigating to the right pages. Many dot-coms have failed during the last couple of years, because of missing customer retention and repeat sales. Poor usability of their websites is one of the major causes of their failure [3].

Usability evaluation in the classic sense is often conducted under defined test conditions in a lab environment. However, typical problems that may arise from real usage of business-to-customer (B2C)-systems can only be discovered by focusing evaluation tools on real websites and real users. In order to allow for useful results, relevant issues like information architecture of the target application, website design and usability questions have to be taken into account by the evaluation process.

In a generic layered model of business websites, the marketing and the usability layers seem to be separated from each other, at first glance. The usability layer lies beneath the marketing layer. The marketing layer defines issues like products, prices and target customers. However, some marketing aspects, e.g. the definition of target groups and business goals may directly interfere with usability issues and therefore have to be taken into account when an overall usability assessment of a site is done.

The AWUSA usability analysis system presented in this paper covers these aspects. It uses a task based approach and integrates usability issues with topics of the information architecture of websites. In AWUSA the analysis of a website and its
usage is divided into static analysis at design/definition time and dynamic analysis at analysis time. In both phases statistical and data mining techniques are applied.

The produced results are important for improving a website and for reflection on the underlying information architecture. As we base our approach on real usage data, the achieved results contain predictive information about the interaction behaviour and mirror realistic situations.

The AWUSA prototype was implemented in XML [17] and Java. All input and output data, as well as processing data are represented in XML. For graphical output the scalable vector graphics (SVG) [17] is used.

This paper is structured as follows: The next chapter Related Work discusses relevant work related to this paper. Chapter 3, The AWUSA Integrated System Approach, proposes a generic framework for analyzing websites and their usability and a prototypical realization of the framework, the AWUSA System. Some limitations and practical aspects are covered in chapter 4, Discussion. Chapter 5, Conclusion, summarizes the integrated approach and the resulting AWUSA system.

2 Related Work

This chapter describes research topics and work that is related to this paper.

2.1 Log file Analysis

Log file analysis supplies basic statistical results e.g. summarizing or average counts. Common results are the percentage of browser usage, access counts of given resources, the origins of the users of a site etc. Log file analysis is fundamental to subsequent usability and mining activities. Many commercial products for log file analysis exist. Two examples are Webtrends and HitList [1].

2.2 Automated Usability Evaluation

Usability evaluation in its classic sense mainly focuses on defined (laboratory) experiments. One approach is user testing, where the user has to accomplish some given tasks (the most often used method is the think-aloud protocol while the user is clicking through the site). Usability-expert walkthroughs [13] are another approach. User testing is partly dependent upon cooperative test users. Both types are time-consuming, but helpful during the contextual design of highly individual software and in software developments for the mass market, where the user remains unknown.

Several approaches can be found that try to automate usability evaluation. Most of these approaches fall into one or more of the three categories usability data capturing, analysis or critique. A good survey is given in [10].

In the area of web usability evaluation tools there is a wide range of approaches from syntax-checkers over web page design advisors [7] to analyzers, which compare
the characteristics of web page objects with empirically found metrics. A system realizing the latter approach is WebTANGO [9].

A recent approach for automatic analysis is WebRemUSINE [14] which compares recorded real user interaction information with the task model of the web application. The tool is capable of identifying tasks by event/basic task association. The tool discovers whether a task was performed, which amount of time was consumed and which (navigation) errors have occurred.

In [5] a comprehensive approach for modeling the predictive behaviour of web users on the basis of WWW protocol analysis is presented. It is planned to make the method replicable for generic use and to develop a WWW protocol coding guide that is based on the results of a web survey among 2000 users.

2.3 Web Mining

Web mining can generally be divided into three areas: Web usage mining, web structure mining and web content mining. A survey of the field is provided in [12]. All three approaches have the same goal: Extracting more information from the web and its huge amounts of data, e.g. about the usage of particular websites, about the structure of one or several websites, or about the content of a website. Web structure mining and web content mining are used for analyzing the information architecture of websites. Our approach focuses on web usage mining. It can be characterized by the steps identification of users, identification of single sessions and identification of the navigation paths within these sessions. In [4], [6], [16] and [21] possible solutions for implementing these steps are given.

3 The AWUSA Integrated System Approach

In this chapter AWUSA – Automated Website Usability Analyzer, a comprehensive framework for automated website analysis and usability evaluation is presented together with the prototypical implementation of its key components. In contrast to other automated usability evaluators AWUSA combines in one system components for log file analysis, automated usability evaluation and web mining. The Extensible Markup Language (XML) and Java serve as a platform for the suggested framework.

3.1 Goals

The framework incorporates various techniques for capturing relevant data and website/web page analysis, illustrated in figure 1. A system that is constructed according to this approach has the following goals for automatic website analysis:

Find and visualize users’ paths on the website
Find and visualize deviations between intended tasks and actual usage
Find and visualize locations/events where tasks are canceled prematurely
Find and visualize areas and situations with poor usability
Provide plausible reasons for detected usability problems
Find new goals/tasks of users while touring the website
Classification of different user groups and their mapping to the various tasks/goals

Fig 1. AWUSA’s integrated approach.

Interpretation and visualization of usability data are part of the approach as well. Marketing aspects like the overall purpose of the analyzed website (e.g. community, information, shopping site) and target groups are also exploited during analysis.

3.2 System Model

The underlying web mining process that is applied in order to reach the goals formulated in 3.1 can be characterized by the following sequence of activities:

1. Data Generation
2. Usability Problem Detection
3. Visualization

The data resources, used by the AWUSA process, are described in 3.3. AWUSA uses static and dynamic analysis techniques. The static website and the webpages are analyzed at design/definition time. For each web page several metrics are generated. Especially the Document Object Model (DOM) [17] and the visual representation of
web pages are analyzed. Web content mining and web structure mining are applied in this phase [15]. Later the log data is preprocessed. Sessions and session paths are generated from the access log files and analyzed. (step 1)

Usage data is compared with the defined task structure and data mining techniques are applied to defined, gathered and generated data. During dynamic analysis web usage mining is applied for path mining, cluster generation and detection of association rules (step 2).

Finally the defined structures and the results of the analysis are visualized (step 3).

3.3 Data Resources

In the following the major data resources used by AWUSA’s components are discussed in some more detail.

3.3.1 Abstraction levels

Information in AWUSA is organized in several abstraction levels from physical resources (directories and files) to goal/task-structure as shown in Figure 2.

![Abstraction levels in AWUSA](image)

Fig 2. Abstraction levels in AWUSA

On the lowest level the physical resources are located, i.e. the files stored in directories on a web server. All we know about these files is the information provided by the operating system.

Logical resources form the next level. Resources are either master resources (e.g. web pages) or auxiliary resources (e.g. images) that are assigned to a master resource
via the embedded links in the HTML code of the web page [6]. From these logical resource objects the information architecture of a website (as described by the next levels) is built. For the rest of this paper resource and logical resource are used synonymously.

The (navigational) links (see 3.3.3) constitute the next layer, the link structure, which is the basis for all navigational paths on a website.

The information structure level divides the website into several sections with logical resources (e.g. shopping or information section).

Finally, the task/goal structure defines the business model of the website. The task structure of a website defines the goals of a website and the tasks, subtasks and logical steps to accomplish a goal. The goals are typically defined by an information architect during design. To reach a goal, a task has to be accomplished. Examples for a goal are the purchase of a product or the viewing of a video. A task describes the basic steps a user needs to perform in order to reach a goal. The underlying task structure corresponds to the task ontology presented in [20].

In AWUSA tasks are associated to sequences of logical resources, so-called task paths. Each task is associated with one goal. Different task paths can be executed in order to reach the same goal. The final logical resource in a task path is called a goal resource and symbolizes that a task was finished successfully.

A task “purchase”, for example, has a taskpath consisting of the following resources: the user navigates from the homepage (path step 1) to a web page containing a list of products (step 2). From this list the user selects one product and gets a detailed description of the selected product (step 3). By clicking the button (link) “buy this product” the user is taken to an order form (step 4). After filling out this form and sending it, he or she arrives at a page where the purchase is confirmed (step 5 and goal resource).

The AWUSA approach also includes the concept of target user groups with different characteristics [2]. Different target user groups can be assigned to one website or particular sections of the site. Task paths can be assigned to target groups.

3.3.2 Web Page Data
To provide data for the subsequent web usage mining process, a static analysis of the web pages is carried out first. Static analysis provides information about code and object structure as well as the layout of web pages. Code analysis parses the (X)HTML code of the web page and discovers syntax errors, areas of non-conformance to coding standards and browser compatibility problems.

Object structure analysis provides item counts for graphical and image elements, dialog boxes, search masks, audio/video multimedia elements and links to other pages. From these numbers usability-relevant information like the text to image ratio is computed [9]. Finally the layout analysis provides positioning data for the various elements, grouping information, colour information and size information [7].
3.3.3 Website Data

Parts of the description of the static structure of a website are automatically generated. Other parts are typically defined by the information architect during design. The static structure covers *meta-information* about the website, e.g. type of the site, target group, language, hyperlink structure. The hyperlink structure that connects the logical resources of a website is generated by a tool (WebsiteCrawler) and are represented as graphs.

Resources are mainly web pages, but can also be other objects like non-HTML documents and multimedia objects (video and audio files). Each resource is characterized by a type. The following table shows some of the resource types used in AWUSA. This categorization is based on [6].

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>HomePage</td>
<td>first page of the website</td>
</tr>
<tr>
<td>FrontPage</td>
<td>first page of a particular section within a website</td>
</tr>
<tr>
<td>NavigationPage</td>
<td>page that is mainly used for navigation</td>
</tr>
<tr>
<td>ContentPage</td>
<td>page that contains the actual content the user is searching for</td>
</tr>
<tr>
<td>FormPage</td>
<td>page containing a form</td>
</tr>
<tr>
<td>FormConfirmation</td>
<td>page confirming the processing of the form data (“thank you ..” — page)</td>
</tr>
</tbody>
</table>

Links within the website are categorized according to their function, like in [21]. The following table contains examples of link types.

<table>
<thead>
<tr>
<th>Link Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NavigationLink</td>
<td>link within the basic navigation grid</td>
</tr>
<tr>
<td>ContentLink</td>
<td>link located within the (text) content section of a webpage</td>
</tr>
<tr>
<td>Function link</td>
<td>link that is used to realize a specific function (printing, closing the window)</td>
</tr>
</tbody>
</table>

Website data also include some metrics information like page counts, section counts, pages per section, width to height ratio. For the pages of a website also interpage metrics are kept that inform about the link distance between pages, conceptual similarities and visual similarities of the pages.

3.3.4 Usage Data

Usage data is extracted from access logs, application logs (transaction data) and some additional log files. Prior to analysis the raw log data have to be preprocessed. This leads to preliminary results which are stored within the usage data base. The preprocessing steps are needed for data preparation and include:

- Elimination of unreal users (robots, search engines, test purposes)
- Identification of erroneous requests (they may indicate usability problems)
Generation of an XML representation of one request to a logical resource from several requests to physical resources. The association from master resources to auxiliary resources as indicated in the source code of the master resource is exploited for this purpose. Missing auxiliary resources are logged (failed access). Data reduction which may be necessary on large websites. In these cases only part of the logged data or a random sample is preprocessed for further exploitation.

Static usability problems can be recognized before the actual usage of the website. Such problems include dead links, dead ends due to bad structure, HTML syntax errors, overloaded web pages.

3.4 Data Generation

In figure 3 the overall architecture of AWUSA with its data resources, components and dataflow is shown. The architecture is described in chapter 3.4, 3.5 and 3.6. Parts underlayed with grey colour indicate the prototypical implementation of the framework’s key tool components.

3.4.1 Website Data Generation

The static data (see 3.3.2 and 3.3.3) for analysis is generated by the following components:

The WebsiteCrawler walks through the entire website and generates an XML document WebsiteData containing data about the website. Data about each resource as well as about each link is stored. The link structure is checked by the information architect. The information architect defines the information structure, i.e. the sections of the website. He or she also defines the task/goal structure. In this way he or she creates the underlying structure and defines the goals of the website which are communicated to the analysis system.

Additional data for each resource (e.g. web page) is gathered by a WebpageAnalyzer which provides metrics information like word count, link count etc. Weighted keywords are used to describe the content of the web page. This data is stored in the document Results.

3.4.2 Usage Data Generation

The usage data base is generated mainly from the access log file and from the transaction log file (see 3.3.4), created by the web-application.

After the data has been preprocessed (extraction of non-interesting log-data), path mining is applied, which performs user identification, session identification and path completion. Path mining in detail is described in [8]. Users and their complete session paths can also be implicitly identified by integrating an access logging framework into the web application.

AWUSA focuses on analyzing the identified and completed paths. Usage data as the base for the following analysis and processing steps is stored in the XML-Document UsageData in the form of sessions with session paths.
Fig 3. AWUSA architecture and prototypical implementation (grey shaded)
3.5 Usability Problem Detection

Usability problems are grouped into the categories *interaction*, *information architecture*, *design* and *technical*.

<table>
<thead>
<tr>
<th>category</th>
<th>examples for usability problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>interaction</td>
<td>form/dialog box-problems, functions (print, close window),</td>
</tr>
<tr>
<td></td>
<td>orientational problems, no navigation within the content,</td>
</tr>
<tr>
<td>architecture</td>
<td>missing cross-links</td>
</tr>
<tr>
<td>design</td>
<td>inconsistent <em>look and feel</em> of the website</td>
</tr>
<tr>
<td>technical</td>
<td>syntax errors, broken links, overloaded pages</td>
</tr>
</tbody>
</table>

There are two different ways of detecting usability problems: Directly, by analysing the website and especially the webpages. Doing this, the analysis component discovers usability problems like broken links, wrong HTML syntax, overloaded pages etc. The other way of detecting usability problems is indirectly by analysing users’ behaviour on the website and comparing it with usage patterns that are known for their inherent usability problems.

The AWUSA system interrelates the two ways of usability problem detection. Different analysis and knowledge discovery techniques are applied in order to create an integrated analysis framework. Basic concept is the combination of static and dynamic data. This is used to establish relations between the static structure and content of the website and its usage with the help of data mining techniques.

The AWUSA assistant (figure 4) helps the analysis operator to perform the AWUSA process consisting of generation, analyzing/detection and visualization.

### 3.5.1 Usage Pattern Miner

The first step performed by the usage pattern miner extracts significant patterns from the usage data containing the session paths. This is done by cumulating similar session paths. A similarity ratio for each pair of paths is computed, corresponding to the sessions’ resource sequence. Sessions with a ratio greater than a given threshold value are cumulated to usage patterns.

The usage patterns have one main path which is weighted according to the number of users traversing it. References to all requested resources are added to the usage pattern along with their target resources and the number of users traversing each target link. This is done in order to not only achieve a straight pattern path but to also generate usage patterns along with their surroundings (fuzzy patterns).

### 3.5.2 Usage Pattern Analyzer

In the next step the extracted patterns are analyzed. This is done by comparing the patterns with the underlying website structure contained in the WebsiteData XML document. Entry- and exit-resources are of special relevance. An interesting question is: How many users enter the website via entry- or nonentry-resources, how many users leave the site via non exit resources, and why do they so?
Statistical techniques are applied to extract data about the request count, the number of traversing users etc. Data mining techniques are applied to generate association rules. Data mining can be divided into several steps: First, associations between requested resources are detected (e.g. a user that visits page A will visit page B). In the second step it is found out, which attributes of resources determine a noticeable navigation behaviour, especially break-ups and strange transaction data. Therefore data mining relations are built by the usage pattern analyzer and passed to the mining engine. This process will discover coherencies between attributes of resources and significant usage behaviour with respect to these resources.

Usability problems can also be detected by comparing the detected patterns with stored patterns that are considered to be symptomatic for usability problems. Some of these navigation patterns are described in [21]. The results of this pattern analysis are stored in the XML document Results, which is passed to the Result Analyzer later.

### 3.5.3 Task Usage Miner

The task usage miner extracts information about the task’s usage from the XML-Document UsageData. This is done by assigning usage patterns to defined task paths that are contained in the XML document WebsiteData. AWUSA uses two methods to assign usage patterns to particular task paths. First, by selecting all patterns with a similarity ratio greater than a given value. The second way is to assign usage patterns to a task path that contains its goal resource. The task path usage information can directly be passed to the task usage analyzer which is implemented within the same Java application. It can also be stored in the intermediate XML document TaskUsage.
3.5.4 Task Usage Analyzer
AWUSA distinguishes between successful and not successful usage patterns. A pattern is successful with respect to a task path, if it contains the task’s goal resource. As a first analysis step the success of task usage is evaluated. After this the single usages of task paths are analyzed. By comparing the defined task paths with their assigned usage patterns deviations and breaks in task-path navigation can be detected. Resources related to such significant spots are marked as problem resources and added to the XML document Results. Navigation data of this kind is further analyzed.
Successful task usage paths with no or a low similarity ratio to the defined task paths can be considered as new task paths, if a significant number of users take these paths in order to accomplish the task (i.e. if they reach the goal resource).
Statistical techniques are applied to generate metrics information about the usage of the task paths. The traversal ratio describes the number of users traversing a task path. The task completion rate describes the rate of users that complete a specific task.

3.5.5 User Analyzer
One goal of the approach is to detect user groups. In order to achieve this, user and navigation data is clustered. The characteristics (connection speed, browser systems etc.) and navigational behaviour of the detected user groups are compared with the characteristics of the defined target groups. User analyses can be performed website wide or for particular tasks or task paths.

3.5.6 Result Analyzer
The results (problem resources, evaluated task usage, statistical metrics, association results) are stored in the XML document Results. The results of the analysis steps are interpreted using stored interpretation rules and patterns. For detection of possible relations between problem resources (e.g. breakoff from task path), their usage pattern and their statical attributes data mining techniques can be applied. The mining engine tries to generate possible reasons for these negative data by generating association rules that describe relations between statical data and usage data. The relevant attributes and relations found at this step are stored in the resulting XML document.
To complete the analysis process, final statistical metrics (e.g. count of problem resources, ratio of successful/non successful patterns) are computed from the results.

3.6 Visualization
Visualization is another major part of the AWUSA approach. The goal is to generate an interactive visualization that can be understood intuitively and interpreted by the target groups of the analysis system. The website and all discovered results are visualized based on the website’s information structure. The visualization is designed for information architects, designers, programmers and website publishers (e. g. the customer of the internet agency).
Visualization is based on a layer model. The base layer is the website, the display type is oriented at the website’s information structure. There are two major views on the website. The usage view shows navigation patterns and possible problem spots in general. Visualization of tasks and their usage is provided by the task usage view as shown in figure 5. The visualization engine offers the following services:

The website component displays the resources and main navigation links of a website according to its information structure. The resource type is also coded in the visualization, as well as several other attributes of the resources.

Resources that probably contain usability problems are marked by a special component. A screenshot and a direct link of the resource is offered.

The usage pattern component displays the most significant paths of the website according to the number of users traversing each path. Task paths and their usage are displayed in different colours. Deviations and breaks can easily be recognized.

Generation of the visualization again is done by an XML-application. The scalable vector graphics (SVG) is used for visualization of the website and the analysis results.

4 Discussion

Automation is an advantage, when we compare AWUSA to classical approaches. But classical usability evaluation methods also have some significant advantages, e.g. the possibility of directly observing the test user in a defined environment. Also some
problems cannot be detected by automated approaches, especially common sense reasoning problems. An ideal system would combine classical evaluation with an automated approach, so the advantages of both can be used.

AWUSA integrates real-world semantics into the process of website analysis. The information architect has to categorize the resources of the website and has to check the link structure etc. Later on, in the business of every day the static website data has to be kept up to date in order to get useful results from analyzing the website with the AWUSA system.

The purpose of the presented prototype is to show how the AWUSA framework proposed by our integrated approach can be realized and how XML can be used to combine the single components of the framework. Complete analysis of a big website with a lot of traffic is a performance intensive application and requires suitable hardware. Another possible way of using the system is to work on smaller random samples. In order to handle great data amounts, efficient XML databases could be used to store the XML objects and documents.

A very interesting field for further activity is the intensive analysis of single web pages, especially their visual representation. This could drive analysis even closer towards real user experiences.

In the foreseeable future, when XHTML will be broadly used instead of HTML, link types and other additional semantical information could be stored directly in the web page files using the XML namespace concept. The Resource Description Framework [17] and other semantic web innovations can be applied. Now that the XML family is based on a stable recommendation by the W3C, this is a very promising area of research.

5 Conclusion

Recently several approaches entered the field of automated usability analysis. However, some significant features distinguish AWUSA from other advanced approaches: The structural integration of the defined information architecture, semantical enhancements, analysis of business tasks and the visualization system. Another feature of the framework is the comprehensive and consequent application of XML. Thus, AWUSA offers a framework for integrating separate approaches in order to construct a global and flexible website usability analysis system.

Acknowledgements

The authors want to acknowledge the cooperation of argo tec GmbH, Munich, for supporting the development of the AWUSA prototype. We also thank the anonymous reviewers for their valuable hints.
References

Models for Task-Object-Based Web Site Management

Gerd Szwillus¹ and Birgit Bomsdorf²

¹ Universität Paderborn, Fachbereich Mathematik/Informatik,
Fürstenallee 11, D-33102 Paderborn, Germany
szwillus@uni-paderborn.de
² FernUniversität Hagen, Fachbereich Informatik,
Universitätstraße 1, D-58084 Hagen, Germany
Birgit.Bomsdorf@fernuni-hagen.de

Abstract. A concept is proposed to model the web site structure update process and its links to the business of the web site owner. The underlying web object life cycle model is based upon the business objects to be represented on the web and the tasks performed in reality. The model is introduced as core part of a proposed web site management environment, which allows web site owners who have minimal Internet know-how with adequate support to update web sites on their own, thus allowing them to keep the sites always up-to-date.

1 Introduction: The Web Site Update Problem and its Origins

The number of web sites in the World Wide Web is growing by several thousands every day. Almost every person, institution, or organization wants or is expected to be „on-line“. Usually, smaller groups do not have the know-how to create web sites themselves [10]. Web site owners such as a butcher, a professor at a university, a pigeon breeders club, or an art gallery, typically do not have the know-how to design, create, and maintain a web site on their own. Hence, a web design company is contracted, creating a web site based on a dialogue with the web site owner. To keep the web site up-to-date, the web site owner usually needs external help continuously. Available tools are not adequate to his knowledge and needs: Web editing and site management tools [5, 6], content management systems (CMS) [3, 9], or workflow management systems [4] are either too hard to use as they require too much Internet or database technology knowledge, are too expensive, or are of too coarse granularity, thus supporting document flow “in the large” but not single web page updates. So, for the “small web site” owner, every change has to go through the design company, which implies time delays and substantial costs, every time a change has to be made. The result can be observed throughout the web:

- Information on web sites is frequently out of date, e.g. an „upcoming event“ is promoted, with a date which lies in the past.
- or web sites show only static information, like in a colour brochure, thus making no proper use of the Internet as a fast medium relaying current information.
Due to the site owner’s inability to update the web in a timely fashion, no proper use of the possibilities to relay current information easy and fast to potential customers, clients, or people who are interested in the organization, is made. This diminishes the usefulness of the web site, and gives it incredibility in the eyes of its readers [1].

2 Task-Object-Based Web Site Management

In this paper we propose a model for the explicit specification of the web site owner’s web process. A Web Site Management System (WSMS) is presented which is based upon the tasks performed and the objects manipulated in the „business“1 and ultimately provides a sophisticated update assistance dialogue to the web site owner. The core part of this concept is the Web Object Life Cycle Model (WOLM). Starting point of the modelling process is a classical task analysis phase resulting in a task model of the „business“ describing which tasks are performed and which objects are affected in what way. This task analysis and modelling phase is restricted to the part of the business which is to be represented in the web. Once the information units this representation is made up from have been identified, they are described as web objects in the WOLM. Within this model, the life cycles of the web objects are captured, i.e. the conditions and effects of creation, state changes, and deletion of these objects. Overall this results in a model with clear behavioural semantics reflecting the web object modifications as triggered by the business. From this model, an individualized dialogue is created, which „speaks the language of the business“, and is employed by the web site owner to feed this representation with real data from his business, which in turn enables the WSMS to update the corresponding web pages accordingly. As a consequence, the only technical knowledge the web site owner needs when using this system is how to access the Internet and how to use a browser, to modify the web site on his own and without any time delay.

Content Management Systems [3, 9] provide the user with a data base interface, allowing the modification of the business data as stored in the data base. These changes then automatically trigger the corresponding modifications of the web pages. As a consequence of this architecture, however, the user interface presented to the web site owner is based upon the data base structure, hence it is data-centered and system-oriented, and not does not explicitly support the web site owner in performing his tasks.

1 Although the example we use in the following is from a non-commercial environment, we refer to the organizational unit which owns a web site and uses it as its Internet representation as „the business“ for the sake of simplicity of language.
3 Overview of the System

First, we present a big picture of the approach. The figure above gives an overview of the WSMS components and their interrelations. It shows the central role of the WOLM, from which both the individualized update assistance dialogue as used by the web site owner and the final web pages as seen by the web site visitor are derived. This second part is structured into two intermediate steps, capturing the web site structure and its contents and the web site layout information in separate components. The development of the systems starts with the creation of the WOLM. As a prerequisite, a task analysis of the business with respect to the intended web representation is performed. The result of this analysis is a task model of the business restricted to tasks which are supposed to influence the web site, i.e. adding, changing, or deleting information shown on the web pages. This task model is refined in more detail by means of the WOML.

The **Web Object Life Cycle Model** contains components corresponding to the "objects of the business" to be represented in the web site, as they have been identified during the task analysis and modeling phase. The WOML describes the modifications these objects undergo, based on the tasks executed in the business reality. The model is formulated strictly in terms of the business itself, and is oriented towards the events occurring and tasks executed in the business. It is explicitly *not* written in terms of web sites, pages, or web page elements, let alone technical categories such as HTML or XML files. We deal with the WOLM in more detail in section 4, "Web Object Life Cycle Model".

**Abstract Web Site Structure.** The abstract web site structure is derived from the WOLM. It contains a hierarchy of web pages, tables, lists, text elements, graphics, links, etc. It is a web site structure model, as it defines *what* is shown on the web site pages; it is called “abstract”, as it does not specify how things look. While the WOLM defines the information to be displayed and its modifications, the abstract web site structure defines the distribution of the information onto web pages, into lists, tables, texts, or graphics, and deals with navigation issues on the site. The
abstract web site structure is designed and linked to the WOLM within a distinct step. The connection between the two components defines the modifications the web site structure and contents undergo, if changes occur in WOLM.

The HTML design files are HTML files containing placeholders, referring to pieces of information in the abstract web site structure. HTML design files are combined with data from the abstract web site structure to create the finally published web pages. At this point, web design information is added to the process. The HTML design files can contain arbitrary rich design components, such as textual, graphical, animation, or multimedia elements. Although denoted as HTML design files, there is no restriction to actually use HTML – any other technology can be employed, as long as the information to be presented is linked into these files through the placeholder mechanism from the abstract web site structure. Some few remarks on the abstract web site structure and the linking of layout are contained in section 5, “Adding Contents, Linking Form” below.

Administration web pages. To actually perform the web site updates, the web site owner must be enabled to modify the contents of the WOLM. He does so by using a update assistance dialogue which is derived from the WOLM, enabling him to inspect and modify the model contents corresponding to the current business situation. With the help of various representation and interaction techniques (see section 6, “Administration Component”), the web site owner feeds update information into the WOLM, using standard Internet browser technology. The structure of this dialogue follows the object modifications as specified in the WOML, thus guiding the web site owner along the procedures and functions of his business. Based on the web site owner’s input the web site is updated by filling the abstract web site structure with values and by automatically executing the process linking contents with the HTML design files. We deal with the design of this dialogue, which is crucial for the approach, in section 6, “Administration Component”.

4  Web Object Life Cycle Model

4.1  Task Model as Basis

The task model as described during task analysis builds the basic input for defining the WOLM the web site management system is working on. When creating a task model, the tasks of the analyzed entity are identified. As mentioned before, we do not try to create a complete picture of the business, but restrict the task analysis and modeling phase to those tasks which are relevant to the web site update process.

As example, used throughout the paper, let us assume the web site owner to be a professor at a university and let the “business” be the organizational issues of the professor’s working group. A typical web site like this will, for example, contain the names of the people in the group, theses work that has been done, is currently under way, and is planned, scheduled talks, lectures, and meetings within the group, and so on. Tasks of the professor are for instance “assign a thesis to a student”, “define a new thesis topic”, or “assign a new student to the group”.

A task model does not just enumerate tasks, but shows their interrelations and dependencies as well.
When performing the task “assign a thesis to a student”, the professor also has to “assign a new student to the group” if the student has not formerly shown up in the group.

Additionally, a task model captures the affected objects and their properties relevant in the context of the tasks.

In the context of “thesis assignment” relevant objects are the thesis and the student. Both the professor and the students visiting the web site are interested in whether a thesis topic is already worked on, or whether it is still “on the market” to be assigned to an interested student.

Most approaches [12, 8, 7] use the hierarchical decomposition of tasks into subtasks as the primary modeling technique for refining a task. In contrast to this, we shift the focus to the objects of the business and how business task execution affects them. Thus, from the task model, we extract information about the objects, their states and how during task performance these states change. This creates a picture of business task execution in terms of object state modifications.

When performing the task “assign a thesis to a student”, the object thesis changes its state from “waiting for a candidate” to “under way”. At the same time, the student object changes its state from “looking for a thesis” to “working on a thesis”.

Overall, the task analysis and modeling phase prepares the ground for creation of the WOLM, which collects the properties of the relevant objects of the business and their modifications as a result of business task execution.

4.2 Web Objects

The term “web object life cycle model” reflects the fact that this model deals with the web representations – denoted as web objects – of real objects of the business. Web objects are abstractions of the „real thing”: they do not carry the complete information, static or dynamic, of the real object, but only the subset of the object’s properties necessary or suitable for display on the web.

In our example of the professor’s work group web site, relevant web objects could - among others - be

- the people co-operating in the group, including students, secretaries, and the professor himself,
- the past, present, and future theses written under the professor’s supervision,
- events taking place in the group, such as talks, lectures, or meetings.

Web objects can be anything that plays a role in the business and are chosen to be represented in the web. In reality, a student, for instance, would not be considered as an object – for the professor’s web site, however, a student is represented by a corresponding student web object. Deliberately, we do not speak of web objects in terms of web pages, text fields, let alone HTML files – instead we want to introduce the notion of an abstract web representation object capturing the behavior of web representations and their modifications, closely linked to things happening in reality.

The web object describing a master’s thesis, for instance, might consist of the title, a short descriptive text, a reference to the student (web object) working on the topic, an URL of the thesis working web page, and the date of the thesis defense.
The development of the „real things“ guides and directs the modification of the corresponding web objects. Events in the real world necessitate changes in the web object’s contents and representation on the web. Without any support as we propose here, the web site owner or some responsible administrator, must be on the alert to react on events happening in reality, and actively trigger or execute the corresponding web site updates. Especially in the case of continuous, but infrequent changes, these modifications can easily be forgotten, leading to web sites with out-of-date information.

In our example, we pick up the example of the current “theses situation” in the professor’s group. We want to show, for instance,

• thesis topics waiting for a student to pick it up,
• which student is currently working on which theme,
• when a thesis was turned in and the defense date is scheduled as an event.

Hence, the web site must be updated, when a student is assigned a thesis, a new topic is raised and promoted by the professor, when a thesis is turned down or finished by the student, when the defense is scheduled and held, and in similar situations.

As the examples show, to keep a web site up-to-date, the web site administrator needs to closely observe the reality, and be experienced enough to actively trigger corresponding modifications of the web pages. There is a permanent risk of things being forgotten or overlooked, leading to web sites with out-of-date information.

4.3 Web Object Attributes

Representing some important unit of information in the web necessitates the storage of information to be displayed in the web – hence, web objects have attributes with commonly known types, such as integer, boolean, or string. Important additional types are object references, i.e. links to other web objects. This type of attribute is needed for specifying object cooperation, as web object modifications frequently are linked to each other.

A thesis web object might be structured as containing the attributes

- String title,
- String description,
- Student candidate,
- URL workingPage,
- Date defenseDate,

where “String”, “Student”, “URL”, and “Date” are attribute types, followed by the attributes’ names.

Without going into detail, we allow attributes to have more than one value, i.e. to represent lists or sets of values, without explicitly defining a special data structure, such as array or list, for their organization. This list property is denoted by a star (*) following the attribute type name.

The professor’s web site is represented by a Chair web object, containing among others the list attribute

- Student* candidates,

denoting the set of all students, currently writing a thesis in the group.
It is an explicit and important concept for a web object attribute, whether it is defined or not. This corresponds to the reality of the web site update process, which can lead to increasing or decreasing the amount of information about some real object. Initially, all attributes are undefined, hence, they can not be represented as pieces of information in the web. During the life time of an object, attributes receive values, i.e. become defined, which is the pre-requisite to display them on web pages. Attributes can be modified, but they can also “loose” their value, i.e. become undefined again.

A new Thesis web object initially contains only undefined attributes. Once a topic is defined, the “title” and “description” attributes carry values, while the object reference “candidate” is still undefined. When a student takes the thesis as his topic, then the “candidate” attribute receives a reference to the corresponding student object as a value. If the student gives up, and returns the thesis, the reference becomes undefined again.

### 4.4 States

An essential concept of the model we propose is the explicit modeling of web object states. In reality things change, and we model this by introducing a finite set of explicit, named states for the objects. This well-known concept relates to the finite state machine known from computer science theory, or state charts, as developed by Harel [2] as employed, for instance, within UML [11]. The different states represent situations or configurations a web object can be in. They describe the changes a web object undergoes during its life time, and they have a clear semantic meaning to the web site owner within the business. Apart from the internal names used to identify the states, we add longer texts to describe states in the language of the business. These texts will be used during the simulation of the model. It is important to note that the states and their descriptions have a clear semantic meaning to the web site owner and specify the different relevant situations of a “real” object and its corresponding web object.

The state set of a Thesis web object contains the states “new”, “waitingForCandidate”, “underWay”, “turnedIn”, “defenseScheduled”, “finished”, and “done”. As textual description we have, e.g. text(“underWay”): “The thesis has been assigned to a student, who is currently working on the topic.” or text(“defenseScheduled”): “The student has finished the work on the thesis; now the defense date is scheduled for the near future.”

The objective of defining states is to specify that a web object might be represented differently in the web corresponding to state changes. Hence, to identify states within the model, we think about the web object in terms of its later representation, i.e. which different states of a web object do we want to exploit when displaying information about it in the web. The representation might change its level of detail and its visual appearance, according to the current state of the web object.

A Thesis web object which is “new” is still invisible in the web. If it is “waitingForCandidate” then its title and description might be listed as possible topic in the list of future theses, together with standard information on how to apply for it. If it is in the state “defenseScheduled”, the event might be listed in
In different states different constellations of attribute definition of the web object exist. During state changes, treated next, attribute values can be defined, modified, or deleted, such that state transitions will ultimately result in modifications of the web representations of the web objects. Exactly one of the states is marked as being the initial “start” state of the object: If a new object is created, it is initially in this state.

4.5 State Changes

The definition of a web object type is completed by defining the trigger and the effect of state changes. We assume that at any time every web object is in a well-defined state. For every state we define a set of applicable state transitions. The triggers of state transitions correspond to events happening in reality to the real „objects“. They are assigned textual descriptions, written in the language of the business describing the events in “real world” language. Later on, these textual descriptions of state transitions will be used to create the individualized web update dialogue mentioned above. The state transition itself models task performance, happening in the real world, and changing the state of a real object; the descriptive text asks about the condition reached once the task is completed. This reflects the situation that the execution of the web object model corresponds to a “simulated world” existing in parallel to reality, capturing changes having taken place there.

A Thesis web object contains e.g. the transition “finalizeThesis” from the state “underWay” to the state “turnedIn”, described in a text like

\text{text("finalizeThesis")}: “The student has finished working on the thesis, and has turned it in officially.”

It also contains the transition “scheduleDefense” from the state “turnedIn” to “defenseScheduled”, described as
text(“scheduleDefense”): “After examination of the thesis which was turned in, and its acceptance as a thesis, a defense date was scheduled.”

Within the paper, we depict the state transitions of a web object as state transition diagram, as shown on the previous page. The nodes represent the states, the arrows represent transitions. For the Thesis objects, the state transition diagram is shown above. The state “new” is the initial state, denoted by the solid circle line. This diagram reflects the complete life cycle of a thesis within the professor’s chair. All states and the state transitions have a well-defined meaning in terms of the web site owner’s business. The possible outcome of a thesis defense (success or failure) is modelled as well: After a failure the professor might decide to give the topic up (“giveUpTopic”), or try to find another student to work on it (“tryAgain”).

The effect of a state transition is specified by describing attribute modifications and the target state reached by the transition. An attribute modification can be the definition of a previously undefined attribute, the modification of a given attribute value, and the deletion of information of an attribute. Later on, once a state transition is triggered, we will use the attribute modification data to request the user to input data, to assign new or modified values to attributes.

The transition “scheduleDefense” from the state “turnedIn” to “defenseScheduled”, for instance, will contain the new definition of the “defenseDate” attribute; the transition “giveUpTopic” will delete “title” and “description”; and the transition “tryAgain” might modify the attributes “title” and “description”.

4.6 Object Cooperation

The concept, as developed so far, models the behaviour of single web objects in terms of state changes, and attribute modifications. It is obvious, however, that the life cycles of different objects influence each other. Formally, we express the link between objects with object reference attributes:

A Thesis object \( t \), for instance, is linked to a Student object \( s \), through the attribute candidate. When assigning a thesis to a student, \( t.\text{candidate} \) (i.e., the attribute candidate of the object \( t \)) has to be defined, to reference the student working on the thesis. It can be assumed, on the other hand, that the Student object \( s \) changes its state when the student takes over a thesis, and that there is an attribute named, say, \( \text{myThesis} \) of type Thesis within \( s \), which has to receive the value \( t \).

“Knowing” another object enables an object to trigger state transitions of the other object by “calling” the corresponding state transition. No other influence between objects is possible – hence, every single object keeps full control over its attributes, thus implementing strict data encapsulation. Hence, for the specification of object cooperation, a state transition not only executes modifications of attributes, but also “asks” other objects to perform state transitions.

Consider the model of a web object for a Student as shown on the next page. We only model the student’s relation to the chair with respect to taking over a thesis. The student after introducing himself at the chair, can be either still be looking
for a thesis, or working on one. While working, he can not leave, but if he owns no thesis topic from the chair, he might decide to leave the group completely. The Student object might contain the attributes

- String name,
- String address,
- int studentId,

The transition “introduceYourself” requests basic data as input, i.e. “name”, “address”, and “studentId”. The transition “receiveThesis”, asks to specify the value t of “myThesis”, which is a reference to a Thesis object. In addition, this transition requests that the transition t.assignStudent is performed. As an input parameter of this transition, a Student object is requested – the Student object assigns “itself” as parameter to the transition, i.e. it performs t.assignStudent(this). “this” is a reference of an object to itself.

Leading on, another case of object cooperation is a state transition leading to the creation of a new object, thus allowing the web object life cycle model to grow in number of instantiated objects during its lifetime. On the other hand, an object may reach a state with no further possibility of a state transition – a final state. If all attributes of the object are deleted by that time, it is no more represented in the web site, and it might disappear from the model completely.

5 Adding Contents, Linking Form

Abstract Web Site Structure. To represent the web site page and site structure, we follow an approach strictly separating form and content. The abstract web site structure on one side is a hierarchical pure information structure. It is a tree built up from node of the types Project, Site, Page, List, Field, Text, Graphics, and Link, denoting the corresponding elements of the future web pages. The concept supports mechanisms to incorporate repetitive structures, such as lists of structured elements (pages, tables, items in a text list, and so on). Technically, two corresponding tree structures are used: the project structure tree defines the structure of the information
trees used for a specific site, while the data tree refers to the actual data and is structured along the rules of the project tree.

When designing the abstract web site structure, the designer decides how to distribute information onto different pages and how to group information on single pages. Also, links are defined in terms of the definition of their target and from which other element (such as a text or a graphic) of the page they start. The abstract web site structure is linked to the WOLM by adding modification scripts of the structure to the state transitions of objects. When the web site owner works with the WOLM by adding, modifying, or deleting information, state transitions are triggered, which in turn perform the scripts mentioned, and hence the abstract web site structure is modified correspondingly. No information, however, is included in the abstract web site model about the graphical representation of the web pages.

**HTML Design Files.** The graphical design part is separated into the HTML design files, as mentioned above. These files are fully designed web pages, with contents, such as concrete texts, or graphics, substituted by placeholders, referring to the data tree of the web site. Whenever requested by the system, an automatic process combines the HTML designs with the actual data, as referred to in the placeholders, into the final web pages. The process we implemented is flexible enough to create variable sized tables and variable numbers of interlinked web pages making up the complete site. The design possibilities are not in any way limited by the process.

### 6 Administration Component

The site representing the owners business has to be updated according to changes resulting from business activities. This update process is modeled by state and attribute modifications of the web objects according to the behaviour description as defined in the WOLM. For this purpose the web site management system provides an administration component, which enables the web site owner to access the WOLM and to trigger state transitions. Hence, through these administration web pages the web site owner “executes” the model according to the web objects’ life cycle specifications.

#### 6.1 Executing the Web Object Life Cycle Model

As mentioned before, the WOLM describes the creation and modification of the web objects. In the following, we describe how such a model is interpreted in terms of its behaviour. Hence, we define the run time semantics of the model which are the basis for the finally resulting web update process.

The behaviour of a single web object is very simply defined, based on the concept of the finite automaton: The object starts to exist at some point of time in its initial state, then “waits” for transitions to be triggered. If this is the case, i.e. a state transition is triggered, for example, by a human user, the object performs the data operations on the attributes as specified in the transition, and then changes its state. Now the system is stable again, and the cycle re-iterates. The situation is more complex, however, if object cooperation is concerned.

277
Assume that the web object life cycle model is in a stable situation. All objects are in a well-defined state; all previous state transitions are done. Now, one of the possible state transitions is triggered externally. This state transition contains attribute modifications and, as we saw in the section on object cooperation, “calls”, i.e. triggers, of state transitions of other objects. There are three issues special to this situation:

**Input parameter values.** When state transitions are triggered by a human user, the necessary input data stemming from the “world” has to be provided by the user. When, on the other hand, a state transition is triggered by some other state transition, the input parameters may be provided both by the calling object and the user. In the model we specify calculation expressions for internally provided input parameter values, and denote those input values to be requested from the user with a special keyword.

**Buffered modifications.** When dealing with a state transition, all modifications to attributes are buffered. This means, that all modifications of attributes and state changes are computed and stored in a buffer, and not immediately assigned to the web objects. If a state transition is requested for some object, which is not available for the current state of that object, then the whole transaction is discarded, the buffer is deleted and the system stays in the stable state it was in before. Only after the computation process finishes without any contradiction, the new attribute and state values are assigned to their proper targets. As the web site owner will in fact work with the executable WOLM when entering data to update his web site, the administration component has to cope with such situations for the sake of the inexperienced user. The model must be carefully analyzed for detecting such potentially critical situations.

**Object dependencies.** A state transition of an object over a sequence of requests may result in the request of another state transition of another object. This is a frequent case, as in our example:

*Triggering the state transition “leaveGroup” in a Student object, may trigger a “formerMember” transition in an object FormerGroupMembers. Both transitions require date input from the owner: By executing the first one the user has to fill in the date when the student is leaving the group, while the second transition may require data about, say, the thesis result.*

In such cases the administration component has to provide forms by which the user can fill in the requested data. This can be realized using different strategies: All user input data of dependent transitions is collected in advance and requested by a single form. The owner can fill in the information and after submitting it, all transitions are executed in one step. Alternatively, as such a form can be very large and unstructured, required information could be grouped by collecting input parameters from the first transition, afterwards from the next sub-set of transitions and so on.

The run time semantics of the web object life cycle model as described above lead to a situation, where the different object types, included in a model, can cooperate dependent on each other, while still keeping full control over „their“ states and the modifications of „their“ attribute values. This modeling approach allows the designer and user to think and describe the behavior of an overly complex model of cooperation in a way which is very focused on the local behavior of the single object, but still allows the specification of sophisticated mutual influences between objects.
The only modifications to the WOLM invoked by the owner are state transitions of objects. At no point in time is he able to directly alter attributes values. Thus, as long as the model is consistent in itself, all data will remain consistent all the time.

6.2 Composition of Administration Web Pages

The user interface of the administration component is realized as a web site. Its web pages show the web objects, their attributes, current states, and the transitions. As the specified web sites can consist of a large number of objects and corresponding data, the administration web pages have to be carefully structured to provide a good overview to the user. We found different useful categories for representing possible transitions to the user. Each of these categories result in a list of transitions, but they are organized from different points of view taking into account different sorting criteria.

In the task related view, the tasks identified during the early task analysis are represented providing access to all objects and transitions related to those tasks.

Let us assume that the professor assigns a thesis $t$ to a student $s$. As his web site should reflect this change in the real world he has to modify the WOLM by trigerring the transitions related to the task “assign thesis to student”. In our example he has to invoke the state transition “receiveThesis” in the Student object $s$ or the “assignThesis” transition in the corresponding thesis object $t$.

This grouping of transitions per task is not generated from a formally defined task model. As described above, our approach uses the task model as a starting point for developing the WOLM. While constructing the administration web pages we use the information held by the task model, i.e. the textual descriptions, the object and transitions related to the context of a task, to define task groups. Each group holds all the objects and transitions necessary for modifying the WOLM according to the performed business task. This may also comprise transitions of neighbouring tasks.

A student may come up with an excellent idea of a thesis topic, which the professor readily accepts. For assigning this thesis to the student, he also has to trigger the transitions of the task “define a new thesis topic”. Therefore, this transition is also provided in the task related view of the task “assign thesis to student”.

The object type related view provides access to all objects of the same type. We introduced this view as not all necessary modifications occur in the context of one of the tasks identified during the task analysis phase. Thus, representing possible transitions within the pure task related views only would be too restrictive.

The professor may want to check all his unassigned thesis objects and to decide for each one whether he wants to give up the topic (i.e. trigger the transition “giveUpTopic”) or not. For this purpose the professor can invoke a view representing all thesis objects.

Depending on the existing objects per type such views may become very complex. To enhance their structure we defined additional sorting criteria which help the user’s orientation.

By applying the criteria of active transitions, all current states are inspected and the outgoing transitions are collected. This results in an overview of all transitions which can be performed in a current situation, i.e. at the time when the owner invoked the administration web pages.
In a similar way, the criteria of reachable states lists all states that are reachable based on the current state information. In addition, for each state the transition leading to that state can be shown.

The time stamp criteria is based on the definition time stamps in conjunction with a time stamp scheduling. Typically, the web site owner has rich experience about the time behaviour of the business. This information is collected and included in the WOLM for reminding a web site owner to keep the web site up to date. The administration component utilizes these time stamps to order information modification requests, resulting in a timely ranking of the transitions.

Each of these criteria can be applied independently from an object view, which results in a view showing the transitions of all objects filtered by the selected criteria. For example, by sorting the transitions using the time stamp criteria, all possible transitions are listed according to their importance defined by time stamps. Hence, the user finds a list of all transitions to be triggered for keeping the web site up to date. In addition, all transition whose time stamps are out of date, are listed. This listing indicates that the final web pages might not be up to date and the transitions should be triggered immediately.

All views and criteria have to be provided in arbitrary combinations. By providing this much flexibility, the user should be enabled to do the necessary update operations with ease.

7 Conclusions

Currently, the web object life cycle model is defined, and a first group of tools is or will soon be working. This includes a graphical editor for web object life cycles, a simulator for the WOLM, and a Java implementation of the WOLM run time semantics. The abstract web site structure has been formally defined (as XML structure) and tools to combine attribute values with the project structure and with HTML design files are operating. The work for the administration web pages part is under way.

The approach is well-suited, as reported in the paper, for the inexperienced web site owner of the “small” web site. We are well aware that complex web sites are reasonably based on databases and transaction concepts instead of single objects and their life cycle. But even “big” web sites with a sophisticated web application underneath can profit from the modeling and specification approach used here, by combining web object life cycles and database specifications. Also, the approach is useful for web site owners even if they are able to work with available tools to create and maintain web sites, as it helps them to keep the web site up to date. We did not treat this to a large extent in the paper, but the time stamp mechanism is extremely useful in that respect, as it can also be employed for sending reminder emails or faxes to the web site owner, thus actively triggering him to update the site information.
References

Dynamic and adaptive e-commerce architecture based on agent technologies

Ana B. Gil*, Francisco García*, Zahia Guessoum**

* University of Salamanca, Dept. Informática y Automática – Facultad de Ciencias, Plaza de la Merced s/n, 37008, Spain
{abg, fgarcia}@usal.es

**LIP6, UPMC 4 Place Jussieu, Case 169, 75252 Paris Cedex 05
Zahia.Guessoum@lip6.fr

Abstract. The emergence and growing popularity of web shopping has made it a necessity to define flexible and adaptive e-commerce systems and architectures. The on-line stores are accessed by customers with different backgrounds, expertise and preferences, therefore these differences should be taken into account to improve the usability of systems. Our work describes the e-CoUSAL architecture, an e-commerce system that is suitable for small and medium enterprises, where several enterprises share a common e-commerce site; the site automatically adapts its interface to offer end-users the products organised in electronic catalogues (e-catalogues). The presented architecture is based on an agent-oriented technology. We also describe the actual state of our system and the implementation of an adaptive agent in the server side of the architecture.

Keywords. Adaptive agent; E-commerce; Adaptivity in E-commerce; Adaptive interface; E-catalogue; XML.

1. Introduction

With the expansion of the Internet, several kinds of systems with worldwide scope have appeared recently. Perhaps the major application area of Internet development is precisely e-commerce. Since the on-line store is accessed by heterogeneous users, it should satisfy different needs and preferences in the selection of goods. Nowadays web stores are hypermedia systems, they should fit the user expectations, in this way they should meet the user’s needs in what concerns the interaction style and their technological constraints in order to facilitate their access to the stores and their products. The personalization of the front-end of on-line stores has become a critical issue of this kind of system [1].

One particular e-commerce segment that is especially popular is the one based on product catalogues, also known as electronic catalogues or e-catalogues. An e-catalogue can be defined as the electronic presentation of information about the products and/or services of an organisation [15]. E-catalogues are widely accepted in web-based business. Also, while other applications can provide similar services, e-catalogues provide a range and effectiveness of service that exceed the capability of
any competing application, such as physical or CD catalogues. The interactive possibilities of e-catalogues eliminate physical storage and make continuous updating possible and efficient [3].

However, although Internet is rapidly becoming the business platform through which many enterprises deliver services to other enterprises and end-clients [5], the entry of an enterprise into the e-commerce world involves strategic decisions that are not cost and risk free. These costs and risks are often an obstacle for the integration of an organisation into the virtual commerce community.

The impediments are more important in small organisations or businesses, the so-called Small and Medium Enterprises (SME), in which the amount of investment in technology solutions cannot be very large.

Our group is interested in the definition of e-commerce models that allow the entry of the SMEs into the virtual commerce bandwagon. We have implemented an e-commerce system of this kind. It is based on a more generic and ambitious architecture for product catalogue-based e-commerce architecture, which we call e-CoUSAL [8, 9].

This architecture is based on two main components: a visual catalogue-designer tool and an e-commerce web server. The first one is the authoring tool used by the enterprise to generate and manage the e-catalogue that presents its business in the e-commerce site. The e-catalogue will be published in the e-commerce web server automatically. Then, the e-commerce site needs to adapt itself to show the correct interfaces and the actual information to its end-users through its e-commerce services.

The e-commerce site supports a catalogue-based policy; this means that this kind of system diverges quite far from classic on-line information systems. While a hyperspace of information items still constitutes a major part of these systems, browsing this hyperspace is not a major activity, but only a by-product of the major activity (such as shopping for goods). In fact, the better these systems work, the less browsing should be required. Adaptive characteristics are particularly interesting here [4].

In this work the overall e-CoUSAL is presented from an agent-based perspective, and we also describe the first prototype of this e-commerce system; it implements an adaptive agent in the server side of the architecture. Thus, the paper is organised as follows: Section 2 explains the proposed agent-based e-commerce architecture. Sections 3 and 4 examine the components of this architecture that are actually implemented. Finally, Section 5 closes the paper, presenting our conclusions and some ideas on further work needed.

2. An agent-based e-commerce architecture

In this section we introduce the different agents that comprise the proposed architecture. First of all, we will describe the components of this architecture.
2.1. Components of the e-CoUSAL e-commerce architecture

In a schematic way, we show in Figure 1 the major components of the proposed architecture, where exits three main actors: the SME, the e-commerce site and finally the client or end-user all connected through their roles. As we stated above, the main commercial policy is based on e-catalogue shopping, supported by two main components: the e-commerce web server and the visual catalogue-designer tool.

There is a central element, the e-commerce server, which interconnects the different parts involved in a typical commerce environment, but rather more dynamically than traditional business forms. The e-commerce site is just a technological intermediary that supports the task for the SME in relationship with their business, now in a virtual level.

The SME becomes the main actor of its own virtual business approach. It is responsible for the inclusion and management of its own contents in the e-commerce site, which allows the enterprise to enter into the e-commerce environment. The use of a specialised software tool to design the e-catalogue permits the SME to be an active element within the commercial process.

![Fig. 1. Components of the e-CoUSAL e-commerce architecture](image)

The designer tool allows the SME to build the definition, publication and update of an e-catalogue, and also the configuration of a web server architecture that permits end-users to have access to this e-catalogue into all the e-market there hosted. However, for an efficient communication between the tool and the server, and for an automatic adaptation of the server, the e-commerce server has to arrange the restrictions for the e-catalogues to which the software tool is conformed.
The data interchange between the SME and the e-commerce site, and the later automatic publication for final client’s accessibility, is based on an XML storage format that defines the structure of the e-catalogue ontology.

The relationships between the SME and the e-commerce server, through the catalogue-designer tool and also through the server management services, represent a B2B dimension in this e-commerce model.

Moreover, the server has to provide the end-users with the commercial services needed for browsing the e-catalogue and for purchasing; this includes searching for a product in any published catalogue, shopping cart management, selling certificates, navigation help, and so on, and all these through a uniform, intuitive and adaptive interface. These functionalities in the server site define a B2C dimension of this model.

Thus, the overall architectural model defined above presents a B2B/B2C hybrid e-commerce model [7].

2.2. Agential view of the defined architecture

The e-commerce architecture introduced in Subsection 2.1 is now more precisely defined in terms of a set of cooperative agents or multi-agent system.

In a multi-agent e-commerce environment it is necessary to organise agents into different categories, depending on their functionality and competencies. Several different forms of agents for e-business systems are distinguished in [12]:

- **Application agents**: Each agent specialises in a single area of expertise; it provides access to the available information and knowledge sources in that domain and works cooperatively with other agents.
- **Personal agents**: Also called interface agents, they work directly with end-users to help supporting the presentation, organisation, and management of user profile, requests, and information collection.
- **General business activity agents**: The basic commercial services need some agent technology support.
- **Information brokering agents**: This kind of agents provides facilities such as locating information on web sources or other agents.
- **Negotiation and contracting agents**: They are in charge of the necessary negotiation processes expressed in terms of business transactions.
- **System-level support agents**: They represent the top of the distributed middleware system, which provides objects or services in a transparent way.
- **Planning and scheduling agents**: For collaborative actions a plan that specifies the future activities and interactions for each agent is needed. Typically, in e-commerce applications an agent may act as the group planner for a cluster of agents.
- **Interoperate agents**: They support the communications and interoperation with legacy systems.
- **Business transaction agents**: They collect, manage, analyse and interpret the commercial data to make more intelligent and effective transaction-related decisions.
- **Security agents**: They provide the security services required for conducting e-business.

According to this classification, we have identified the agents and relationships needed to support our e-commerce architecture proposal. These agents are shown in Figure 2. We present a first attempt of the agent-based architecture in coarse granularity and in high abstraction levels, because we are defining the architectural layer of the system, relating the agents to the main components presented in Figure 1 (in [1], for instance, another agent-based architecture for web stores is introduced, but with a thinner granularity view).

In Figure 2 ovals represent agents and arrows represent communication between them or between external entities, as end-users. Bold shapes or lines mean that the agent is actually completely or partially implemented; dotted shapes or lines mean future development in our architecture. Also, the agents are related to the components presented in Figure 1, including the agents inside them.

![Fig. 2. Agents in e-CoUSAL architecture](image-url)

The **application agent** is the visual designer tool used by a SME to create e-catalogues. We shall describe it in a more detailed way in Section 3.

The **broker agent** is in charge of receiving the e-catalogues, expressed in XML format, validating them, and storing information in the proper internal database. The tool sends these catalogues through the **system-level agent**.

The **negotiation agent** is still in its first stages of development. This kind of agent will receive the business components of each SME in the same way that the broker...
agent receives the e-catalogues. This property will not only allow SMEs to be
presented in the e-commerce site; it will also let them personalize their business
policies (discount, payments and so on) through these components.

General business activity agents are a set of agents that manage the typical e-
commerce services of the site: shopping-cart management, selling certificates and so
on.

The authentication agent is a security agent type that is in charge of identifying the
end-user, which makes it possible to adapt its interfaces to the user’s shopping tastes.
The personal agent is responsible for customising the interaction with the user and
the presentation and navigation of e-catalogues. It is implemented as an adaptive
agent (as explained in Section 4). This agent dialogs with the web server module to
interact with the user and dynamically generates the hypertext pages that represent the
e-catalogues. A further work in the customisation area is the implementation of a
personalization agent that will be in charge of acquiring and maintaining the end-
users’ profiles, which represent the system’s hypotheses on the users’ needs and can
be used to tailor the layout of the hypermedia pages to each specific end-user. So this
agent should learn from the visitors of the e-commerce site to infer the end-users’
interests. From the visitor’s viewpoint, the agent should help the user make sure that
useful information is not overlooked. On the other hand, a personalized profile should
be used to make the proper recommendations to the users [14], thus reducing their
need to browse through pages.

Finally, a planning agent is due to the presence of heterogeneous problems to be
faced and the fact that many tasks could be carried out at the same time suggest the
design of the multi-agent architecture outlined above. This is a planner agent that
controls the main data flows in the system, initialises and updates the user models
during the interaction. In a few words, its responsibilities includes manage the e-
catalogues database, accepting the orders of the e-commerce service agents, and in
cooperation with the personalization agent giving to the personal agent the proper
data to generate customised information pages handling the products and user
databases.

We shall now describe the two main components of this architecture in a more
detailed way; we mean, of course, the designer tool and the e-commerce site.

3. The visual catalogue-designer tool

The axis of interaction of this business model is the catalogue-designer tool that
allows both e-catalogue creation and publication activities. This authoring tool is
distributed to every SME that belongs to the e-commerce platform. The tool has a
three-level hypertext allocated in three template definition views, one Product
template view describing product categories, the second one Product definition view
represents the concrete characteristics of each one of the products and the third one
the Catalogue manager View groups all the products in e-catalogues for their later
publication in the server, and also their self-maintenance by the SME.

The intermediary has to facilitate to the SMEs the contents creation, the update and
the management of their products, and also a universal means for the supported e-
catalogues. Then the majority of the transactions between the SMEs and the intermediary are made through this tool.

The SME develops the contents of the e-catalogue with the authoring tool, saving the catalogues in an XML-format file; The designer of the e-catalogue will be the person in charge of e-catalogue maintenance, which is done on local computers, and the resulting catalogue is subsequently sent to the intermediary server. The designer and the enterprise need not be worried about e-catalogue integration, because this is an automatic task on the intermediary side.

Using a tool such as a SME/intermediary interface facilitates the e-catalogue creation and publication processes, because the designer of them does not have to be a computer expert; he/she only needs to be familiar with basic office automation concepts, because all the knowledge concerning the saving format of the catalogues and the communication for catalogue publication aspects are encapsulated inside the tool functionality.

One of the major constraints in the design of this tool was to succeed in using it for a wide number of users. Related to this constraint we work in two directions: the portability of the tool, we use Java as implementation language, and its language independence, the tool is designed to allow multiple-language support.

The working process with the catalogue-designer tool is structured around the work-view concept. The idea is that the information could be shown by different angles, one for each defined work-view. This perspective is justified by two main reasons: first, we want to reduce the amount of information that is shown to the user of the tool, thus avoiding an unnecessary information overload. On the other hand, we want the users of the tool to centre their efforts on the specific task in the overall process of the e-catalogue design.

The main work-views defined in the presented visual design tool are: the template definition view, the product view, and the catalogue organisation view.

These three work-views represent the functional process used to create or design an e-catalogue. They are chosen after a deeper task analysis of the e-catalogue creation process, and they can be found in the user interface of the tool in separate tabs, thus making it easier to switch.

The basis of this work philosophy lies in the Model View Controller (MVC) pattern [9]. In this case the model part is formed by the complete catalogue information, the view part is represented by the work-views introduced above, and, finally, the controller part is the one responsible for maintaining the work-views related one to any other in a consistent and lasting way; for this reason a fourth view is introduced, the repository view.

The repository view is not a working view; its purpose is to group the different elements managed by the tool, maintaining the cross-references among them to guarantee the consistency.

Now, we shall proceed to give a brief description of the work-views defined in the tool.
3.1. Template definition view

It is something like a data type definition mechanism that can be applied to describe the fields of the products. Inside this view two different elements are considered: the data template and the product template. The data template could be defined as the mechanism that allows us: first, to define the data format of one field for product description, and second, to help to the user to define the contents in a more effective way. Furthermore, a product template serves as a model for the later definition of the concrete products of the e-catalogues. A product template defines a set of fields that describes those products that will be defined using the product template, also this kind of template can be organised in a hierarchical way, inheriting field descriptions from its ascendants. The objective pursued is to group common characteristics to a set of products, reducing the efforts to define them.

An example of a product template, called COMPUTERS, is presented in Figure 3. At the left side of the user interface the template organiser is located. Using it, the user can work with product templates or data templates. The product template organiser presents the template hierarchy, which is an important notion for knowing the origin of the inherited fields of the template. In the example presented, the COMPUTERS template has several inherited fields from BASIC TEMPLATE (NAME, TRADE MARK…), each one with its own characteristics (Name, Data type, Default value, Mandatory and Description). But the product template can also add its own fields (CPU, MAIN_MEMORY…) to specialise the inherited template.

![Fig. 3. Product template example](image-url)
3.2. Product view

All the functionality for concrete products is gathered in this work-view. While the templates make the work easier, the products are the conceptual definition of each element that compounds the e-catalogue.

This work-view is like a product-description repository that makes it possible to create new products, maintain existing ones, and so on. The products could be presented in this view in different ways: classified by product categories, classified by product templates or ordered by name.

3.3. Catalogue organisation view

This work-view presents the grouping of the products in e-catalogues for their later publication in the server, and also their maintenance.

The functionality associated with this view allows the inclusion of personalised security constraints to control what persons or what entities have access to the e-catalogue.

When an e-catalogue is defined there is a separation between its conceptual definition and its visualization in the intermediary. This characteristic directs the e-catalogue composition process as can be seen in figures 4 and 5.

Fig. 4. Catalogue layout
In this work view the e-catalogue designer always knows how many catalogues are managed by the tool, their properties, and which of them are published in the intermediary. This is controlled at the left side of the user interface, in the “List of catalogues” area (Figure 4).

An e-catalogue is organised by sections and subsections. Each section has sheets (Figure 4) where the products are located in a visual way (Figure 5).

Fig. 5. Catalogue sheet definition

4. Personalizing capabilities in e-commerce server site

The e-commerce site offers its clients efficient access and shopping management for the different products that are published in the server. From an end-user perspective, the e-commerce presents the specialised supermarket metaphor, where a client can find several related products from different suppliers.

4.1. Adaptive Agent Model

The e-services and the information about the products are supported for the different types of agents presented under the e-commerce site component in Figure 2.
For an end-user the e-commerce site should be like another commercial site in Internet (it should present the same facilities and an easy and familiar interface), but the variety of the sources of the products should be transparent for the user. For this reason the customizing capabilities of the site are very important and they are actually supported by the personal agent.

Concretely, the implementation of this personal agent is based on an adaptive agent-pattern (Fig. 6), derived from the adaptive agent model proposed by Guessoum in [11]. This model proposes a meta-level layer that gives each agent the ability to take appropriate decisions about control or adapt the specific attributes of the system (interface, behaviour, navigation, contents...) over time to new circumstances.

The meta-level relies on data about the agent itself, its environment, and the decision system used by the work level and the way to modify it too.

We have adapted this general adaptive agent model to the characteristics of the e-CoUSAL e-commerce architecture, as we present in Figure 6. The user interacts with the e-commerce server that presents the customized pages to offer relevant information to the client.

![Fig. 6. E-commerce adaptive agent model](image)

The hypermedia pages that represent the e-catalogues are generated on the fly, getting the contents from the e-catalogues that were sent by the SMEs, which is the knowledge layer. However, to take out the information from the e-catalogues and in order to insert the data into the server database, there must be an ontology or meta-knowledge for e-catalogue definition that is shared between the server and the catalogue-designer tool.

4.2. Adaptive goals approach

Two main objectives are considered to develop this model: flexibility in the business environment and also the business property should be able to maintain its electronic business in self-sufficient way.

As we can see in Figure 1, the central element is the e-commerce server, which interconnects the different parts involved in the commercial environment. The SME, is responsible for the inclusion, across a specialised software tool, of the e-catalogue
that introduces the offers into the dynamically generated web pages in the server, where a client could find and buy the commodity.

At this time there is a prototype of this architecture on working, but in order to obtain the first objective *flexibility in the business*, we want to develop e-CoUSAL with an adaptive component.

The adaptive agent model, explained in previous section, is our ground basis. E-CoUSAL is being implemented through a multi-agent architecture, where specialised agents have to be designed to carry on the activities of the front-end of the web store. For this purpose, we count with the multiagent platform DIMA tool [10], founded on open and modular agent architecture and extended to build adaptive agents.

The *requirements* for the adaptation system architecture in order to approach work can occur at three levels: presentation, information and navigation. These three faces are combined and connected to conform the proposal architecture [13].

The user in order to the level of expertise and interest, needs an agent that assist him to navigate the web site (*Navigation*). The main goal is to endow *business-object* to the architecture, therefore *negotiation processes* reinforced with adaptive user interfaces (*Presentation*) for making offers in way of discount stockbrokers (*Information*) to the final client will be another task.

Therefore it is necessary monitor the interactions on the client-side taking into account the system where customer is working on (kind of medium, downloading time, content attributes). Also agents analysing web logs, determining similarity between documents and searching for making recommendations and learning about users. At this point there are twofold ways of learning, (i) about a concrete user, and (ii) from patterns of use. For modelling this information and combined all the users aspects, dynamic user modelling techniques are necessary.

The adaptive model we propose for e-commerce (see figure 6), interacts with the final client due to negotiation techniques helped with the e-commerce site client interface adapting in according to the inputs. The user is suggested with different products by the architecture on the basis of the users’ selection of items.

So the structure allows agents to register the users interests and personalized pages to offer relevant information to the client. Negotiation will locate the mutually preferred deals. User’s preferences are expressed to the agent that performs negotiation. On this way a flexible business adapts strategy and the site.

5. Conclusions and further work

The work reported in this paper is part of our ongoing research effort to develop an adaptive Web-Store Site and increase its degree of automation in all the properties around it, forming an autonomous architecture taking into account the SME-customer and the client-customer.

Adaptable systems are usually defined as these systems that allow the user to modify some of their parameters and thereby adapt their behaviour accordingly. If the system adapts to the user automatically based on monitoring the users’ interaction during runtime it is called adaptive [6, 13]. According to this definition, in this paper we have introduced an adaptive system for e-commerce proposal, a system that is
especially suitable for small and medium enterprises, which are organisations that have difficulties to entry in the virtual commerce scene.

To be precise, the paper defines an agent-based architecture for the e-commerce system, and two main components are identified: a visual catalogue-designer tool to generate e-catalogues and an e-commerce server site that stores the generated catalogues. These components comprise several different kinds of agents that support the commercial policy of the system.

Various types of adaptation could appear in a system. Actually the server site implements an adaptive agent that performs the interaction with the user, adapting the content, the presentation and the navigation properties of the dynamically generated information-pages from the e-catalogues stored in the server.

Further work is needed to implement the whole architecture, in which there are two more adaptive agents: the negotiation agent and the customisation one. In order to implement these agents we can use the same pattern applied to develop the personal agent actually used in the functional prototype of the architecture, which is presented in Figure 6.

The definition and implementation of these two agents are very important to achieve a flexible and adaptive e-commerce system, because we have now an adaptive system in the interface area (including the language of the contents and the available products organised in e-catalogues), but with the new agents we will have customisation profiles that will allow a better system for the end-user, and, on the other side, the enterprises could then define their virtual business policies through business components.

References


294
Using declarative constraints to specify the data model of a multi-user application

Boris van Schooten

Parlevink group,  
Faculty of Computer Science  
University of Twente,  
Enschede.  
P.O. Box 217, 7500 AE, Netherlands  
schooten@cs.utwente.nl  
http://wwwhome.cs.utwente.nl/~schooten/vekt/

Abstract. Complex applications such as multi-user applications may fruitfully be viewed as being composed of a number of independent agents which access and modify a shared data structure. We will view this shared data so as to include the domain data being viewed and edited as well as the entire user interface state.
Such a shared data concept may be effectively modelled using database techniques. One of the aspects of such modelling is the specification of requirements on the structure and evolution of this shared data. This may enable a desirable level of abstraction from the implementation of the user interface. Such a specification may be used to check the correctness of a given agent or ensemble of agents working on this shared data. To enable such checking, a formal coupling between the requirements and the executable system is desirable.
This paper proposes a constraint specification language, providing formal versions of traditional UML-type structure diagrams and statecharts, in combination with arbitrary constraints, which are similar to Object Constraint Language (OCL) specifications. These are coupled to applications written in a multi-user toolkit by means of code assertions. The usefulness of this specification technique is assessed by means of specification of a small multi-user application.

1 Introduction

In this paper, we will address system design issues of complex interactive systems, such as multi-user systems. We will view an interactive application as consisting of a number of separate (that is, concurrent, or at least, independent) agents that access and modify shared data. With shared data we mean both domain data (the data being viewed and edited) and user interface data (presence of windows, positions of scrollbars, item selections, etc.).
Such a concept of shared data is central to many user interface modelling techniques, though emphasis is usually laid on domain data. One of the goals of architectural styles [4], for example, is to identify and compartmentalise this
data, and restrict and clarify its access by means of architectural constructs and interface definitions. For example, in the MVC architectural style [9], one splits the shared data into a number of Model objects, and specifies the interface of each Model object separately. In the PAC style [9], an additional architectural constraint is prescribed that limits access to compartments by means of an encapsulation hierarchy.

Such models assume that the data can be fruitfully separated into a number of relatively isolated compartments. However, in many practical cases, there are complex queries and operations that need to be done on the data. These go beyond that of single compartment that is only accessed by a limited number of agents. For example, in systems providing inter-user awareness such as *What You See Is What I See* (WYSIWIS) [16], the user interface data of one user (such as scrollbar and mouse cursor positions) is concurrently accessed by different users in different ways, in order to enable the users to see each other’s activities. Another example are multimodal systems and personal assistant agents, which typically operate by tracking the activities of the user in different parts of the user interface.

In some multi-user toolkits we already find that the different user applications interact by accessing a single global shared data structure [13]. Here, we go a step further, and use a database approach for the modeling of all shared data [17]. An important aspect of specifying a system based on such a shared data model is describing global properties of this data. In this paper, we will discuss how formal versions of structure diagrams, finite-state machines, and other types of constraints may be combined to formally specify global properties of an application’s shared data in a declarative manner. We propose a new language for describing these models and show how an application can be specified using this manner of specification.

The paper will be structured as follows. First, we will discuss existing efforts on constraint specifications, and their relation with this work. Then, we will describe the constraint language itself, and how it is coupled with a working application. The language will then be assessed by means of an example specification, and its potential will be discussed.

2 Related work

Many formal languages exist that are suitable for defining constraints on data, though the means they provide for coupling to executable systems vary. A proper coupling with the executable system would however benefit the practical usefulness of such a language. A recent attempt at standardisation and stimulation of wide acceptance of such constraint specifications is OMG’s *Object Constraint Language* (OCL) proposal [1]. OCL is a practical language that is specially designed to be easily executable. Basically, OCL provides straightforward correctness assertions in the form of truth expressions over sets and other collections, using quantifiers. These collections are defined by means of ‘database’ queries,
with the collections being the queries' results. This combination of assertions and queries provides a means to specify structural constraints.

OCL's query facilities basically consist of navigational expressions, which enable navigation through object structures by following references. Note that this bypasses the encapsulation principle as prescribed by object-oriented design. In this regard, OCL's manner of specification is database-oriented rather than object-oriented. An OCL-like language could fruitfully be used to describe a system based on database concepts.

Some compilers exist that couple OCL to executable code, with the most notable one being the Dresden compiler [10]. It enables OCL constraints to be added to Java classes using concise annotations which generate extra code when compiled.

Another language that combines structure-based specifications with assertions is the Alloy language [2]. Though Alloy enables model checking, it provides no easy coupling with a full executable system.

Various efforts have been made to employ traditional abstract models such as structure diagrams and finite-state models for interactive software. While such (typically diagrammatic) specifications are not necessarily formal, they may for example be formalised by translation into an OCL-type language.

Structure diagrams, such as entity-relationship diagrams (ERDs) and UML's class diagrams [3,6] are well-used. Many (CASE) tools exist that couple such specifications to executable code, for example by means of generation of code skeletons [11]. Usually though, these specifications are primarily used to describe the domain, and not the user interface. However, informal usability methods such as ERMIA [3] have shown that they may prove quite useful for describing usability aspects of user interfaces as well.

Many models exist that specify the behaviour of a user interface using some form of finite-state machines. Finite-state machines and their many variants are generally intuitive and easily executable, may be coupled to sequence specifications, such as usage scenarios [18], and may carry all the way into a full system [15,8]. However, they are limited in their expressiveness because they can only practically describe a very limited number of states, requiring a large (and heuristic) abstraction step to be made with respect to the executable system. In the well-known statechart model [7], transitions are specified by means of truth expressions, enabling such an abstraction step. UML 1.4 [6] specifies that a statechart corresponds to the behaviour of an object in a structure diagram. Various development environments exist that combine state machines with structure diagrams in a similar manner [11,12].

There are much fewer models that integrate state machines, structure diagrams, assertions, and executable code, in such a manner that the correspondence between them is easy to understand and the formal coupling between them is ensured. In UML, there are plans to formally integrate structure diagrams and statecharts with OCL. The Teallach environment [5] provides both diagram notations, and enables OCL expressions to be used in some places. Still, there is little experience with using such combined specifications in interactive systems.
3 The constraint language

In this section, we describe a constraint language that enables constraints on shared data to be specified. The language enables OCL-type assertions, and has special notations for description of entity-relationship constraints and state machines. A code module may be generated from a specification, which may be added to a running system to provide diagnostic information, such as state transitions and constraint violations. The language provides a visual representation, with a comprehensive mapping of statements in the language to structure and state diagrams. Currently, there is no graphical tool for editing specifications, and diagrams are instead generated from the textual specification using a graph layout tool.

It is convenient to use a programming language or toolkit that is easily mappable to data structure constraint specifications. For this purpose, we use the VETk system [17], which models all shared data (which includes user interface state, even including mouse clicks and movements) using a special low-overhead relational database. Since everything is modelled as database accesses, the mapping between the constraint specifications and the executable code is quite close. Since this system already provides database queries as a basic construct, we get query facilities for free. Additionally, it provides a publish-subscribe model, which makes the tracking of changes made to the database easy.

Mapping of basic structure diagrams is relatively straightforward: one just defines a mapping between entities, relations, and attributes as defined in the constraint language with those in the database. This mapping is used to check relational multiplicities.

Mapping of state machines is more problematic however. The necessary abstraction step between the concrete state of the system and the abstract state machine should be formal, yet it should allow proper flexibility. Following Harel’s statechart approach [7], we specify transitions by an event, a condition on the state of the system, or a combination of both. Events and conditions may be specified as arbitrary OCL-type truth expressions.

Unlike OCL, our language separates the definition of the sets (by means of queries), and the constraints on them. First, we will explain how sets are defined, then we will describe the constraint language.

3.1 Database queries

A set is defined by means of a query. The contents of a set at a specific moment in time is the result of this query at that specific moment. Queries are written in Prolog style. Database tuples (i.e. entities and relationships) are specified in the form:

< typename >  ‘ (’  List0f[< field> ] ‘ ) ’

Fields (which represent attributes or references) may be literal values, such as integers, strings, or database keys. In queries, they may also be variables: the
tuple then specifies a search pattern in which the variable is a free variable. In our constraint language, queries are of the form:

```
' set ' <setname> '(' ListOf[ <variable> ] ')' ' : ' SequenceOf[ <tuple> ]
```

The right hand side (the clause after the colon) specifies the query pattern. The pattern's query result is a set of bindings of the free variables found in the tuples. For each binding, those variables are taken that are listed in the left hand side. This results in a set of tuples: the query result. Here is an example taken from the card game specified below:

```scala
set userelected("player","card") :
   selected("player","card") participant("player,self,\_)
```

The set that we specify here is a set of 2-tuples called `userelected`. This stands for the cards that are selected by each of the players (indicated by the relation `selected("player","card")`) which participate in a specific game (indicated by the relation `participant("player,self,\_")`). Note that the identifiers which start with a tilde `~` symbol indicate variables. The identifier `self` indicates a constant. In fact, `self` is a special constant that stands for the entity that the set is defined in, in this case the entity `card`. Finally, note the underscore `_` symbol, which is a wildcard that denotes that the value at that position does not matter.

It is these query specifications that are used to provide query facilities to the OCL-type expressions found in our language. They are in some ways more powerful than OCL navigational expressions, because they enable Prolog-style pattern matching. If we do not consider efficiency and ease of implementation, there is no reason why even more powerful query facilities might not be added to any OCL-type language.

### 3.2 The language

Now, we will continue with the specification language. The structure specifications are basically entity-relationship models. One may define entities and relationships in the following way:

```
[ 'entity' | 'relation' ] '('
    ListOf[ <attribute> | <entity> ':' <multiplicity> ]
')', '{',
   <values and events>
   <sets>
   <statemachines>
   <assertions>
')' 
```

An entity or relation may have attributes, and may have references to any number of entities, in which case the multiplicity of each reference is specified. Note that entities too may directly refer to other entities. While this practice is
often considered too implementation-oriented, at least the designer does have the
ability to specify this if desired. An entity or relation has a body, which specifies
attributes that are optional (called values) and operations (called events).

Other constraints may also be specified as properties of the entity or relation.
These are general assertions and state machine specifications.

The state machines that may be specified are basic finite-state machines,
with the difference that a transition may be specified by two arbitrary truth
expressions: one specifies an event, and the other a state condition. A transition
occurs when the state condition is true and the event occurs (i.e., the condition
specified by its truth expression goes from false to true). Either event or condition
may be left empty. A state machine is specified by:

\[ 'statem 4 \{ \text{SequenceOf[} \]
\[ \text{<sourcestate> \text{\rightarrow} \text{<destinationstate> \}
\text{\text{'event \text{'} \text{\text{'condition \text{'}} \}
\text{\text{\}} }\]

\] assertions may be specified inside an entity’s or relation’s body, or as separate
statements. Assertions are of the format:

\[ 'assert' \text{OptionalListOf[} \text{<state> \}
\text{\text{'assertion \text{'}} \]

In case we specify one or more states, the assertion only has to hold when
the entity in which the assertion is stated is in one of those specific states.

The events, conditions, and assertions are OCL-like truth expressions. The
expressions have the following form:

\[ 'exists' \text{\text{'<setexpression> \text{'}} \]
\[ 'forall' \text{\text{'<setexpression> \text{'}} \]
\[ <setexpression> \]
\[ 'size' \text{\text{'<setname> \text{'}} \text{<comparison_op> <value> \]
\[ <expr1> \text{\text{'\text{'and' \text{'or' \text{' <expr2> \]
\[ 'not' <expr> \]

With the \text{exists} and \text{forall} quantifiers, the specified setexpression is evaluated
for each combination of elements in each of the sets mentioned in the expression.
In case of \text{exists}, the result is true as soon as one of the expressions results in true.
In case of \text{forall}, the result is true if all of the expressions result in true. Specifying
just a setexpression is equivalent to using an \text{exists} quantifier. The \text{size} operator
counts the number of elements in a set. Finally, we may combine any of the
above expressions using the Boolean operators \text{and}, \text{or}, and \text{not}. Setexpressions
are Boolean expressions in which individual fields of previously-declared sets
may be addressed using the notation \text{setname.fieldname}.

Here are some examples of truth expressions:
not userselected
  /* true if and only if userselected is the empty set */
forall(userselected, player != ForbiddenUser)
  /* becomes false if the user ForbiddenUser selects a card */
size(userselected) <= 3
  /* true as long as userselected contains <= 3 elements */
player.joined==0
  /* true when there is at least one player with joined = 0 */

4 Case study

Though a more thorough evaluation would be desirable, at this stage of research we will begin with a small assessment of the language using an example application. The application we will assess this specification language with is a multi-user card game, a computer version of the game of Set. A working implementation of this application already existed. Usually, one would start specifying the constraints much earlier in development, but such methodological issues, that is, when to specify what, will not be considered here.

The suitability of the constraint language will be assessed in the following manner:

1. We give a constraint specification of the system. Since a complete specification describing all constraints of a system is not really possible, we instead try to provide a reasonable coverage without getting too verbose or complex. This specification illustrates the language, providing a means for a qualitative assessment of its usefulness. For this reason, we will give the full specification here.
2. We track how the specification co-evolved with the implementation, that is, what kind of re-design steps were prompted by the specification, and what kind of inconsistencies between specification and implementation were found.

4.1 The application

The example application is implemented using the following agents: a game agent incorporating the game rules, and standard agents corresponding to the user interface objects: buttons, textfields, textarea, lists, and canvases. First we will give a natural-language specification of the application. Then we will give the constraint specification (figure 4) and the corresponding diagrams. Figure 1 shows a structure diagram of the system, figure 2 and 3 the state diagrams of the game and user.

Natural language specification. Users are able to log in via the Web, after which they are presented with the game window. Here, they can fill in their name, create and delete games, or join games that have not yet begun. When a game is joined, the user can talk to the other users that have joined the game. Once a game is started, the cards being played with are displayed. The game proceeds as follows: 12 cards are laid out from the deck on the table. Now, the players
have to search for combinations of three cards that satisfy certain criteria; such a combination is called a ‘set’. The first one to see a set calls ‘set!’ and then has a couple of seconds to point out the three cards. These are removed and replaced with cards from the deck. The game continues until the deck is exhausted. The player with most sets wins. When no player can find any sets, extra cards may be laid out. The game should also have the option to point out sets. This pause the game, and enabling all players to see the sets on the table, after which the game may be continued with fresh cards.

4.2 Assessment of the specification process

The process of specification was found to improve insight in the application, as several changes were made to the design of the system that were prompted by insights made from the constraint specification. Assertion violation detection helped in adjusting the specification and the implementation to each other.

The program was originally developed with help of an informally-specified structure diagram specified in the earliest phases of development, which was found very useful. The formal structure diagram was however more complete and accurate. From this formal diagram, it became clear that a redundant relation was present, and that some of the references were implicit. These deficiencies were easily corrected.

Initial specification of the state machine for the game showed that the different modes that the game could be in were not very well modelled. In fact, some illegal user actions were possible under some conditions, such as selecting cards when the player did not call ‘Set’ first. After specification of the state machine and the corresponding assertions for each state, the conditions under which user actions were available or not were made quite clear. Also, it became clear that some transitions were possible that were not thought of at first, such as the transition from Playing to Watching.

5 Conclusion

While the general specification techniques used are relatively traditional, the subject and focus of the specification is relatively unusual. In this example, it resulted in a simple, straightforward specification that nevertheless covers an interesting part of the application. The approach provides a level of abstraction that may also be suitable for describing an abstract user interface model which may be filled in by alternative concrete user interfaces [14].

The close coupling of the specification with the executable system provided some extra insight, showing that such a coupling is indeed useful. Some discrepancies of previous informal specifications with the system were clarified and some failed assertion checks and state transitions uncovered further errors. Still, the resulting structure and state diagrams are relatively easy to read, and could effectively be used as part of usability specifications. The notation could provide a formal bridge between usability methods and the implementation.
Fig. 1. Structure diagram of the application. The structure diagram notation we use here is similar to the UML notation [6]. The boxes are entities. The rhomb shapes are relations. The arcs are annotated with the relational multiplicities (plus a number indicating the reference's parameter number, which is only useful for implementation purposes). The corresponding graphical notation for references in entities is similar to that often used for SQL tables. Here, an arrowhead on the reference's arc points to its target.

The entities and relations are annotated with their names (boldface, at the top), their attributes (regular font), values (regular font, names between parentheses), and events (underlined, names between parentheses). The numbers following the '/' after the values and events indicate their size, i.e. '0/0' means the attribute indicates a signal, '1/2' a 2-tuple. The default is '1/1'. The arc from setcard to game indicates a direct reference found in setcard.

The user and game entities should be obvious, but some of the user's attributes and events require explanation. game_event indicates that something happens in the game that concerns specific user, such as his/her success or failure of getting a set. selcard indicates that the user is now allowed to select cards to form a set. The other events indicate basic user interface events: typing a message, starting a new game, requesting more cards, requesting the game to show solutions, and calling 'set'.

Participation in games is modelled by the participant relation. A user may participate in at most one game, and any number of users may participate in a game. A game has a number of cards on the table, indicated by the setcard entity. Finally, when the system is displaying solutions, for each solution a triple of setcards is related through the solution relation.
This finite-state model of the user emphasises how the user may enter or leave games. The boxes indicate states, the arcs indicate transitions which are labeled by events (truth expressions) and/or conditions (truth expressions between square brackets). A horizontal arrow on the left of a state indicates the begin state. The transitions are modelled with help of a single set player, which indicates the games that the user is in, and the mode of participation. A new user is not associated with any games (Free). A user may then choose to examine a game (Watching), in which case it is not actually playing, but may view the cards and talk to the other users. Then, the user may join the game (Playing) by setting joined to 1. The other state transitions should now be obvious.

This finite-state model of the game models the different modes that a game is in. In its transitions, it uses three sets: new_game indicates that a user signals a new_game event; setcalled indicates that a user has just called ‘set’ and is now selecting cards; solutions_shown indicates that one or more solutions have been provided and are now being shown. When a game is created (Initialising), there are no cards on the table. The game starts running (Begin Turn) as soon as a user starts the game. When a user calls ‘set’, the card selection mode (SetCalled) is entered. When a user requests solutions, the game goes into solution display mode (SolutionsShown). The other state transitions should now be obvious.
entity user(username) {
  value game_event, setcalled:0;
  event speak_event, new_game/0, more_cards/0, show_solutions/0, call_set/0;
  set player("game","joined"): particpant(self,"game","joined");
  set game_event("mag"): game_event(self,"mag");
  statemachine {
    Free -> Watching {} { player.joined=0 }
    Watching -> Playing {} { player.joined=1 }
    Free {} { not exists(player) }
    -> Watching {} { player.joined=0 }
  }
  assert Free,Watching { not game_event and not setcalled }
}

entity game(name_in_progress) {
  event scoring, game_over/0;
  set new_game("player") : new_game("player") participant("player",self,1);
  set setcalled("player","joined"): participant("player",self,"joined") setcalled("player");
  set solutions_show("cd1","cd2","cd3"): solution("cd1","cd2","cd3");
  setcard("cd1",self,\_\_\_\_) setcard("cd2",self,\_\_\_\_) setcard("cd3",self,\_\_\_\_);
  set usersellected("player","card"): selected("player","card") participant("player",self,\_\_\_\_);
  statemachine {
    Initialising -> BeginTurn {} { new_game }
    BeginTurn -> SetCalled {} { setcalled }
    SolutionsShow -> BeginTurn {} { not solutions_show }
    BeginTurn {} { new_game }
  }
  assert BeginTurn,Initialising { not usersellected }
  assert SolutionsShow { not usersellected and not setcalled }
  assert {} size(setcalled) <= 1 and not exists(setcalled.joined=0)}

entity setcard(game:*:picture,xpos,ypos) {}
relation participant(user:*:game:0..1:joined) {}
relation selected(user:*:setcard:0..1) {} relation solution(setcard:0..1:1:1:setcard:0..1:1) {}
set solutions("game1","cd1","game2","cd2","game3","cd3"): solution("cd1","cd2","cd3");
setcard("cd1","game1",\_\_\_) setcard("cd2","game2",\_\_\_\_) setcard("cd3","game3",\_\_\_\_);
assert not exists({
  solutions.game1 != solutions.game2 or solutions.cd1 == solutions.cd2 or solutions.game1 != solutions.game3 or solutions.cd1 == solutions.cd3 or solutions.game2 != solutions.game3 or solutions.cd2 == solutions.cd3})

Fig.4. The full constraint specification.
This is the full specification from which the diagrams were generated. The correspondence of the supplied diagrams with the specification should be obvious. Therefore, all that is left to discuss are the assertions.

user specifies one assertion that states that the event game:enter and the value setcalled may not be defined in case the user is in the Free or Watching state.
game specifies three assertions. The first states that no user may select any cards (userselected is empty) at the beginning of the game or turn. The second states that neither may any user select cards, nor may s/he call 'set' (setcalled is empty) when the solutions are being shown. The second states that only one participant at a time may be ready to select cards, and that this user must actually participate in the game.
Finally, there is a global assertion that specifies that any solution should contain three different cards, and that these cards belong to the same game.
The structure diagram makes quite clear how agents should access their required data. For example, if we want the users to be able to view a list of participants, they can just query the participant relations. If we want users to become aware of the cards that another user is selecting, we can simply query the selected relations. This structure specification is similar in spirit to ERMA specifications, and can be used to decide on usability issues, such as which entities and relations should be visible to the users.

The state diagrams contributed to intuitive insight, but they could be more than just part of the constraint specification. They may be useful as part of the specification of the executable system itself. Once the state machines were specified, there was often the desire to read out their state from within the application. This is not exactly surprising, as much of the systems’ behaviour can be effectively expressed in terms of these states.

References


Design of Handheld Interactive Support

Carmine Ciavarella, Fabio Paternò
Via G. Moruzzi 1, Pisa, Italy
e-mail: {c.ciavarella, f.paterno}@cnuce.cnr.it
tel: +39 050 3153066
fax: +39 050 3138091

Abstract. The growing availability of small devices whose computational and interactive resources are continuously increasing in terms of power and capacity has raised an interesting discussion on how to exploit them to support users in various contexts of use. We propose a solution that can be easily adopted for users such as museum visitors. The basic elements are the use of a multimedia PDA (without the use of location-aware technology) whose main purpose is to support a user who can move freely about within a museum. The structure of the user interface allows users to easily orient themselves through appropriate visual representations and then sees to providing the information that can be interesting for them by exploiting the multimedia capabilities of the device. In this paper we describe and discuss our initial design in a concrete case study, the related user feedback received from many visitors, and the design improvements obtained by considering users’ experiences.

1 Introduction

The growing availability of small devices whose computational and interactive resources are continuously increasing in terms of power and capacity has raised interesting issues on how to exploit them to support users in various contexts of use. In this work we consider users who freely move about a building (in particular, in a museum). In such environments the most effective support is currently provided through either interactive multimedia kiosks or interactive audio recorders. In the former case, the main limitation is that the kiosk does not allow the user to move while receiving information, whereas the latter allow the user only to hear predefined texts associated with each work.

In the meantime, especially at research level, an increasing interest in location-aware systems has arisen with the goal of better assisting users. However, systems based on automatic generation of location-aware information suffer from the limitations of adaptive systems, in which often users interact with an interface that changes in the attempt to better support them, but which in doing so actually either makes wrong inferences or causes disorientation. For example, one typical problem with location-aware, in-door systems is that they can automatically generate information regarding the closest work of art while the user is actually looking at one that is lo-
cated farther away. In addition, location-aware systems often require technology that is either expensive or difficult to install in a widespread manner or do not work perfectly in all circumstances. For example, infrareds need to be installed for each work of art and require that the emitters and the receivers are lined-up in order to communicate. This can be difficult to afford, for examples in museums with hundreds of works of art and limited budget, such as the one considered in this study.

In our work we have designed an application that aims to overcome such limitations. We propose a solution that can be easily adopted without requiring expensive, sophisticated or difficult-to-install technology. The basic elements are the use of a multimedia PDA (without the support of location-aware technology) whose main purpose is to support a user who can move freely about within a museum. The user interface is structured in such a way as to allow users to easily orient themselves and then provides the information that can be interesting for them. Our evaluation involved 95 users, whose opinions and suggestions led to the final version of the application. This version is, at the present, in use and available for all the museum visitors. In this paper we discuss and report on our initial design, the user feedback, and the improved design. Some features of the two designs are also described using the ConcurTaskTrees notation in order to provide precise descriptions of how tasks are supported and their possible dynamic behaviour.

2 Related Works

The growing availability of small devices whose computational and interactive resources are continuously increasing in terms of power and capacity has raised interesting issues at research level on how to design usable interfaces for applications exploiting such devices. For example, in [9] some techniques are proposed to support multiple users communicating through heterogeneous devices. The concept of plasticity [3] has been introduced to indicate user interfaces able to support adaptation to different interaction platforms.

Some project addressed similar issues. The aim of GUIDE [11] is to investigate the provision of context-sensitive mobile multimedia support for city visitors. It is based on wireless network, located in Lancaster: information about positioning are transmitted from strategically positioned base station. This system use out-doors technologies that cannot be proposed for in-doors environments like museums. Museums are an interesting application domain for interactive mobile devices. Museum visitors can be assisted in various manners. One possibility is the use of Audio Tours: visitors can use a sort of large telephone receiver; see for example [1]. They can select the work of interest by entering its code through a numeric keypad. Audio Tours are precursors of electronic guides. They are useful but have a very limited visual channel.

This problem can be solved by using devices, like PDAs, that support both audio description and images. Another problem remain unsolved: user orientation inside the museum. An approach is suggested by IrReal project [12]: they build a navigation system using infrared transmitters. These transmitters, placed at strategically important points throughout a building, are useful to provide for way directions or other
localized dynamic information. This approach, is not completely suitable for museums where visitors have specific requirements.

A more interactive support has been adopted in the Whitney Museum of American Art in New York. The application is implemented on a tablet PC and integrates the description of the works of art with videos and interviews. The basic idea [6] is that visitors can download information from the museum web site [10] during the visit through a wireless network. The problem is that the time it takes to find and download interesting information is often long and the tablet is difficult to manage by mobile visitors, as it is rather cumbersome.

In location-aware systems the information regarding works or sections is selected depending on user position and length of stay in that position. This information is used to understand what the user’s interests are. This approach has been used for the Hippie system [4] within the HIPS [5] project. The authors have also considered how to effectively present information to the user while taking into account the user model (interests and preferences of users). This project also addresses the problem of how to adapt the user interface to the user model. The model can be modified either directly by the user at the beginning of the session or by the system’s taking into account the history of user interactions and the choices performed by the user; in both cases the system highlights proposals for further information to the user through a blinking light-bulb. The suggested information can be accessed through links to the descriptions of the works that best correspond to the current user model. When accepted the suggestions are used to update the user model.

The limitation of this approach is that often the user’s position alone is not enough to indicate interest in the closest work of art; external reasons, such as a crowd preventing movements, can be the reason for a user’s stopping. Thus, the risk is that the system wrongly identifies the user interests and determines the corresponding user model.

One solution to the limitations of location-aware systems has been proposed [2] for visiting “Filoli”, a Georgian Revival house. In this case the application provides the users with an image of the current room with the works of interest highlighted by red borders. Then, the user can select the object of interest with a pen, which activates an audio comment or a video. It is possible to change the viewpoint of the room’s representation by selecting one of the device’s buttons. In this case, one possible limitation is the use of pictures to represent the room content duplicating the information that the user is already seeing, with the risk of requiring multiple interactions to identify the selectable elements of interest. In addition, this solution is valid only for those museums where the elements of interest are arranged along each wall, while it becomes difficult to follow in cases where they are spread throughout the room.

In the next section we will discuss our solution that, while agreeing with the approach proposed in the aforementioned work on the limitations of location-aware systems, proposes a different way to represent and select information of interest.
3 Initial Design

In our approach the design is driven by three main elements: the context of use that includes both the device used for the interaction and the environment where such interaction occurs, the tasks users wish to perform and the objects they need to manipulate in their performance (both interface and domain objects).

3.1 Context of use

For the context of use, we consider both the interaction resources used and the environment where the user performs the tasks.

The application has been developed on a Compaq Ipaq 3660, with windows CE and additional 64 Mbytes Flash Memory Card. We decided to use text-to-speech synthesis for supporting audio comments. Unfortunately, the possibility of dynamic text-to-speech generation is not supported in these environments because the necessary libraries are lacking for Windows CE. In addition, the synthesized Italian voice was considered too unpleasant and was replaced with audio-recorded comments.

Currently, the application contains description of about 130 works of art, each of them with an associated Jpeg picture (dimensions are about 140x140 pixels). The audio files are in MP3 format. For the English version we have used text-to-speech provided by Text Aloud MP3. Overall the application requires about 30 Mega of memory.

The application has been developed for the Marble Museum. The managers of the museum decided to provide their visitors with information additional to that contained in traditional labels. They often had the problem to find guides able to provide such information and in some cases the guides were not able to communicate with foreign people. The structure of the museum forces to some extent the order of visit among the rooms. Such rooms contain many types of objects from the ancient Romans to pieces of quarrying technology of the past century. Thus, visitors need support able to interactively select those more interesting for them and receiving related information.

3.2 Tasks

In the design of the user interface we considered three types of tasks that users can perform in the context considered:

- **orientation within the museum**, for this purpose three levels of spatial information are provided: a museum map, a section map, and, for each physical environment composing the section, a map with icons indicating the main pieces of work available in the room and their location. By selecting such icons the picture of the related element is displayed along with some basic information and the corresponding audio description is activated. The purpose of the picture is not to show the details of the work of art (that is supposed to be in front of the user), but to allow users to check that the information they are receiving regards the work that they are viewing.
• control of the user interface, for example, to allow changing the volume of the audio comments, to stop and start them, and to move through the various levels of detail of the museum description;

• access to museum information, also this is provided at different abstraction levels (museum, section, physical environment, single work).

At any time the application was able to highlight where the users are in the museum area, assuming they are in the same room as the works last selected (see for example Figure 1). The orientation information was triggered by selecting the “i” button on the bottom menu-bar that appears when the map of a physical environment is displayed (for example, see Figure 3).

In order to represent the task model underlying our design we have used the CTT (ConcurTaskTrees) notation [7]. It also allows us to provide precise descriptions of how tasks are supported and their possible dynamic behaviour.

Figure 2 shows an excerpt of the task model associated with the initial design of the application developed, in particular, it represents the initial part of a user session.

In the first version, the application’s opening presentation contained basic information regarding the museum, then, after user selection (StartVisit task), the application identified the current level of information to provide (CalculateApplLev task) and showed the museum map and provided audio comments regarding the museum and its history. In the map users could select the section (SelectSection task) where they were
in order to obtain the related section map with indications of the positions of artworks while receiving audio comments on the section. It was also possible to display information regarding the location of the current room in the map and explanation of the icons and controls available in the user interface. The part of the task model related to the visit of the selected section is not expanded for sake of brevity.

3.3 Objects

The information regarding the museum and the works that it contains is provided using both the audio and the visual channel. The visual information is mainly used to allow users to orient themselves and receive some supplementary information. It provides information at different logical levels:

- **The museum**, it displays a map that shows the logical organization and the physical structure of the museum.
- **Sections**, the map of each thematic section of the museum is provided, when it covers multiple physical environments it is possible to select each of them to get more detailed related information,
- **Environments**, they are either rooms or separate environments partitioned with various techniques; the system provides a map with icons indicating where the main objects of interest are located;
- **Works of art**, in this case a picture and basic information are provided.

Since the museum considered is an interdisciplinary museum that contains various types of works, different icons are used to represent each type:  

---

Figure 2: Part of the task model associated with the initial design
for showcases containing historical artefacts, models or reproductions;

for pictures or photos hung on the museum walls.

for capitals.

for representing sculptures.

for the marble exposition.

for the reading-desks.

An alternative solution would have been to use pictures of the works considered in the room map instead of icons. However, the resolution of the PDA (240x320) would have made it difficult to interpret such images. The picture below shows an example of room map annotated with icons highlighting the main works of interest. The doors represented in the map are interactive and allow the user to change the room representation in the PDA (while physically moving in the new room).
4 Presentation of information and navigation

The audio part has been implemented reflecting the logical structure of the information to provide. There are comments introducing the museum, its sections, each environment and each work located in them. They are provided in two languages: English, using text-to-speech synthesis. The resulting audio message is a bit metallic, but clearly understandable even by non-native English speakers. The other language is Italian, for which a pre-recorded female voice was used for the comments because the synthesised speech was considered unpleasant.

The resulting navigation was based on the museum map. The user started the visit from the museum map. Then, they selected the section of interest. Lastly, in the section map they could select the physical environment of interest and at any time they could go to the museum map to select another area. Figure 4 shows an example of sequence of accesses.
Figure 4 Levels of navigation in the first version of the guide.

5 Evaluation

This evaluation aimed to establish whether our electronic guide satisfied museum visitors’ demands. We were particularly interested in determining:

- If our user interface allowed visitors to use the application properly;
- The quality and quantity of information provided about the museum art;
- How well our system helped visitors to orient themselves in the museum.

The test took place in the Marble Museum of Carrara during the summer, when the number of visitors is highest. The evaluation involved tourists who were given a PDA on which our application was installed. After their visit to the Museum, they were asked to answer some written questions about:

- Their previous experiences visiting museums (7 questions).
- Quality of information given (15).
- Quality of images and audio presentation offered (4).
- The electronic guide used during the visit (20).
- Some personal information.

The questionnaire was composed of items asking for a rating on a scale from 0 to 10 and open questions asking users to provide comments or suggestions on each aspect of the electronic guide. In the following we discuss the results of the questionnaire.
5.1 The users

The test involved 95 users; 34 of whom were Italian. The following charts show some data about users.

**Nationality of Users**

- Italian: 36%
- Others: 64%

**Visitors**

- Male: 9%
- Female: 29%
- NA: 62%

**Frequency of museum visits**

- Never: 13%
- Rarely: 3%
- Frequently: 18%
- Very Frequently: 66%

**Educational level**

- Degree: 54%
- Upper Secondary school: 15%
- Lower Secondary school: 4%
- NA: 27%

**Used PDA?**

- Yes: 36%
- No: 51%
- NA: 13%

**Type of use of PDA**

- School: 57%
- Job: 18%
- Fun: 18%
- Other: 7%

Figure 5: PieCharts representing users’ characteristics.
The foregoing charts reveal that most visitors were male (62%). The average age was 35. Most visitors (28%) were aged between 31 and 40 and had a university degree (54%). Moreover, a large majority (66%) visits museums frequently: 28% of these in fact had had at least one experience with electronic museum guides. An interesting fact was that only 36% of visitors had previously used a PDA, and most of them (54%) only for work. These data highlight that our system was quite novel, in both its aspects: the concept and the device itself.

The test results have been grouped according to three criteria: frequency to museum visits, nationality and experience with electronic guides. The first criterion was adopted to check the quality and quantity of information. The second one is useful to understand how much the choice of synthetic versus natural voice can affect the user’s interaction with the electronic guide. The third criteria was chosen to analyse the reactions of those who had already had experience with electronic guides in museum visits with respect to those who had never used similar tools.

In the next diagrams we report the mean of the user ratings for each aspect considered and, when meaningful, the standard deviation.

5.2 Evaluation of quality of information given

These questions regard the information concerning the museum, its sections and the works of art. Users are grouped according to the frequency of museum visits. We have identified four categories: users who had never visited a museum before (3%); those who visit museums rarely (18%), often (66%) and frequently (13%).

Visitors who had never visited museums before particularly liked the information regarding the museum and the works of art (mean rating was about 8 on a scale of 0 to 10, with a standard deviation of 1.5), while they found information regarding the section less complete (mean rating about 7). One of them suggested providing additional information.

Users who rarely visited museums liked all the types of information provided by the guide (mean rating 7.5, standard deviation 1.5). One interesting suggestion regarded proposing alternative itineraries for the visitor to follow.

Frequent visitors liked the information provided and gave similar ratings. They suggested providing additional information on the living and working conditions of marble workers and on how marble is currently quarried. One of them commented: “I enjoyed being able to freely move about the museum, to stop where I wanted and to get related information”.

As Figure 6 shows, very frequent visitors preferred the information regarding the sections of the museum. They suggested introducing videos on the life of marble workers and techniques for working marble.
5.3 Evaluation of quality of images, audio presentation and maps

In this case we consider the quality of audio comments and of the images used to show works of art and maps. As mentioned before, the only difference between the version for Italian visitors and foreign visitors was in the audio presentations: in the Italian version audio files were obtained by recording an actual woman’s voice, whereas in the version for foreigners we used a synthetic English female voice. Thus, if we divide the tests between Italian and foreign visitors, we can study how the choice of using a synthetic voice affects the usability of the application.

Italian visitors enjoyed the presentations provided by the application and gave very high ratings (on average about 8). The audio presentations received high ratings with a mean of 9 and standard deviation of 1.25. They provided some suggestions such as the possibility of receiving more in-depth information on request and longer comments.

The ratings provided by foreign visitors where lower than the previous category. As can be seen in the diagram, the mean rating regarding audio presentations was 6 with standard deviation 2.39. The images and maps received better ratings (about 7.5 in both cases), though the standard deviations were still high (respectively 1.8 and 2.3). This may indicate that the problems with the audio presentations have somehow affected the evaluation of the other elements.

Some comments explicitly indicated that the audio presentations were considered monotonous, that is, lacking natural prosody and therefore sounding too “metallic”.

Figure 6: Evaluation of information given
Some suggested using a more human voice and adding also Germany and French to the languages supported.

Figure 7: Evaluation of quality of images, audio presentation and maps

The next diagram can be useful to understand how much the quality of audio presentations has affected the use of the electronic guide.

Figure 8: Italian and foreign visitors evaluation of guide
This diagram highlights the difference in evaluation between Italian users (who heard a human voice in the audio presentations) and the others. Apart from the evaluation of the graphical interaction, the ratings related to the usability of the electronic guide are substantially different in the two groups of users.

5.4 Overall evaluation of our electronic guide.

In this case the questions concerned an overall evaluation of the electronic guide: its utility, usability, and user interface design. To analyse the results of the test we have divided the users who had previous experience with palmtops from those who used it for the first time.

The novices were the majority, as shown by the diagram (they were 65). They liked the user interface design more than the utility or usability of the application. The mean rating for these three parameters was similar (about 7) but the standard deviation was rather different (1.4 versus 2.5). One user commented “At the beginning some confusion, but then it was easy and useful”. Some suggestions regarded the possibility of adding videos and identifiers in the rooms in order to ease the visitor’s orientation in the museum.

The palmtop expert visitors were 27. They provided judgments similar to the other visitor category. One visitor commented “I liked to see works of art without having to stop to read the labels”. Some of them had some problems orienting themselves in the museum.

![Quality of Guide](image)

Figure 9: Novice and export visitors evaluation of guide.
5.5 Evaluation summary
As we can see from the previous analysis, the aspect of our system that was rated most highly was the quality of information offered.

Other highly rated aspects were:
- the opportunity to visit museum by oneself.
- the exhaustive information given in a timely fashion.

Users’ most frequent suggestions were:
- To give more information about the geological formation and working of marble, and the life of quarry-men.
- To supplement the information with video (e.g., about the working of marble).
- To use a human voice in the English audio presentation.
- To add presentation in other languages, especially French and German.
- To use arrows, colours and numbers to help users orient themselves in the Museum.

Another interesting observation was that most visitors preferred to use the guide without headphones because they wanted to share it in small groups (two or three people).

6 The Improved Version

In the new version we have taken into account the results of the user tests. In particular, we wanted:
- A different way to navigate that would allow users to better orient themselves;
- Better highlight how to get into and out of the rooms;
- Different ways to provide help information
- Support use of videos played through the PDA to enrich the user experience.

Part of the new design is described by the next CTT specification. Again, we consider the initial part of a user session. The main differences are: the general presentation regarding the museum is provided as soon as the user starts the session (StartSession task), we added a system task showing explanation of the icons used in the application (ShowIconExplanation task), the access to the section is not performed through the museum map, and the system provides explicit indication of the section position in the museum (LocalizeSection task) in order to help user’s orientation.
The reason for these changes was that users always start their visit from a given point and that the structure of the museum imposes a linear path to the visit. Although visitors can at any time go up and down along this path and get disoriented, they cannot take a different route. Thus, we decided to have the PDA displaying where they are at the outset of the visit. Next, after having shown some basic information on how to use the application (such as the meaning of the icons), the application displays a map of the first museum section while providing audio information about it. Sections are made up of one or more rooms with common theme. Then, they can select a room and receive indications as to where it is, followed by its map with the possibility of selecting specific works of art and receiving related information. Figure 11 shows an example of this pattern of interaction from the starting point. Arrow heads on each door in the map of the room clearly highlight the suggested order of access. When they move to the next room they can just select the related door in the map and the new room map will appear. If a new section is encountered, then a general map with related information is first provided and subsequently a map of the selected room is displayed. One of the main differences with respect to the previous version of the application is that user access to information is no longer driven by the museum map. However, the overall museum map is still available on request in the event that visitors do not want to follow the path suggested by the physical museum structure.
7 Conclusions and Future Work

The new version is now available to the museum visitors. We are planning a new evaluation study with revised questionnaires and automatic analysis of logs of user interactions.

The system has been implemented with Embedded Visual C++ 3.0 as programming language and the Microsoft Foundation Class toolkit for the user interface development.

Technology for location detection is improving in terms of both cost and accuracy. Its introduction (for example with Bluetooth) will be investigated in the near future, at least to automatically identify the room where the user is.

Future work will be dedicated to identifying adaptive features of the application that can increase the users’ interest without disorienting them.

Acknowledgements

We gratefully acknowledge support from the Carrara City Council and the Marble Museum and the Cameleon project (http://giove.cnuce.cnr.it/cameleon.html).

References


Performance Evaluation as a Tool for Quantitative Assessment of Complexity of Interactive Systems

Xavier Lacaze¹, Philippe Palanque¹, David Navarre¹, and Rémi Bastide¹

LIHIS-IRIT, University of Toulouse III
118, route de Narbonne
31062 Toulouse Cedex 4, France
{lacaze, palanque, navarre, bastide}@irit.fr

Abstract. While research in the field of HCl seems to focus on usability issues real-time safety critical systems deals with specific requirements that make usability only one of the main factors that designers of such systems have to deal with. Indeed, reliability and efficiency are also to be taken into account. This paper presents extensions to previous work done in the field of formal description techniques for interactive systems by adding temporal elements to the ICO formalisms. These temporal elements are used for evaluating in a quantitative way the relative efficiency of different designs. Such results are aimed at supporting design decisions in the design process of safety critical interactive systems.

1 Introduction

Interactive systems development process promotes iterative user centred design allowing for testing and evaluating the usability of the system with users. This kind of approach leads to usable systems that meet user needs and user activity. The primary factor favoured by such approaches is usability and it is well agreed upon that this is at the detriment of other factors such as reliability, safety or efficiency.

It is reasonable and even desirable to have usability as the primary factor when business-oriented applications are considered. Indeed, such applications target routine tasks and a large number of potential users. Considering usability as a primary concern allows reducing training costs that could be prohibitive due to the large number of users. Those applications generally present the property of reversibility i.e. that most actions can be reversed in order to put the application back to its previous state.

Safety critical interactive applications require more attention to other factors such as reliability and efficiency even though usability is also of primary concern. However, for such applications efficiency may be considered as a priority with respect to usability. In such cases difficulties faced by users may be reduced by extensive and appropriate training. Command and control applications in the field of military systems belong to this type of application where performance
and efficiency is of first priority. Indeed, the ability for an operator to respond in a timely manner or to handle multiple requests at a time may be of primary concern for the safety of the whole operation. Of course in such real time systems there is (most of the time) no possibility to put the system back in a previous state as its evolution is mainly based on time rather than on sequences of events produced by users when interacting with the system (as it is the case for most business-oriented applications).

In this paper we present a set of temporal extension to the ICO formal description technique allowing for quantitative modelling of time in order to assess the complexity of several designs. These extensions are exploited within a methodological framework that provides guidelines on where temporal elements should be embedded in the models.

The content of this paper has been developed within a military-funded research project on command and control interactive systems for drones (UAV \(^1\)) for which the three factors usability, reliability and efficiency are of equivalent concerns. However, due to confidentiality concerns and by sake of simplicity we will present the use of the temporal extensions and its related framework on a simple case study.

Section briefly recall the basic principles of the ICO formal description technique and its case tool PetShop.

Section describes the case study based on currency converters which aims at presenting how the temporal extensions introduced in the ICO formal description technique are integrated into models.

Section is dedicated to performance evaluation and present how ICOs temporal extensions allow for quantitative and objective assessment of complexity and efficiency between two different designs.

Section presents how PetShop is currently extended for editing and simulating temporal extensions of ICOs.

Section presents conclusions and perspectives for future work.

2 Interactive Cooperative Objects

This section is dedicated to the ICO formal description technique. First an informal presentation of the formalism is given. Temporal aspects that have been added to the ICO formalism are then presented.

2.1 Informal Description

The Interactive Cooperative Objects (ICOs) formalism is a formal description technique dedicated to the specification of interactive systems [0]. It uses concepts borrowed from the object-oriented approach (dynamic instantiation, classification, encapsulation, inheritance, client/server relationship) to describe the structural or static aspects of systems, and uses high-level Petri nets [0] to describe their dynamic or behavioural aspects.

\(^1\) Unmanned aerial vehicle
Petri Nets is a graphical formalism made up of four components: the state variables (called place, depicted as ellipses), states changing operators (called transitions, depicted as rectangles), arcs, and tokens. Tokens are hold by places; arcs link transitions to places and places to transitions. The current state of a system is fully defined by the marking of the net (i.e. both the distribution and the value of the tokens in the places). For a state change to occur a transition must be fired. A transition is fireable if and only if each of its input places holds at least one token. When the transition is fired, one token is removed from each input place and a token is deposited in each output place. The functioning of a Petri net model is presented on the case study in section.

ICOs are dedicated to the modelling and the implementation of event-driven interfaces, using several communicating objects to model the system, where both behaviour of objects and communication protocol between objects are described by Petri nets. The formalism made up with both the description technique for the communicating objects and the communication protocol is called the Cooperative Objects formalism (CO and its extension to CORBA COCE [0]).

In the ICO formalism, an object is an entity featuring four components: a cooperative object with user services, a presentation part, and two functions (the activation function and the rendering function) that make the link between the cooperative object and the presentation part.

Cooperative Object (CO): a cooperative object models the behaviour of an ICO. It states how the object reacts to external stimuli according to its inner state. This behaviour, called Object Control Structure (ObCS) is described by means of high-level Petri net. A CO offers two kinds of services to its environment. The first one, described with CORBA-IDL [0], concerns the services (in the programming language terminology) offered to other objects in the environment. The second one, called user services, provides a description of the elementary actions offered to a user, but for which availability depends on the internal state of the cooperative object (this state is represented by the distribution and the value of the tokens (called marking) in the places of the ObCS).

Presentation part: the Presentation of an object states its external appearance. This Presentation is a structured set of widgets organized in a set of windows. Each widget may be a way to interact with the interactive system (user system interaction) and/or a way to display information from this interactive system (system user interaction).

Activation function: the user system interaction (inputs) only takes place through widgets. Each user action on a widget may trigger one of the ICO’s user services. The relation between user services and widgets is fully stated by the activation function that associates to each couple (widget, user action) the user service to be triggered.

Rendering function: the system user interaction (outputs) aims at presenting to the user the state changes that occurs in the system. The rendering function maintains the consistency between the internal state of the system and its external appearance by reflecting system states changes.

329
ICO are used to provide a formal description of the dynamic behaviour of an interactive application. An ICO specification fully describes the potential interactions that users may have with the application. The specification encompasses both the “input” aspects of the interaction (i.e. how user actions impact on the inner state of the application, and which actions are enabled at any given time) and its “output” aspects (i.e. when and how the application displays information relevant to the user). Time-out transitions are specials transitions that do not belong to the categories above. They are associated with a timer that automatically triggers the transition when a dedicated amount of time has elapsed.

When included in a system model such transition is considered as a system transition. They can also be included in a user model representing spontaneous user’s activity. Such modelling is presented in section 4.

An ICO specification is fully executable, which gives the possibility to prototype and test an application before it is fully implemented [0]. The specification can also be validated using analysis and proof tools developed within the Petri nets community and extended in order to take into account the specificities of the Petri net dialect used in the ICO formal description technique.

2.2 Temporal Aspects

One of the advantages of Petri nets with respect to other formal description techniques is their possibility to handle in a very efficient way both qualitative temporal aspects (before, after, meanwhile, ...) and quantitative temporal aspects (seconds, dates, stochastic durations, ...).

Quantitative time is what is used for performance analysis. Several temporal extensions to basic Petri nets have been proposed in order to deal with it.

Adding Time to Transitions. Adding time to transitions is the first extension that has been proposed. The basic idea in this work is that firing of a transition can take time. This has been used, for instance, in the field of flexible manufacturing systems in which an action performed can take some time.

First work described in [0] proposed timed transitions. For each transition of this type in a Petri net, a predefined firing time \((d_t)\) added. Tokens used for firing are reserved by the transition and when time has elapsed the transition is fired removing the reserved tokens.

Other work such as [0] proposed to add indeterminism in duration. For each transition an interval \([t_{\text{min}}, t_{\text{max}}]\) is added to each timed transition. In this temporal model the models tells that the firing of the transition takes more than \(t_{\text{min}}\) and less than \(t_{\text{max}}\). The precise amount of time taken for the firing is unknown at first.

Adding Time to Place. Adding time to places is another temporal extension in Petri nets. The underlying idea is that places in the Petri net model represent activities or workstations for instance and that transitions only represent the moving of resources from one workstation to another one. This work has been presented in [0]. As for transitions, duration \((d_t)\) is associated to the place. If a token enters a place then it will not be available for a firing before the delay \(d_t\) has expired.
The main problem of these two approaches is that they require managing two different markings in the Petri net: available marking and reserved marking. This makes most of the analysis techniques not available for these Petri net dialects.

**Stochastic Time.** Generalised Stochastic Petri Nets [0] associate time to transitions. However, GSPN feature two main differences with respects to the Petri nets presented above. The first one is the stochastic nature of duration i.e. duration follows a distribution of random variables. The second one is the fact that time is associated with tokens. When a transition is fireable a timer is set with each token that makes this transition fireable. The timer goes on until the duration associated to the transition has elapsed. If the tokens are still available then the transition is fired. Meanwhile the tokens may have been used by other conflicting transitions that would prevent the timed transition for being fired when the time has elapsed.

The work presented in this paper exploits this kind of time modelling in Petri nets. We have already used temporal extensions for performance evaluation in previous work [0] and this work builds upon it and goes further by embedding values coming from human factors studies and by providing a framework for going from a system model to a extended system model dedicated to performance evaluation.

### 3 Currency Converters Case Study

Prior to the arrival of the Euro (1st January 2002) and its apparition as a real currency in the wallets of most citizens in the Euro zone, banks and other organisms distributed to their clients various currency converters. As quite a lot of people have been using extensively these systems recently, we have decided to take them as the basis for the description of how performance evaluation techniques could help in evaluating the complexity of interactive systems.

#### 3.1 Informal Presentation of Currency Converters

As a basis for the performance evaluation we have taken two currency converters provided by the same bank. The first one was issued by the bank before about one year prior to the deployment of the Euro while the second one was issued in late fall 2001. The first one is on the left-hand side of Fig. while the second one is on the right-hand side. The first one features only one display while the second one is offering two.

The first one is a basic currency converter. It is made up of buttons for interaction and one display. A button labelled rate is also available for changing the rate between Franc and Euro (this offers the opportunity to use this converter for other currencies than Euro). The use of the converter is quite trivial and can be guessed after few trials. A new rate can be set by first entering the digits for the rate and then pressing the button “Taux²”. A conversion can be made by

² “Taux” is the French word for rate
entering the digits corresponding to the amount you want to convert and then pressing either the red button with the € (for Euros) symbol or the red button with the F (for Francs) symbol.

The second convertor offers more user services than the first one. Some buttons appear both on the first and the second system but three main features make it significantly different:

- it proposes two displays,
- it is impossible to change rate,
- it allows for basic arithmetic operations.

Another user service is provided by this converter the calculation of change merged with a change of currency. This is due a political decision within the Euro zone to allow customers to use both currencies for a limited period of time. During this period, customers were allowed to pay in the national currency but the change would be given back in Euro. This task is quite complex for a customer not equipped with a currency converter. Indeed the customer has first to remember the price in Euro of the product to be purchased to pay with national currency notes (after a conversion from Euro to the national currency) and receive the change in Euro. To check if the change is correct another conversion has to be made. In order to support this task the second convertor is equipped with a button labelled “Paie\(^3\)”

Using the second convertor the customer only needs to enter the price in Euro, press button “Paie\(^3\)”, enter the amount of the note in national currency, press the button labelled “Chg\(^4\)” and the system displays the change in Euro.

The second convertor is modeless as both displays are used at the same time. The user only has to select which one is the input as the upper display is dedicated to Euros and the lower one to the national currency (actually French Francs). To use this converter, your first action is to choose the currency for which you need to enter an amount. Pressing the button labelled “€/F” switch the input to the currently unselected display.

\(^3\) “Paie” is the French word for pay
\(^4\) “Chg” stands for change
3.2 Tasks Models

Tasks models are used in this paper to represent possible activities from the user that will be used for comparing the respective efficiency of the converters.

We use ConcurTaskTree notation to represent tasks [0]. CTT represents hierarchical level and temporal relation between elementary tasks. CTT defines four types of tasks: abstract task (represented by a cloud), system task (represented by a computer), interaction task (represented by a user in front of a computer), and a cognitive task (depicted as a user). These tasks are organised hierarchically from abstract to concrete levels. The temporal relationships between tasks of the same level are described by means of temporal operators. $T_1||T_2$ indicates the tasks can be performed in concurrent way. $T_1||T_2$ describes a choice between two tasks. $T_1 >> T_2$ is used for describing a sequence of tasks i.e. $T_2$ only starts when $T_1$ is completed. $T_1[ ] >> T_2$ expresses a sequence between tasks and that some information from the task $T_1$ are transmitted to task $T_2$. $T_1> T_2$, when $T_2$ starts the task $T_1$ is disabled. $[T_1]$ models the fact that task $T_1$ is optional while $T_1^*$ model iteration.

Task Models for the Currency Converters. We construct the task model with the CTT graphical editor called CTTE. We decomposed the task model of the first convertor in two major subtasks: change rate and convert a value, see Fig. . A task model for the second currency converter is presented in Fig.

![Fig. 2. A task model for the first currency converter](image)

This task model is divided into two subtasks: verify the mode and convert a value. These two task models feature commonalities and discrepancies. Both task models contain the subtask “convert a value” whereas the other subtasks are specific for each currency converters (“change the rate” and “verify the mode”).

Scenarios. Using CTT Environment it is possible to extract scenarios from the simulation of task models. We have extracted the same scenario (convert 10 Euros) from these two tasks models. In order to take into account the values the scenario is transformed into a high level Petri net (the same dialect as the one used for modelling the ObCS of the objects). As the tasks are not supported in the same way by both currency converters two Petri nets have been built. These two Petri nets are presented in Fig. . Common actions between scenarios are:
switch on the converter, read the value, press_1 and verify press_0 and verify, read the converted value. For the first scenario we suppose that the correct rate has been already set. If this is not the case the user will have to test the converter for instance by entering a basic value (one Euro for instance) and checking it with respect of some knowledge he/she has or some external material. Such considerations make a huge difference between the two devices under consideration but for space reasons we do not report them here.
3.3 System Model

In this section we describe, using high-level Petri Nets, the system model of each currency converter. We do not use all the features of the ICO formalism as performance is mainly based on sequences of actions rather than on presentation aspects (even though the layout of the converters will also be used). Input is also not considered as for both converters inputs only occur through similar press buttons.

**First System Model.** As introduced in section the model of the system integrates external and behavioural aspects. Fig. presents the behaviour of the first currency converter. For sake of simplicity the dot button is not represented. Fig. presents an excerpt of the activation function of the first currency converter. First column of the activation function represents the interface element (widget), second column the event corresponding to user action on the widget. And the last column represents user services and its related transition in the ObCS.

<table>
<thead>
<tr>
<th>Widget</th>
<th>Event</th>
<th>User Service (transitions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button ON</td>
<td>Press</td>
<td>On (ON, OFF, ON, OFF)</td>
</tr>
<tr>
<td>Button P</td>
<td>Press</td>
<td>Franch (Franca)</td>
</tr>
<tr>
<td>Button €</td>
<td>Press</td>
<td>Euros (Euros)</td>
</tr>
<tr>
<td>Button 0</td>
<td>Press</td>
<td>Zero (TD, Off, D)</td>
</tr>
<tr>
<td>All buttons</td>
<td>Press</td>
<td>Off</td>
</tr>
<tr>
<td>Button Taux</td>
<td>Press</td>
<td>Taux</td>
</tr>
</tbody>
</table>

**Fig. 5.** Excerpt of the activation function of the first currency converter

Fig. presents the rendering function i.e. how internal state of the system is presented to the user. The first column contains the name of the place in the ObCS, the second column describes the state change (for instance a token has been removed from a place or has entered a place) and the last column the method triggered when the state change occurred.

<table>
<thead>
<tr>
<th>ObCS Element</th>
<th>Feature</th>
<th>Rendering method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place Off</td>
<td>token entered</td>
<td>Show (Off)</td>
</tr>
<tr>
<td>Place On</td>
<td>token entered</td>
<td>Show (token)</td>
</tr>
<tr>
<td>Place RateValue</td>
<td>token entered</td>
<td>No rendering</td>
</tr>
</tbody>
</table>

**Fig. 6.** The rendering function for the first currency converter

In the initial state place Off holds a token representing the fact that the system is not active. Place RateValue holds a token too and this token is not of a basic type as it contains a value (the actual value of the rate). By default we consider the official rate 1 € = 6,593,57 FF.

Places Off is of basic type i.e. the tokens in the place are simple token carrying no value. At the contrary place On is a triplet < v, n, t >, where v
represents the current value presented on the calculator, \( n \) the number of digits and \( l \) the last button pressed.

Whatever state the system is in, at least one transition for each user service is always available. This is typical of non software systems where it is impossible to disable interactive objects. In classical interactive systems when no transition related to a user service is available the corresponding interactive object is disabled. Transition \( Time\_out \) is not related to a user service. It is a temporal transition that fires after a predefined period of time after becoming available.

**Second System.** In order to take a common ground for comparing the converters in the second converter system model we do not take into account new functions with respect to the first one (+, -, *, and /). Fig. presents the activation function and Fig. the rendering function corresponding to the behavioural model in Fig. .

A significant difference in the behaviour of the second converter is related to button €/F. Indeed, according to the state of the system pressing this button changes mode from Euro to French Francs and vice versa (if there is no value entered) or plays as a backspace key removing the last digit (if a value has already been entered).

Activation and rendering functions must be read in the same way as for the first converter (see previous section).
Fig. 8. Excerpt of the activation function of the second currency converter

<table>
<thead>
<tr>
<th>Widget</th>
<th>Event</th>
<th>User Service (transitions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button ON/C</td>
<td>Press</td>
<td>On (ON, ON, ON, ON)</td>
</tr>
<tr>
<td>Button Face</td>
<td>Press</td>
<td>Face (Face, Face, Off, Off)</td>
</tr>
<tr>
<td>Button Cig.</td>
<td>Press</td>
<td>Cig (Change, Change, Off, Off)</td>
</tr>
<tr>
<td>Button 0</td>
<td>Press</td>
<td>Zero (TOJ, TOJ, Off, Off)</td>
</tr>
<tr>
<td>All buttons</td>
<td>Press</td>
<td>...</td>
</tr>
<tr>
<td>Button /F</td>
<td>Press</td>
<td>Convert (BackSpace, BackSpace, BackSpace, BackSpace, BackSpace, BackSpace)</td>
</tr>
</tbody>
</table>

Fig. 9. The rendering function for the second currency converter

The system model of the second converter is more complex in terms of number of Petri net components. Place Off models the same behaviour as for the first converter. Places Euro and Franc models the two input modes for the system. Indeed, user can choose to enter either Euros or Francs. When the system is switched on the token in place Off moves to place Euro (default input mode) and this is represented in Fig. by transition ON_C relating place Off and place Euro.

Fig. 10. The system model of the second converter
Tokens that can be stored in places Euro and Franc are not of Basic type. Type of the tokens is a 5-uplet $< f, e, n, p, l >$. $f$ (resp. $e$) contains the amount in French Francs (resp. Euro), $n$ represents the number of digits in the zone entered, $p$ the value typed-in by the user after pressing the PA button, and $l$ the last button pressed by the user. This variable is dedicated to performance evaluation activity as it stores the last position of users finger on the converter.

4 Performance Evaluation

This section is dedicated to performance analysis. This analysis aims at evaluating among several designs which one is the most efficient. In this work, efficiency is measured according to the time needed for a user to perform a scenario which is extracted from a task model. Therefore, hereafter, we use the tasks models, the scenarios and the system models presented above.

4.1 Performance Analysis Framework

In order to be able to make some performance analysis, it is necessary to add temporal information to the models. With respect to previous work we proposed in this field [0] this paper presents a framework describing how such temporal information should be added to the models and present precisely how accurate information could be calculated using human factors results on human performance.

4.2 Temporal Information Modelling

Main user model like for instance ACT-R [0], EPIC [0], and the human model processor [0] promote decomposition of users behaviour in three main components. According to terminology used in [0] these three components are: Motor processor, Perceptual Processor and Cognitive Processor.

In earlier work we proposed to add temporal information to transitions in the system model represented using the ICO formalism. Even though this approach works for generic performance evaluation it does not allows for more complex and specific performance evaluation taking into account, for instance, factors such as the impact of previous action on the performance of next action. In order to deal with such specific aspects that have a real impact on performance evaluation we propose a different approach based on the extension and the transformation of the system model. With adequate modifications we are able to represent temporal information relating to interaction history (if needed). Information related to user activity (both cognitive and perceptual) cannot be added to system model. Thus, we build a dedicated model for each of these elements that will be extended with temporal information.

Motor Activity. Human factors resources we have been investigating two approaches: the keystroke-level model [0] and Fitts law [0]. The first approach has not been used in this study as it is more suitable for interaction with keyboard
and mouse and only provides average values. We decided to use Fitts law is more accurate and its application to motor movements can be directly exploited. Fitts law is presented in Formula (1) and is more general as it represents an index of difficulty for reaching a target (of a given size) from a given distance. Movement time for a user to access a target depends on width of the target (W) and the distance between the start point of the pointer and the center of target (A).

\[ MT = a + b \log_2(2A/W) \]  

(1)

For predicting movement time on the systems under consideration constant are set as follows: \( a = 0 \) and \( b = 100ms \) (mean value for users).

![Fig. 11. Distance between buttons in first system](image1)

Values in Fig. and Fig. are used together with Fitts law to produce movement time matrix presented in Fig. . For space reasons we do not present here the same elements that have been used to calculate movement times presented in Fig. .

![Fig. 12. Buttons width](image2)

According to the restrictions made of the second converter (not taking into account +, -, *, /, ) each converter is made up of 14 buttons as modelled in Fig. and Fig. . Fig. presents the system model extended in order to embed temporal information for Fig. . For sake of simplicity we do not present the complete
Fig. 13. Temporal values (in ms) for user interaction using Fitts law (first converter)

<table>
<thead>
<tr>
<th>Time</th>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>On</td>
<td>120</td>
</tr>
<tr>
<td>0.2</td>
<td>On</td>
<td>210</td>
</tr>
<tr>
<td>0.3</td>
<td>On</td>
<td>310</td>
</tr>
<tr>
<td>0.4</td>
<td>On</td>
<td>410</td>
</tr>
<tr>
<td>0.5</td>
<td>On</td>
<td>510</td>
</tr>
<tr>
<td>0.6</td>
<td>On</td>
<td>610</td>
</tr>
<tr>
<td>0.7</td>
<td>On</td>
<td>710</td>
</tr>
<tr>
<td>0.8</td>
<td>On</td>
<td>810</td>
</tr>
<tr>
<td>0.9</td>
<td>On</td>
<td>910</td>
</tr>
<tr>
<td>1.0</td>
<td>On</td>
<td>1010</td>
</tr>
<tr>
<td>1.1</td>
<td>On</td>
<td>1110</td>
</tr>
<tr>
<td>1.2</td>
<td>On</td>
<td>1210</td>
</tr>
<tr>
<td>1.3</td>
<td>On</td>
<td>1310</td>
</tr>
<tr>
<td>1.4</td>
<td>On</td>
<td>1410</td>
</tr>
<tr>
<td>1.5</td>
<td>On</td>
<td>1510</td>
</tr>
<tr>
<td>1.6</td>
<td>On</td>
<td>1610</td>
</tr>
<tr>
<td>1.7</td>
<td>On</td>
<td>1710</td>
</tr>
<tr>
<td>1.8</td>
<td>On</td>
<td>1810</td>
</tr>
<tr>
<td>1.9</td>
<td>On</td>
<td>1910</td>
</tr>
<tr>
<td>2.0</td>
<td>On</td>
<td>2010</td>
</tr>
<tr>
<td>2.1</td>
<td>On</td>
<td>2110</td>
</tr>
<tr>
<td>2.2</td>
<td>On</td>
<td>2210</td>
</tr>
<tr>
<td>2.3</td>
<td>On</td>
<td>2310</td>
</tr>
<tr>
<td>2.4</td>
<td>On</td>
<td>2410</td>
</tr>
<tr>
<td>2.5</td>
<td>On</td>
<td>2510</td>
</tr>
<tr>
<td>2.6</td>
<td>On</td>
<td>2610</td>
</tr>
<tr>
<td>2.7</td>
<td>On</td>
<td>2710</td>
</tr>
<tr>
<td>2.8</td>
<td>On</td>
<td>2810</td>
</tr>
<tr>
<td>2.9</td>
<td>On</td>
<td>2910</td>
</tr>
<tr>
<td>3.0</td>
<td>On</td>
<td>3010</td>
</tr>
<tr>
<td>3.1</td>
<td>On</td>
<td>3110</td>
</tr>
<tr>
<td>3.2</td>
<td>On</td>
<td>3210</td>
</tr>
<tr>
<td>3.3</td>
<td>On</td>
<td>3310</td>
</tr>
<tr>
<td>3.4</td>
<td>On</td>
<td>3410</td>
</tr>
<tr>
<td>3.5</td>
<td>On</td>
<td>3510</td>
</tr>
<tr>
<td>3.6</td>
<td>On</td>
<td>3610</td>
</tr>
<tr>
<td>3.7</td>
<td>On</td>
<td>3710</td>
</tr>
<tr>
<td>3.8</td>
<td>On</td>
<td>3810</td>
</tr>
<tr>
<td>3.9</td>
<td>On</td>
<td>3910</td>
</tr>
<tr>
<td>4.0</td>
<td>On</td>
<td>4010</td>
</tr>
<tr>
<td>4.1</td>
<td>On</td>
<td>4110</td>
</tr>
<tr>
<td>4.2</td>
<td>On</td>
<td>4210</td>
</tr>
<tr>
<td>4.3</td>
<td>On</td>
<td>4310</td>
</tr>
<tr>
<td>4.4</td>
<td>On</td>
<td>4410</td>
</tr>
<tr>
<td>4.5</td>
<td>On</td>
<td>4510</td>
</tr>
<tr>
<td>4.6</td>
<td>On</td>
<td>4610</td>
</tr>
<tr>
<td>4.7</td>
<td>On</td>
<td>4710</td>
</tr>
<tr>
<td>4.8</td>
<td>On</td>
<td>4810</td>
</tr>
<tr>
<td>4.9</td>
<td>On</td>
<td>4910</td>
</tr>
<tr>
<td>5.0</td>
<td>On</td>
<td>5010</td>
</tr>
<tr>
<td>5.1</td>
<td>On</td>
<td>5110</td>
</tr>
<tr>
<td>5.2</td>
<td>On</td>
<td>5210</td>
</tr>
<tr>
<td>5.3</td>
<td>On</td>
<td>5310</td>
</tr>
<tr>
<td>5.4</td>
<td>On</td>
<td>5410</td>
</tr>
<tr>
<td>5.5</td>
<td>On</td>
<td>5510</td>
</tr>
<tr>
<td>5.6</td>
<td>On</td>
<td>5610</td>
</tr>
<tr>
<td>5.7</td>
<td>On</td>
<td>5710</td>
</tr>
<tr>
<td>5.8</td>
<td>On</td>
<td>5810</td>
</tr>
<tr>
<td>5.9</td>
<td>On</td>
<td>5910</td>
</tr>
<tr>
<td>6.0</td>
<td>On</td>
<td>6010</td>
</tr>
<tr>
<td>6.1</td>
<td>On</td>
<td>6110</td>
</tr>
<tr>
<td>6.2</td>
<td>On</td>
<td>6210</td>
</tr>
<tr>
<td>6.3</td>
<td>On</td>
<td>6310</td>
</tr>
<tr>
<td>6.4</td>
<td>On</td>
<td>6410</td>
</tr>
<tr>
<td>6.5</td>
<td>On</td>
<td>6510</td>
</tr>
<tr>
<td>6.6</td>
<td>On</td>
<td>6610</td>
</tr>
<tr>
<td>6.7</td>
<td>On</td>
<td>6710</td>
</tr>
<tr>
<td>6.8</td>
<td>On</td>
<td>6810</td>
</tr>
<tr>
<td>6.9</td>
<td>On</td>
<td>6910</td>
</tr>
<tr>
<td>7.0</td>
<td>On</td>
<td>7010</td>
</tr>
<tr>
<td>7.1</td>
<td>On</td>
<td>7110</td>
</tr>
<tr>
<td>7.2</td>
<td>On</td>
<td>7210</td>
</tr>
<tr>
<td>7.3</td>
<td>On</td>
<td>7310</td>
</tr>
<tr>
<td>7.4</td>
<td>On</td>
<td>7410</td>
</tr>
<tr>
<td>7.5</td>
<td>On</td>
<td>7510</td>
</tr>
<tr>
<td>7.6</td>
<td>On</td>
<td>7610</td>
</tr>
<tr>
<td>7.7</td>
<td>On</td>
<td>7710</td>
</tr>
<tr>
<td>7.8</td>
<td>On</td>
<td>7810</td>
</tr>
<tr>
<td>7.9</td>
<td>On</td>
<td>7910</td>
</tr>
<tr>
<td>8.0</td>
<td>On</td>
<td>8010</td>
</tr>
<tr>
<td>8.1</td>
<td>On</td>
<td>8110</td>
</tr>
<tr>
<td>8.2</td>
<td>On</td>
<td>8210</td>
</tr>
<tr>
<td>8.3</td>
<td>On</td>
<td>8310</td>
</tr>
<tr>
<td>8.4</td>
<td>On</td>
<td>8410</td>
</tr>
<tr>
<td>8.5</td>
<td>On</td>
<td>8510</td>
</tr>
<tr>
<td>8.6</td>
<td>On</td>
<td>8610</td>
</tr>
<tr>
<td>8.7</td>
<td>On</td>
<td>8710</td>
</tr>
<tr>
<td>8.8</td>
<td>On</td>
<td>8810</td>
</tr>
<tr>
<td>8.9</td>
<td>On</td>
<td>8910</td>
</tr>
<tr>
<td>9.0</td>
<td>On</td>
<td>9010</td>
</tr>
<tr>
<td>9.1</td>
<td>On</td>
<td>9110</td>
</tr>
<tr>
<td>9.2</td>
<td>On</td>
<td>9210</td>
</tr>
<tr>
<td>9.3</td>
<td>On</td>
<td>9310</td>
</tr>
<tr>
<td>9.4</td>
<td>On</td>
<td>9410</td>
</tr>
<tr>
<td>9.5</td>
<td>On</td>
<td>9510</td>
</tr>
<tr>
<td>9.6</td>
<td>On</td>
<td>9610</td>
</tr>
<tr>
<td>9.7</td>
<td>On</td>
<td>9710</td>
</tr>
<tr>
<td>9.8</td>
<td>On</td>
<td>9810</td>
</tr>
<tr>
<td>9.9</td>
<td>On</td>
<td>9910</td>
</tr>
</tbody>
</table>

Fig. 14. Temporal values (in ms) for user interaction using Fitts law (second converter)

Fig. 15. System model extended to incorporate physical time values
In order to be able to calculate time movement using Fitts Law, we have to know where the movement starts and where it finishes. This is the reason why we have added transitions for representing all the possible movement on the currency converter as time, for instance, is different when pressing button 1 immediately after pressing button 9 or 2 (the figures are shown in Fig. ). For instance time needed for pressing button 1 after button Franc is 283ms. For reading the value from table in Fig., first find the “from” button in first row (here button 1) then read value corresponding to the “to” button (here button Franc). It is interesting to note that the order in which buttons are pressed is of importance value for first Franc the button 1 is 319ms). This is due to the different size of the physical buttons (see Fig. ).

However, as some buttons are of identical size, simplifications could be made. For instance as buttons 2 and 4 are similar, in Petri net model in Fig. transition Four to Two and Two to Four could be merged, but in order to keep models consistent and more understandable we keep them distinct. This is an architectural consideration making models independent from temporal values. This is mandatory for software user interfaces as presentation part is subject to frequent modifications during development.

**Cognitive Time.** Fig., presents the Petri net model for users cognitive activity embedding temporal information. As for system model important information is associated to transitions. The model is very simple as cognitive activity is quite limited for using the devices. However, more complex models could be considered for modelling information that must be remembered as in [0]. Temporal information in the Petri net presented in Fig. comes from work presented in [0]. In this work mean cycle time for cognitive activity is set at 100ms [25~170ms] and this temporal value is associated to transition Generic认知活动. Another parameter that is also taken into account in the model is the comparison of values. Russo [0] estimates that time needed for comparing two numbers is about 33ms. In this paper, the exact temporal values are not very important as the aim of performance evaluation is to compare two existing devices. In the development process of those systems the accuracy of values is usually much more important as it might drive design decisions.

As stated above, some transitions in a cognitive model can be of type Time-out. For instance, as introduced in [0] due to memory overload some information might be lost in short term memory. This would be modeled (in a user cognitive model) using a Time-out transition automatically removing information from the place representing information in short term memory.

![Diagram](image_url)

**Fig. 16.** On the left, cognitive model. On the right, perceptual model

341
Perceptual Time. As for cognitive activities, temporal information for perceptual activity is added to a dedicated Petri net model (see Fig. ). Here again temporal value comes from work described in [0] and is associated to the transition. Average time needed for the eye to capture information is about 100ms but is subject to users variability [50~200ms]

System Time. The work presented here does not take into account time related to system execution. Indeed for such systems we can consider that the with respect to users time frame system time is not significant. This is a more general problem of including time in models and both simulating and computing performance evaluation on those models. However, in classical interactive systems such as the ones we are considering in the project, the system is distributed and time plays a significant role with respect to users interactions. ODonnell, in [0], shows that user’s strategy depends on machine’s response time. In order to deal with these temporal issues we use the temporal policy proposed in generalised stochastic Petri nets [0]. Another advantage of this approach is that it allows for modelling temporal unpredictability in distributed systems by using the stochastic capabilities of this Petri net model.

4.3 Exploiting Temporal Information

Temporal information that has been added to models is exploited through scenarios. Indeed, each action appearing in a scenario can be related to a transition is one of the models.

<table>
<thead>
<tr>
<th>Scenario 1</th>
<th>Type of transition</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verify state converter</td>
<td>Cognitive</td>
<td>100 ms</td>
</tr>
<tr>
<td>Press on</td>
<td>Physical</td>
<td>291 ms</td>
</tr>
<tr>
<td>Read_10 francs</td>
<td>Perceptual</td>
<td>100 ms</td>
</tr>
<tr>
<td>Press_1</td>
<td>Physical</td>
<td>244 ms</td>
</tr>
<tr>
<td>Verify value</td>
<td>Cognitive</td>
<td>100 ms</td>
</tr>
<tr>
<td>Press_2</td>
<td>Physical</td>
<td>381 ms</td>
</tr>
<tr>
<td>Verify value_1</td>
<td>Cognitive</td>
<td>100 ms</td>
</tr>
<tr>
<td>Choose key to press</td>
<td>Cognitive</td>
<td>100 ms</td>
</tr>
<tr>
<td>Press E</td>
<td>Physical</td>
<td>353 ms</td>
</tr>
<tr>
<td>Read value</td>
<td>Perceptual</td>
<td>100 ms</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>****</td>
<td><strong>1599 ms</strong></td>
</tr>
</tbody>
</table>

Fig. 17. Table describing effective time for each action or transition (First scenario)

Fig. describes the mean time needed by a user for performing the selected scenario using the first device. Fig. presents the same information for the second device.

It can be easily seen that there is not a huge difference in performance of the selected scenario between both devices. Other scenarios might have been impossible to perform (for instance convert 10 francs into British pound) with second device but easily performed with the first ones. Scenarios including more
complex activities such as giving change back in a different currency can only be performed using second converter.

This is why we have been using task models as a basis for scenarios description. The idea is to extract interactively scenarios from task models and to use temporal extensions added to system, cognitive and perceptual models to perform, in an automatic way, performance evaluation. This phase requires a bridge between task model and system model as presented in [0] and the extension of PetShop in order to process models extended with temporal information.

5 Tool Support for Performance Evaluation

PetShop is a case tool dedicated to the edition, analysis and prototyping of event-driven highly interactive applications. For more information about the tool and its features see [0] and [0].

Petshop is currently extended in order to:

- allow designers to add temporal information to a system model,
- use temporal information to compute performance evaluation of a system model,
- offer analysis tools for checking consistency between the “basic” system model and the system model with additional temporal information.

At this stage the system is On (one token in place On). Current value of the token can be monitored in the window on the left hand side of Fig. . This window shows the current values of the three variables of the token in place On: $v = 10$, $n = 2$, $l = 0$. This means that the value entered by the user is 10, the number of digit is 2 and last button pressed is button 0.

Fig. presents the time augmented system model augmented with time. Temporal information is related to transitions and the window in the right-hand corner shows the content of transition Two to three. Temporal information in the action part of the transition ($tt = t + 219$) corresponding to the fact that pressing button 3 after button 2 adds 219 ms to time spent in the performing the scenario.

![Table desribing effective time for each action or transition](image)
6 Conclusion

Work presented in this paper deals with performance evaluation techniques for interactive systems evaluation. We have presented a generic framework for adding temporal information to models and for relating tasks models to system models through scenarios. This work has been presented through a simple case
study allowing us to show how such temporal information can be added in a systematic way and how it can be used for assessing designs.

The framework proposed is currently used more thoroughly by taking into account more complex users cognitive and perceptual models. For instance we take into account the impact of practice using stochastic time. Such information is extracted from [0] and called power law of practice in which time $T_n$ to perform a task on the $n^{th}$ trial is $T_n = T_1 n^{-r}$, where $r = 0.4$ [2 ~ 6].

Similarly, time needed for a user to choose between different alternatives is not constant and increases with the judgement or decision to be made function of probabilities of different alternatives. Decision time $T$ increases with uncertainty about the judgments or decision to be made as described in [0]. We are currently working on integrating these temporal elements in PetShop in order to integrate performance evaluation as a interactive assessment technique for helping designers to rationally and quantitatively select design options.

7 Acknowledgements

The work presented here is fully funded by French defence agency (Direction Générale pour l'Armement) under contract #00.70.624.00.470.75.96. Special thanks to Didier Bazalgette for precise information about the field of command and control systems in military field.

References


345
12. Miller, George A. “The Magic Number Seven Plus or Minus Two.”, 1956, 63, pp. 81-97.
Blending descriptive and numeric analysis in human reliability design

Shamus P. Smith and Michael D. Harrison

The Dependability Interdisciplinary Research Collaboration
Department of Computer Science, The University of York
York YO10 5DD, United Kingdom
{Shamus Smith, Michael Harrison}@cs.york.ac.uk

Abstract. Scenario based design allows for the early elicitation of requirements and can be helpful in the design phase of system development. It is typical for cycles of iteration to be used to refine a design so that it more closely meets its requirements. Such refinements are in terms of the original requirements specification and any new requirements that have been identified. However, not all defined requirements are equally essential. Although descriptive methods for scenario analysis can be used to highlight new requirements, it can be difficult to evaluate the impact of these new requirements.

In this paper, we exemplify this problem and investigate how numeric methods can be used to highlight the impact of consequences identified by descriptive scenario analysis. An example from the context of human reliability analysis is presented.

1 Introduction

Iterative methods are common in the development of computer systems. Within software system life-cycles, such methods are used to refine the current state of the design of a system so that it more closely meets its requirements. This is particularly evident in the design and prototyping phases of a system’s development. In this paper we are concerned with the design phase and with the step preceding a new iteration being applied to a design. By design iteration we mean the process of applying newly identified or refined requirements to an existing design. In the literature there are many candidate methods for design analysis (for examples see [15]) that provide mechanisms for identifying the requirements for alternative and/or new designs. These methods need to provide two components: the new requirements and the rationale for the application of the new requirements.

Typically, design analysis techniques evaluate the current design and look for problems (for example usability issues [8]). The identified problems are used to construct recommendations to solve design problems. However, before these recommendations can be considered and applied as new requirements, it is necessary for some rationale to be provided. Therefore after the application of a design analysis technique, there are three issues that need to be addressed:
1. Does the analysis technique provide a structure, for the identified problems, to allow the designer to successfully argue/defend the recommendations for redesign?

2. Which of these recommendations are most important, i.e. preferred requirements, for the new design?

3. Is it possible to optimise the work effort for the redesign by using the two issues above to focus on the critical new requirements?

These three issues are at the core of the work that is discussed in this paper. We are particularly interested in the arguments that can be developed through this process to support the identification of new requirements and to focus the redesign process.

In this paper we investigate a simplified example in the domain of designing for human reliability. However, we consider that the issues involved are scalable to larger problems and over other domains. The format for the remainder of this paper is as follows. Section 2 introduces the work described in this paper in the context of system design and the application of new requirements. Next we define the example context, human reliability analysis (HRA), and the design analysis technique we have applied (Technique for Human Error Assessment - THEA [10]) including a small example. This will be followed, in Section 4, by a discussion of the descriptive arguments that can be developed from the THEA analysis. A treatment of statement belief, expert judgement and requirement impact will then be presented. Section 5 will introduce a numeric approach (Human Error Assessment and Reduction Technique - HEART [17]) and demonstrate its application to the issues raised by the THEA analysis. In Section 6, we investigate how the descriptive and numeric approaches we have considered can be generalised and briefly describe their application in an alternative domain. The paper concludes with a summary of conclusions.

2 System design and new requirements

In this paper we are interested in the use of scenario based design, where cycles of scenario descriptions can be used to refine the current state of a system design so that it more closely meets its requirements and potential faults are eliminated in this context. We have been investigating the choice of requirements to integrate into a design before the next iterative cycle of design evaluation is attempted.

At this stage of the design process problems leading to faults will have been identified. These may be viewed as new requirements for a redesign. However, not all requirements are equal and whether these requirements must be addressed first is a matter for expert judgement. Some requirements may be considered essential to any redesign and others may have only minor consequences for the development [13]. It is common for the consideration of these issues to be approached in an ad hoc manner. For example, the use of problem clusters, as part of change analysis, to identify change needs [16, pg 96]. This is problematic as traceability for the design decisions can be lost in the subjective nature of such techniques.
In this paper we explore how quantification of new design requirements may be used to reinforce descriptive scenario analysis and will discuss its limitations. In order to have a demonstrable example, we have scoped our work to one area of interest for system dependability: human reliability analysis (HRA). One part of the design phase where human reliability analysis is of interest is where analysis is applied so that new design requirements can be identified. When functional deficiencies are found in a design, their correction leads directly from the design specification. However, a design that is complete in terms of functionality may still have problems in terms of human reliability. Human actions are inherently more difficult to regulate and predict than the functions of the technological components [6]. What we are interested in is identifying how vulnerable a system is.

3 Human Reliability Analysis

Kirwan [7] observes that one of the primary goals of human reliability analysis is to provide a means of properly assessing the risks attributable to human error and for identifying ways of reducing system vulnerability to human error impact. He notes that this is achieved through three processes:

- Identifying what errors can occur (Human Error Identification)
- Deciding how likely the errors are to occur (Human Error Quantification)
- Enhancing human reliability by reducing this error likelihood (Human Error Reduction)

We have investigated how descriptive methods for human error identification can be augmented by techniques that generate numeric values for human error quantification. The aim is to enhance human reliability through the reduction of human errors that these processes facilitate. In Section 5 we investigate a technique for human error quantification but before this can happen, we need to explore the process of human error identification. For this we are using a scenario based technique developed at The University of York called THEA [3,10].

3.1 THEA

THEA (Technique for Human Error Assessment) [3,10] is a technique developed to help designers of interactive systems to anticipate interaction failures or human errors that may be problematic once their designs become operational. The technique is intended for use early in the development life-cycle, as design concepts and requirements concerned with safety and usability, as well as functionality, are emerging [3]. Fields, Harrison and Wright [3] note that errors in human reliability can be regarded as failures in cognitive processing. They present an outline of a variant of Norman’s [9] execution-evaluation model of human information processing (see Figure 1).

Five components from Norman’s cyclic model are used as the basis for identifying ways in which human information can potentially fail. THEA consists of
a checklist of questions about the performance of each of the cognitive components in relation to the use of the system with the aim of anticipating where cognitive failures might occur which lead to behavioural errors. These questions are applied to a scenario in order to help uncover places in the scenario where cognitive failure modes may occur. The steps used in this process are shown in Figure 2 (from [3]).

Pocock et al. [10] observe that “THEA explicitly takes contextual and cultural issues into consideration by means of usage scenarios. In this way it is hoped to elicit the way work is actually practised and not simply how designers envisage it as being practised.” This scenario oriented approach distinguishes THEA from techniques that begin by generating error producing scenarios from models of users and domains (for example [4]). Due to space constraint, full coverage of THEA is not possible and the reader is directed to [3,10]. A THEA example for this paper will be presented in Section 3.3 but before we can apply this technique, we must specify the domain, and more specifically the scenario where this analysis is focused. Although the example we present is based in the physical world, the principles of the treatment apply over a broad range to environments including software systems.

3.2 The domain and scenario

The problem domain is in safety dependency issues on-board an oil tanker. In the scenario we are considering, the vessel has entered into some stormy conditions
and the crew are preparing a lifeboat in case there is a need to abandon the vessel. Although the lifeboat can be launched from the ship's bridge, it must be manually primed beforehand. This involves a crew-member going on deck with a key to unlock the storage constraints/handles on the lifeboat. The key used in this task is also the key that is used to initiate the lifeboat launch from the ship's bridge.

### 3.3 Fragment of the THEA analysis

Part of the THEA analysis from the scenario described in Section 3.2 can be seen in Figure 3. A THEA analysis is roughly split into eight components. Four sets of rows associated with the four cognitive failure areas discussed in Section 3.1 and four columns that make up the structure of THEA. These consist of the checklist questions, the causal issues, the consequences and any design issues. The questions column is the same for every THEA analysis and the other spaces are completed by the domain expert doing the analysis. For our example, we have only included a selection of questions taken from each of the four failure areas. In a full analysis, there are four goals questions, four plans questions, four actions questions and eight perception, interpretation and evaluation questions.

### 4 Qualifying the issues

Analysis of a proposed design provides a set of identified issues/problems with the design. This may also be accompanied with some suggested solutions to deal with the identified issues. These recommendations have differing force as requirements for any redesign. Also there may be varying degrees of rigour in the specification of these recommendations that may affect any justification of their use as requirements. This rigour may influence the nature of the argument that can be constructed from the recommendations to support their implementation. For each answered question, THEA produces a triple structure from design analysis comprising of the causal issues, associated consequences and suggested design issues. These triples can be presented as binary statements (either present or absent). Although the output of the THEA process is a set of recommended design decisions this is not an argument for design. THEA provides subjective declarations of possible design needs with an associated rationale that may be subject to external scrutiny. This can be problematic as the statements are constructed by experts and are summary statements and as a result bias can be introduced into the description. Also the level of detail that is described and the number of alternative solutions proposed is determined on a case by case basis at the expert's discretion.

Analysis of THEA, as a descriptive approach, raises two main issues. Firstly, what level of belief can be associated with the expert judgement and the structure provided by the technique? Secondly, how can we decide which recommendations are most important in the context of a redesign, i.e. what is the impact on the redesign process? These two issues will be considered in Sections 4.1 and 4.2 respectively.
### Questions

<table>
<thead>
<tr>
<th>G1: (Is the task triggered by stimuli in the interface, the environment or the task itself?)</th>
<th>G3: (Can a goal be achieved without all its 'subgoals' being correctly achieved?)</th>
<th>G4a: (Can a goal be achieved without all its 'subgoals' being correctly achieved?)</th>
<th>G4b: (Can a goal be achieved without all its 'subgoals' being correctly achieved?)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No, on command of the Captain.</td>
<td>The Captain may fail to trigger the crew-member to remove the handles on the lifeboat.</td>
<td>Some form of interlock should exist to prevent lifeboat launch if handles are not removed.</td>
<td></td>
</tr>
</tbody>
</table>

### Causal Issues

| No, crew-member must return key to the Captain before the lifeboat can be initialised. | If the crew-member is disabled deck-side, the lifeboat cannot be initialised. | Alternative method of initialisation. |

### Consequences

| Yes, crew training. | Crew-member may lose the key. | Secure key to crew-member. |

### Design Issues

### PLANS

### ACTIONS

Yes, crew-member must ensure they have key with them while on deck.

Potential danger from:
1. Bad weather
2. Premature attempt to launch lifeboat

Some means whereby crew-member must remove key from bridge before going on deck.

### PERCEPTION, INTERPRETATION & EVALUATION

Yes, crew-member can see the removed handles from the lifeboat.

- -

---

**Fig. 3. THEA Fragment**

### 4.1 Belief, expert judgement and structure

There is evidence of the questionable value of expert judgement [1] and the associated belief in those judgements. Hollnagel [5] notes that expert judgements are an uncertain and imprecise source of information. What THEA does provide is a structure for documenting the issues that have been elicited via the analysis. On an individual THEA question basis, the triples in the structure could be used to build an argument for the identified issue. For example, a dialectical argument (see Figure 4) could be constructed for the design issue “An alternative initialisation mechanism is needed” from entry G4a in Figure 3.

THEA does not provide a mechanism to show the dependence of individual statements. Therefore, if the statements are changed at a local level in the THEA structure, any global consequences are not necessarily evident. The filtering of any associated changes in the other THEA entries is completely at the whim of the expert and their knowledge of these consequences. Inconsistencies in the
issue, consequence and design issue statements could easily be introduced that would then cast doubt on the validity of any associated design based arguments. Also there is no weighting on any of the statements in such arguments. Therefore all statements may be considered to be of equal importance. This is clearly not the case for most sets of requirements.

![Diagram](image)

**Fig. 4. Example dialectical argument from a THEA design issue (G4a)**

### 4.2 Design issue impact

When there is a large set of requirements the challenge of determining the most appropriate requirements for a redesign is difficult. If a technique like THEA is used, its output is a set of design recommendations/issues. As noted by Pocock et al. [10] the design issues identified by THEA are “intended to assist designers reason about about errors at the early stages of a design before it becomes impractical or prohibitively expensive to effect a longer term design change or implement shorter term procedural ‘fixes’ or limitations.”

Unfortunately, there is typically no indication of the worth of the design issues to any redesign. This worth must be determined by the reader of the recommendations. This can easily lead to an ad hoc approach to redesign and the selection of new requirements for a redesign. However, one approach to measuring worth is to examine the impact of applying candidate recommendations. Dearden and Harrison [2, pg 161] provide an informal definition of impact as:

> the effect that an action or sequence of actions has on the safe and successful operation of a system.

Although THEA based analysis identifies the consequences of actions, it does not explicitly identify the impact on the new design. A notion of impact provides a means of discussing the consequences of possible human-errors, independent of the probability of these errors occurring within a particular design [2]. Not considering impact can be problematic when there are limited resources available.
If there is only a limited budget, in terms of time, person power, money etc., then it may not be possible to implement all the recommendations. Also if new requirements are in conflict, which of them should be given priority when conflict resolution is attempted? Unfortunately, the use of ad hoc criteria to select new requirements may lead to the absence of possibly essential recommendations, in the analysis expert’s judgement, not being selected. Ideally, it would benefit the redesign process to be able to get an objective view of the design recommendations. Hopefully, this would allow informed decisions to be made in terms of the scope of the design and the consequences for the system. The use of a numeric approach to this problem will be considered in the next section.

5 Quantifying issue importance

In this section the ranking of descriptive design recommendations provided by THEA is discussed through one candidate approach for HRA probability generation, namely HEART (Human Error Assessment and Reduction Technique) [7, 17]. Our aim is to investigate the addition of numerical precision to reinforce or highlight a notion of impact for the new requirements.

5.1 HEART overview

HEART [17] is a quick technique for the quantification of human reliability. It is based on a review, by its author, of both literature on human factors and of experimental evidence showing the effects of various parameters on human performance. The technique defines a set of generic human error probabilities (HEPs) for different types of tasks. These are used as the starting point for HEART quantification. After a task has been classified, an analyst then determines whether any error-producing conditions (EPCs) are evident in the scenario under consideration. For each error-producing condition, the generic human error probabilities are multiplied by the error-producing condition which increases the human error probability. An example of HEART will be described in Section 5.2. The reader is directed to [7] for an overview of HEART and other techniques for human reliability analysis. They will not be discussed in detail here as our motivation for using HEART in our example is threefold. Firstly, HEART is based in human reliability analysis. This considerably eases the task of combining the data from the approaches and simplifies the job of our domain expert. Secondly, both techniques have been developed using the same rationale, that is, to be quickly applied methods to identify the “big” problems in a target domain. Thirdly, in the human reliability analysis community, HEART is readily understandable by all interested parties and is a way of supporting dialogue about human reliability estimates [10].

5.2 HEART application to THEA

The descriptive approach, as discussed in Section 3.3, is suggestive of problems in our design. When applying human reliability analysis to high consequence
systems, such problems are of particular concern. For the example in this paper, we have applied a HEART analysis to the THEA material presented in Figure 3 to refine any justifications for the redesign recommendations. In the scenario analysis four design issues were identified by the descriptive method, namely:

1. Interlock to prevent the launch of the lifeboat if the handles are not removed (G1 from Figure 3)
2. Alternative method of initialisation (G4a from Figure 3)
3. Secure the key to crew-member (G4b from Figure 3)
4. Some means whereby crew-member must remove key from bridge before going on deck (A3 from Figure 3)

The scenario is one of pre-evaluation preparation and the task is to remove the locks from the lifeboat. As a starting point for HEART analysis, the expert has identified that this task is one that is a routine, highly practised, rapid task involving relatively low levels of skill. HEART provides sets of generic categories [17] and for this example a starting value of human unreliability of 0.02 is defined. The next stage of a HEART analysis is the identification of error-producing conditions (EPCs) that are evident in the scenario and would have a negative influence on human performance. The domain expert examined the causal and consequence issues defined in the THEA analysis and identified three HEART EPCs that range over the design issues.

1. Little or no independent checking or testing of output
2. No obvious way to keep track of process during an activity
3. Hostile environment

Each EPC has an associated value (from [17]) that indicates the predicted effect and the extent to which unreliability will change due to the EPC. However, there are different levels of significance for any EPC. As the EPC values are general, an assessed proportion of effect significance is used to customise the specific relevance to the unreliability calculation. In this example, the domain expert has determined that the crew are highly trained and this includes training for bad weather. Hence, low significance values have been associated with the EPC effects. A summary of these values\(^1\) can be seen in Figure 5.

<table>
<thead>
<tr>
<th>Error Producing Condition</th>
<th>HEART effect</th>
<th>Assessed significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Little or no independent checking or testing of output</td>
<td>*3</td>
<td>0.12</td>
</tr>
<tr>
<td>No obvious way to keep track of process during an activity</td>
<td>*1.4</td>
<td>0.25</td>
</tr>
<tr>
<td>Hostile environment</td>
<td>*1.15</td>
<td>0.12</td>
</tr>
</tbody>
</table>

\(^1\) The HEART EPC effects are determined from [17] while the significance values were assessed by the domain expert.
HEART provides a formula to determine the final assessed effect for an error-producing condition:

\[ ((\text{HEART effect} - 1) \times \text{assessed significance}) + 1 = \text{final EPC assessed effect} \]

The nominal likelihood of failure for a scenario based task is calculated by the product of the initial starting value of human unreliability (in our example, as noted above, 0.02) and the assessed effects of the relevant\(^2\) error producing conditions. For example, take the design issue G1 from Figure 3 “Interlock to prevent the launch of the lifeboat if the handles are not removed”. This issue is associated with two of the identified EPCs, firstly that as it is a solo self verified task there is little or no independent checking or testing of output (e.g. the Captain can forget to tell the crew-member to remove the handles) and secondly, the task is taking place in a hostile, stormy, environment. The product of assessed effects\(^3\) of these two EPCs and the starting value of reliability is:

\[ (1.24 \times 1.018 \times 0.02) = 0.025246 \text{ or } 2.52\% \]

This number is an absolute probability and is not of interest to us as such. However, the application of the HEART analysis to all the design issues provides us with a ranking for the problems identified by the descriptive method. The ranked design issues can be seen in Figure 6.

5.3 Numeric impact

The priority listing can be used as part of the justification process for arguing about implementing redesign options in terms of maximising the reduction of negative consequences in a scenario. Hence, the ranking can be seen as a numeric measure for consequence impact.

From our scenario, having the crew-member remove the key from the bridge before going on deck is a critical action in terms of the task we have examined. A number of dangerous consequences can result if this is not carried out, for example the lifeboat may be launched while the handles are still in place. This would have a number of dangerous results including damage to the ship, the lifeboat and possibly a crew-member who was on deck. As the example is only a fragment of a full THEA analysis, the analysis of this action, and the resulting consequences if it is not carried out may have been lost in the documentation that results from a full descriptive analysis. The HEART analysis identifies this as the most important design issue in the sense that it is the most likely where human reliability is a contributing factor. What the HEART analysis provides is

---

\(^2\) Not all EPCs, and their associated assessed effects, are necessarily applicable for all design issues. For instance, in the following example only two EPCs are used to calculate the probability of failure.

\(^3\) Each EPC assessed effect was calculated via the HEART formula \((\text{HEART effect} - 1) \times \text{assessed significance}) + 1\) from [17].

356
an indication of likelihood on the design problems but not any level of severity. Expert judgement is required to define any tradeoffs between the likelihood of a consequence (as indicated by the HEART analysis) and the severity impact the consequence has in the scenario context. However, the numeric method has refined the process of recommending design issues to allow the designers to make better, informed, decisions in the context of the redesign of the system.

<table>
<thead>
<tr>
<th>Design issue</th>
<th>Likelihood of failure (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3: Crew-member must remove key from bridge before going on deck</td>
<td>2.78</td>
</tr>
<tr>
<td>G1: Interlock to prevent the launch if the handles are not removed</td>
<td>2.52</td>
</tr>
<tr>
<td>G4a: Alternative method of initialization</td>
<td>2.24</td>
</tr>
<tr>
<td>G4b: Secure key to crew-member</td>
<td>2.04</td>
</tr>
</tbody>
</table>

**Fig. 6.** Design issue rankings based on the HEART analysis

### 6 Discussion

In this paper, we have demonstrated how a descriptive analysis can be augmented by the use of a numeric technique to refine a measure of impact for identified problems in a design. In an example, numeric probabilities of human reliability are used to determine the significance of the descriptive recommendations. While we have been using THEA and HEART as examples of descriptive and numeric methods, to a certain degree, the use of these methods is arbitrary. There is a host of alternative methods that could be substituted in the current examples. What is of more importance is the process that is involved and the role that probabilities play. In our examples we have investigated the development/evaluation of new requirements for a design. This has involved two levels of refinement of the issues identified from example scenarios.

The first refinement is after the initial application of the descriptive method. Initially, the descriptive method identifies areas of concern, and/or issues that require attention, in the current design. The severity of these issues can be identified by impact analysis on the consequence of such issues. Although, as noted in Section 4.1, issues are commonly defined as being of concern without differentiation, in practice, expert judgement would be used to rate the severity of the issues, at least informally, based on the consequences of the issues. This informality, and therefore lack of credibility, is what we wish to reduce by the application of a numeric technique.

The second level of impact refinement is in the use of a numeric method to indicate probabilities that can be used to highlight the severity of a design deficiency. In our example we have been using HEART to identify the likelihood of failure due to human unreliability. As we are interested in dependable systems,
the level of reliability can be used as an assessment criterion, e.g. if some component is very unreliable then this could be seen as having a higher justification for redesign.

The process we have described is focused on the initial use of a descriptive method and later refinement with a numeric method to provide quantitative precision. There are several advantages to this approach. Numeric methods commonly require specialised knowledge for their application. However, descriptive methods, by their informal nature, can be used by a larger user base. For example, THEA has been developed so that the causal model of human error and the questionnaire can guide a non human factors expert with domain expertise. Secondly, the application of numeric methods is notoriously time consuming and error prone. The use of descriptive methods to rank identified issues can reduce the need to apply numeric methods over all cases. Thirdly, methods that use generic probabilities or weighting factors (e.g. error producing conditions) are difficult to apply without bias. Changes in the generic variables chosen [2, pg 162], and changes in the environment or human performance data, can drastically alter the final probabilities. Numeric precision can easily be lost through inaccuracy. Although descriptive methods may not provide exact values, the ranking of issues may be all that is required.

Another advantage of the initial use of descriptive methods is the focus of work in context. The use of scenario based descriptive methods allows the elicitation of problem issues to be identified in the actual work context. This can be particularly useful if domain experts (with limited time commitment) are to be involved in the early stages of the development of a causal model.

An alternative approach is that presented by Galliers, Sutcliffe and Minocha [4]. Their approach is built around the construction of a Bayesian Belief Network (BBN) as a means of combining a set of generic influencing factors into a more formal and predictive model of human error. Their method comprises of ten stages:

1. Analyse domain to list key safety critical properties for the domain
2. Select influencing factors and build BBN as a causal model
3. Calibrate BBN and node probability tables
4. Select domain scenarios
5. Measure and estimate scenario variables
6. Input scenario variables into BBN
7. Analyse user’s task
8. Input sub-goal complexity assessment
9. Run BBN model
10. Walkthrough sub-goals with design guidelines

For Gallier et al. [4], the application of BBN affords numeric precision early in the analysis process. As such, they can generate human error probabilities, in terms of slips and mistakes (see [12]), as part of the causal analysis process (Steps 1-9 in the list above). Step 10, the sub-goal walkthrough, combines the error probabilities with descriptive safety critical user interface guidelines to provide
either generic requirement recommendations for new designs or evaluations of existing designs. Galliers et al. [4] summarise this process as “the BBN analysis first highlights particular scenarios where errors may occur due to a certain combination of operators and the task environment, then given a set of tasks, how the BBN predictions of mistake or slip-errors can be interpreted in a walkthrough of the user’s task to consider which safety-critical guidelines might be recruited to the design.”

Although BBNs can be difficult to build from scratch, Gallier et al. propose that generic BBN models can be calibrated and customised for individual domain analysis. This reuse of the BBN structures can considerably reduce the time needed to generate the error probabilities. Unfortunately, the development of the BBN causal model does require a large commitment on the part of the domain experts and users liaising with the BBN developers. Also much of the data for the BBN is derived from historical data, expert knowledge and user experience. These are subjective sources and it can be difficult to determine their accuracy. This is problematic when their input becomes an integral part of a process that promotes numeric precision, i.e. the use of probabilities.

Also one of the advantages of starting with a descriptive approach is that it can reduce the numeric calculations that are required. The output of Galliers et al’s BBN is numeric data that is then reconciled with general design guidelines, task by task, in the context of a design. Although this may provide ample coverage of possible error scenarios, it may require more work on the part of the analyst to relate the results back to actual likely situations.

Where Gallier et al’s process becomes descriptive, in the application of the numeric probabilities via the walkthrough process, the descriptive method we have investigated begins. While their numeric precision is blended through the descriptive analysis, ours is used to reinforce and refine the conclusions that are informally defined in the early stages of consequence analysis. However, the scope of the examples presented in this paper are relatively small and grounded in one domain, human reliability assessment. We propose that the issues raised in our analysis are of a broader appeal to other domains and systems. For example, in terms of generalising these issues, consider an alternative example in the design of a security software system.

A descriptive approach similar to the THEA checklist could be used to elicit issues in terms of security concerns, for example password validation, encryption schemes or access to physical systems. Scenario analysis could then be used to identify the consequences of failure in these issues and possible countermeasures defined as recommendations for a redesign. Ideally, we would expect that the descriptive checklist would identify areas of failure in the current design (for example, over Randell’s classification of errors, faults and failures [11]) and encourage a dialogue for designers to develop problem solutions. Identifying the issues is the first step. Ranking the issues provides us with quantifiable rationale to proceed with a certain redesign solution.

When investigating the ranked security issues in terms of impact (which may identify “high risk” security concerns), a descriptive measure of their contri-
The elicitation of new requirements in an iterative design process is an important part of system development. The construction of arguments to support the rationale for using new requirements is also valuable. In this paper we have examined use of descriptive methods to build initial recommendations for redesign requirements and have investigated the use of numeric methods to enrich the justification for these requirements.

Although we have applied two particular techniques (THEA and HEART) in a single domain (human reliability analysis), we feel that this type of analysis process is applicable over other techniques and domains. The approach supports the redesign process and provides a mechanism to increase developer confidence that the work is progressing in a cost, e.g. time or effort, effective way.

However, the work presented in this paper is only an initial step. The numbers that are generated, and the associated descriptive statements, are all subjective measures and are influenced by expert judgement bias. Also any arguments that are developed from this process have only a limited notion of structure and are presented in a non-rigorous form. If an approach is to become useful in industry then a more formal structure and defined blending of the descriptive to the numeric will be required. These are issues that the authors are currently investigating.

8 Acknowledgements

This work was supported in part by the UK EPSRC DiRC project, Grant GR/N13999.
References


KWARESMI — Knowledge-based Web Automated Evaluation with REconfigurable guidelineS optiMIzation

Abdo Beirekdar, Jean Vanderdonckt, Monique Noirhomme-Fraiture

1 Facultés Universitaires Notre-Dame de la Paix, Institut d’Informatique
   rue Grandgagnage, 21 – B-5000 Namur (Belgium)
   {abe,mno}@info.fundp.ac.be
2 Université catholique de Louvain, BCHI
   Place des Doyens, 1 – B-1348 Louvain-la-Neuve (Belgium)
   {beirekdar,vanderdonckt}@isys.ucl.ac.be

Abstract. KWARESMI consists of a language, a method and a tool for expressing, structuring, and organising web usability guidelines towards automated evaluation. Traditional approaches transform natural statements of usability guidelines into lines of code in a tool that parses the HTML code of a web page and performs guidelines review. These approaches are inflexible by nature: impossible to introduce a new guideline, to modify an existing one, to select guidelines on demand before evaluation. Furthermore, there is no optimization of how guidelines can be evaluated in an efficient way: all guidelines are tested on all elements, without factoring out common parts. KWARESMI attempts to address these shortcomings by allowing evaluators to express guidelines in a higher level than simply HTML code and using them for automated usability evaluation on demand. At evaluation-time, the tool examines guidelines to be checked and elements subject to evaluation to optimize the evaluation process by identifying only checkpoints that are relevant to a particular evaluation.

1 Introduction

Usability is today recognized as a major quality and success factor of web sites: with a same or comparable level of utility (i.e., similar functions), usability makes the difference. The user who is confused by one web site may easily switch to another web site. To ensure and measure this quality [12], a wide range of usability evaluation techniques have been proposed and a subset of these are currently in use [11]. They range from formal usability testing to informal usability tests conducted by usability specialists at usability labs or among real users.

All these techniques require usability specialists to conduct them or to analyze evaluation results, which is very resource and time consuming especially for very

1 Historical note: Al-Kwaresmi was an outstanding Arabian mathematician who lived around the tenth century. He translated several Indian works on algebra and his name is the origin of the word "algorithm", which is pronounced Kwaresmia in Arabic.
large, continuously growing web sites. When these conditions cannot be met, some
degree of automation is desired [5,11,17]. A possible solution consists of capturing
the knowledge and experience of usability experts and to express it in form of
recommendations or guidelines to be reviewed and applied by web designers and
developers. Unfortunately, studies carried out that apply these guidelines by designers
reveal that this is difficult to conduct, basically because of the inappropriate way of
structuring or formulating them [14].

For this reason among others, automation has been predominately used to
objectively check guideline conformance or review [11,12]. Many automatic
evaluation tools were developed to assist evaluators with guideline review by
automatically detecting and reporting ergonomic violation (usability, accessibility, etc)
and in some cases making suggestions for fixing them [2,8,9]. Representative
elements of these tools include: A-Prompt [1], WebSat [15], LIFT [5], and Bobby
[6,7]. Most of these tools share the same technique: do a static analysis of the HTML
code of any web page of concern, capture a maximum of useful data, identify targeted
usability problems. The most popular set of guidelines considered for this purpose are
the W3C Web content Accessibility Guidelines [18] and Section508 guidelines [16].

A common shortcoming of the above tools is that the evaluation logic is hard
coded in their evaluation engine, which makes them very inflexible concerning any
modification of the evaluation logic or any introduction of new guidelines. In
addition, many of them do not offer much possibilities of controlling the evaluation
process like choosing which guideline to evaluate, level of evaluation, etc., at
evaluation time. For example, the new version of Bobby offers the single possibility
of choosing which set of guidelines to evaluate: W3C or Section508.

In this paper we introduce a tool that attempts to overcome this shortcoming.
KWARESMI (Knowledge-based Web Automated Evaluation with REconfigurable
guidelineS optiMization) enables evaluators or human factors experts to express the
ergonomic body of knowledge provided by guidelines in terms of constraints of
HTML elements (i.e., tags, attributes). Once coded, this knowledge can be evaluated
dynamically at evaluation-time by configuring the guidelines expressions in an
optimized way depending on the guidelines to be evaluated and the elements
contained in the page. This process consequently factors out checkpoints that are
common across several guidelines, even if they come from different sets of
guidelines. This tool is aimed to extend the limits of existing automatic evaluation
tools and to address some of their other shortcomings.

This paper is structured as follows: Section 2 provides a short description of
evaluation approach followed by some existing evaluation tools. Section 3 precises
some of the KWARESMI requirements and introduces its software architecture. Section
4 describes the evaluation process as supported by this tool. Section 5 addresses
practical issues by focusing on the implementation and the application of the tool.
Section 6 concludes the paper by underlying the potential advantages of our tool.
2 Evaluation Approach in Existing Automatic Evaluation Tools

Historically speaking, the first tools performing static analysis of HTML code were intended to check the validity or conformance of the HTML code with respect to its syntax (see some in [10]). WebLint [4] checks the conformance of HTML code and repairs syntactical problems, such as missing tags, unclosed tags. But it does not perform any usability evaluation, automated or not. One step further, some tools performed a static analysis to verify some predefined rules or metrics. Some confusion exists in this category as some tools claim to perform automated usability evaluation of web sites while they do not assess any form of usability. They only focus on design rules, downloading times, HTML conformance. Although these aspects certainly affect usability, they are not related straightforwardly. One step further, other tools still conduct a static analysis of the HTML code to check usability guidelines for the Web. For example, A-Prompt [1] and Bobby [6,7] both check accessibility guidelines.

However, as mentioned in the introduction, all these existing tools hardcode the evaluation logic inside their evaluation engine. Fig. 1 shows the simplistic software architecture of this approach:

- The evaluation logic is implemented in one or more procedures that will be sequentially called by the evaluation engine during the evaluation process.
- The main user input is the URL of the evaluated page (on-line or locally).
- An evaluation report is generated listing identified usability problems. The clarity and the utility of this report may vary from one tool to another.
- The parsing of the web page is generally independent of the guidelines considered and could cause the page to be parsed more than one time to generate a DOM-like structure of the page.

![Figure 1. Software architecture of existing automatic evaluation tools.](image-url)
2.1 Advantages of the approach

- The implementation of the evaluation procedure is optimal because the developer is totally concentrated on one guideline at a time.
- The evaluation is generally rapid because everything is already provided in the procedures. The evaluation process is efficient since everything is implemented in straightforward procedures and functions code.
- Minimal user intervention which is suitable for non expert users.

2.2 Disadvantages of the approach

- It is impossible to add a new guideline, to modify an existing guideline, or to delete one. Of course, established standards like WAI [18] must not be altered, but other sets of guidelines could be considered or the same set can give rise to different interpretations of included guidelines. These two needs are very important with the continuous evolution of ergonomic experience and web technologies: as new guidelines become available or refined, it would be an advantage to modify them accordingly.
- The capabilities for controlling the evaluation process are very limited, or even non-existent. Guidelines cannot be selected on-demand, possibly from various sets simultaneously, in part or in whole. Not all guidelines are always required to be checked for a page to be evaluated.
- There is not optimization of the guidelines review process, thus leading to redundant assessments. For instance, several WAI guidelines share similar checkpoints for different elements. Moreover, some checkpoints exist which provide a natural way to structure guidelines, interpreted or not, and to optimize the evaluation process.
- Potentially, we could need more than one evaluation tool to cover all the desired guidelines.

3 The KWARESMI Tool

3.1 Requirements

To address the above disadvantages, the tool is supposed to meet the following requirements [3,17]:

- **Be knowledge-based**: it exploits the ergonomic knowledge contained in ergonomic guidelines and re-expresses it in terms of HTML knowledge contained in the semantic of HTML elements.
- **Be Web-oriented**: it works on HTML code of Web pages as their HTML code is accessible, which is not always the case for other user interfaces.
- **Be automated**: it should release evaluators from as many evaluation tasks as possible whether theses procedures can be automated.
- **Be reconfigurable:** to address shortcomings of existing tools, the tool enables evaluators to control both the guidelines inclusion and the evaluation process.

- **Enable guidelines optimization:** it should re-express guidelines in terms of HTML elements to identify potential common ergonomic information among different guidelines. This information should be used to optimize evaluation of these guidelines (explanation follows in section four).

- **Support the GDL formal language:** the development of KWARESMI was triggered by the need for a tool to support the guideline structuring language that we proposed in [3].

  By meeting these requirements, KWARESMI would overcome the shortcomings mentioned above (section 2.2). It is worth noting that KWARESMI is intended to enable the evaluation of any ergonomic guideline (for usability, for accessibility, etc.) as soon as we can find HTML elements that enable the evaluation of this guideline partially or totally. Therefore, the evaluation should not be restricted to specific types of guidelines.

### 3.2 Tool Architecture

Figure 2 shows the software architecture of our tool. A data base is used to store all provided or generated information to ensure their reusability for future evaluation sessions. Section 4 gives a brief description of concepts used in Fig. 2. A detailed description is given in [3].
Notice that this architecture ensures giving the evaluator high control at two levels:
- Manipulation of the set of included guidelines: inclusion of new ones or deleting of existing ones.
- Utilization of the tool: the evaluator has many configuring possibilities: decide what HTML tags to target, what guidelines or part of a guideline to evaluate, etc.

3.3 Evaluation Approach

In this section, we present brief description of the evaluation approach adopted in KWARESMI.

3.3.1 Separating Guideline Evaluation Logic from Evaluation Engine

Instead of hard coding the evaluation logic of guidelines in the evaluation engine, KWARESMI enables the evaluator to express guidelines in an evaluation-oriented manner in a declarative language. Rather than being imperative or procedural, the GDL language allows people to declare the way a guideline should be evaluated, not how it would be evaluated. It is the task of the tool to interpret these declarations at evaluation-time to produce any on-demand tailored, customized evaluations. The new guidelines structure is stored in the tool database, thus separated from the evaluation engine. The structuring is realized as follows:
- The module Guideline structurer (Fig. 2) is used to re-express guidelines in terms of HTML elements that are grouped in what we called evaluation sets. Each set represents the zone of a Web page to be examined to evaluate the whole (or a part of the) treated guideline.
- These evaluation sets are then used to define evaluation conditions that determine when to consider that the evaluation set is respected or violated.
- Evaluation sets and evaluation conditions are then used to define the evaluation algorithm that describes how to evaluate the whole guideline. All this information is stored in the tool data base to be used or modified later, thus ensuring more flexibility in controlling the evaluation logic.

3.3.2 Page Parsing

It is obvious that most of the time we do not need to parse the whole Web page to evaluate a given guideline. For example, to evaluate the guideline that recommends providing alternative text for images, all what we need is to check occurrences of IMG tag. Our parsing strategy is based on scanning the Web page and extracting only the tags and attributes that we used when structuring our guidelines in term of evaluation sets. Thus, the parsing process cannot be done before defining the evaluation sets. The result of the parsing is the instances of evaluation sets related to the evaluated guideline.
3.3.3 Page Evaluation

To evaluate the Web page, we use the evaluation algorithm that was defined to determine the order of checking the evaluation sets. Checking an evaluation set is done by checking its instances that were detected in the parsed page to see if they respect or not the evaluation condition that was associated with the evaluation set.

4 Implementation and Application

4.1 Implementation

KWARESMI prototype is implemented using Java language (Forte for Java, from SUN Microsystems) to ensure future portability. We needed to define our HTML specific parser and data structure (not a DOM) to ensure optimal and configurable parsing of the evaluated Web page. Currently, KWARESMI works locally as an off-line tool. A natural evolution would be to make it accessible via the Web as an on-line tool. Many existing tools exist in this similar shape. Any guideline is structured into a GDL statement, consisting of XML-based expressions of the elements sets of concern. The evaluation of any GDL-based guideline starts from interpreting its GDL declaration and turning it into evaluation instructions at evaluation-time.

4.2 Application

In this section we will show an example of application of KWARESMI. Let us assume that we want to evaluate only one guideline for a particular web page: Select colors that will make your page easy to read by people with color blindness [14,18].

This guideline, as it is represented, cannot be automated in a straightforward manner as there is no calculable way to assess to what extend a page is easy to read or not, depending on the users. However, if we refer to the research conducted by Murch [13], an interpretation of this guideline can be produced: “The combination between the background color and the foreground color should belong to the best color combinations or should not belong to the worst color combinations”. In this way, the original guideline is converted into a restriction that is subject to automation, while the original is not calculable.

4.2.1 Guideline Structuring in Terms of Evaluation Sets

By examining HTML semantics, we identify all elements that can be used to manipulate color in a Web page. We found the following elements:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Tag.Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text color via Body</td>
<td>Foreground color for text</td>
<td>Body.Text</td>
</tr>
<tr>
<td>Link color</td>
<td>Color of text marking unvisited hypertext</td>
<td>Body.link</td>
</tr>
<tr>
<td>Visited link color</td>
<td>Color of text marking visited hypertext links</td>
<td>Body.vlink</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Selected link color</td>
<td>Color of text marking hypertext links when selected by the user</td>
<td>Body.alink</td>
</tr>
<tr>
<td>Background color</td>
<td>Background color for the document body or table cells</td>
<td>Body.bgcolor</td>
</tr>
<tr>
<td>Text color via Font</td>
<td>The &quot;FONT&quot; element changes the color for text in its contents</td>
<td>Font.Color</td>
</tr>
<tr>
<td>Table background</td>
<td>Table background color</td>
<td>Table.bgcolor</td>
</tr>
<tr>
<td>Row background</td>
<td>Background color of a row in a table</td>
<td>TRbgcolor</td>
</tr>
<tr>
<td>Header background</td>
<td>Background color of a header cell in a table</td>
<td>THbgcolor</td>
</tr>
<tr>
<td>Cell background</td>
<td>Background color of a cell in a table</td>
<td>TDbgcolor</td>
</tr>
</tbody>
</table>

These elements are then grouped into evaluation sets. This grouping is highly influenced by the interpretation of the guideline by the evaluator. For a very precise grouping, we can have the following evaluation sets:

- **S1**: control of text color in the whole page. $S1 = \{\text{Body.text, Body.bgcolor}\}$
- **S2**: control of text color outside Table structure. $S2 = \{\text{Body.bgcolor, Font.color}\}$.
- **S3**: control of color by Font and Table. $S3 = \{\text{Table bgcolor, Font.color}\}$.
- **S4**: control of color by Font and TH. $S4 = \{\text{THbgcolor, Font.color}\}$.
- **S5**: control of color by Font and TR. $S5 = \{\text{TRbgcolor, Font.color}\}$.
- **S6**: control of color by Font and TD. $S6 = \{\text{TDbgcolor, Font.color}\}$.
- **S7**: control of color by Body and TH. $S7 = \{\text{THbgcolor, Body.text}\}$.
- **S8**: control of color by Body and TR. $S8 = \{\text{TRbgcolor, Body.text}\}$.
- **S9**: control of color by Body and TD. $S9 = \{\text{TDbgcolor, Body.text}\}$.
  
- **S10**: control of color by Body and Table. $S10 = \{\text{Table bgcolor, Body.text}\}$.
- **S11**: control of links color. $S11 = \{\text{Bodybgcolor, Body.link}\}$.
- **S12**: control of active links color. $S12 = \{\text{Bodybgcolor, Body.alink}\}$
- **S13**: control of visited links color. $S13 = \{\text{Bodybgcolor, Body.vlink}\}$

After defining evaluation sets, we can parse any Web page to detect instances of any of these sets. Figure 3 shows a screen shot of KWARESMI module for sets definition:

- We define tree set priorities: “A”, “AA”, and “AAA” with concordance with priorities defined by WAI for WCAG [18].
- The field *Instances* enables the evaluator to decide which set instances to take into consideration. At preset, possible choices are: “All” to get all instances, “First error” to stop detecting instances of this set when detecting first instances violating the evaluation condition associated with the set, and “N” to detect first N instances (if exist).
- A set component is Constant if it must be present in all instances of the evaluation set. For example, in Set2 above, all the instances of the set have *Body.bgcolor* as constant component, a new instance is created for every occurrence of the component *Font.color*. 

369
A set excludes one (many) other set(s) if its scope overcomes the scope of the other(s). For example, `Table.bgcolor` overcomes `Body.bgcolor` in the table scope. Thus, Set3 excludes Set2. Exclusion concept is used to optimize the evaluation execution by ignoring the evaluation of excluded sets.

![Figure 3. Defining evaluation sets in KWARESMI.](image)

### 4.2.2 Evaluation conditions

For every evaluation set we define the evaluation condition. To define condition for our example, we consider the findings of Murch [13] as illustrated in Figure 4.
So, we can define evaluation conditions as follows:

S1: positive evaluation condition:

\[
\text{(Body.text IN ListOfGoodColors(Body.bgcolor)) OR (Body.text NOT IN ListOfBadColors(Body.bgcolor))}
\]

Where ListOfGoodColors and ListOfBadColors are two lists of predefined values (colors) that can be introduced at the beginning of conditions definition.

S2: positive evaluation condition:

\[
\text{(Font.color IN ListOfGoodColors(Body.bgcolor)) OR (Font.color NOT IN ListOfBadColors(Body.bgcolor))}
\]

The remaining conditions are similar to the above ones. Figure 4 shows a screenshot of KWARESMI module for conditions definition:

**Figure 4.** Ergonomic knowledge used to define evaluation conditions

<table>
<thead>
<tr>
<th>Background</th>
<th>Thin lines and text</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>Blue (34%) Black (69%) Red (25%)</td>
</tr>
<tr>
<td>Black</td>
<td>White (75%) Yellow (63%)</td>
</tr>
<tr>
<td>Red</td>
<td>Yellow (75%) White (69%) Black (45%)</td>
</tr>
<tr>
<td>Green</td>
<td>Black (100%) Blue (69%) Red (25%)</td>
</tr>
<tr>
<td>Blue</td>
<td>White (75%) Yellow (63%) Cyan (25%)</td>
</tr>
<tr>
<td>Cyan</td>
<td>Blue (65%) Black (65%) Red (31%)</td>
</tr>
<tr>
<td>Magenta</td>
<td>Black (65%) White (69%) Blue (41%)</td>
</tr>
<tr>
<td>Yellow</td>
<td>Red (65%) Blue (63%) Black (55%)</td>
</tr>
</tbody>
</table>

**Good color combinations for thin lines and text**

<table>
<thead>
<tr>
<th>Background</th>
<th>Thin lines and text</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>Yellow (10%) Cyan (94%)</td>
</tr>
<tr>
<td>Black</td>
<td>Blue (67%) Red (27%) Magenta (25%)</td>
</tr>
<tr>
<td>Red</td>
<td>Cyan (81%) Magenta (66%) (15%) (27%)</td>
</tr>
<tr>
<td>Green</td>
<td>Cyan (81%) Magenta (56%) (25%) (17%)</td>
</tr>
<tr>
<td>Blue</td>
<td>Green (32%) Red (9%) Blue (62%)</td>
</tr>
<tr>
<td>Cyan</td>
<td>Green (31%) cyan (69%)</td>
</tr>
<tr>
<td>Magenta</td>
<td>Cyan (29%) cyan (44%)</td>
</tr>
<tr>
<td>Yellow</td>
<td>White (91%) cyan (59%)</td>
</tr>
</tbody>
</table>

**Bad color combinations for thin lines and text**
Defining evaluation conditions seems to be very complicated. In fact, this is due to our desire to enable the evaluator to have (total) control over evaluation logic definition. We could put colors: Blue, Black, and Red in one predefined value `GoodWhiteColors`, put colors: Green and Cyan in one value `BadWhiteColors` (via the +Content button). In this case, the final evaluation conditions become:

\[
\begin{align*}
ACond_1 &= (\text{Body.bgcolor Equals White}) \\
ACond_2 &= (\text{Font.color IN GoodWhiteColors}) \\
ACond_3 &= (\text{Font.color IN BadWhiteColors}) \\
\text{Evaluation Condition} &= (ACond_1 \ \text{AND} \ (ACond_2 \ \text{OR} \ ACond_3))
\end{align*}
\]

As KWARESMI supports our GDL language, it will be possible to let the evaluator directly type in the final evaluation condition. The typed text will be parsed to see if it respects the GDL syntax\(^2\).

\(^2\) This mode of interaction is called “command line mode”. It is usually convenient for users that have good knowledge of GDL syntax. the interaction mode as actually provided in KWARESMI is the “Menu selection mode” which is more convenient for GDL novice users than the other mode.
4.2.3 Evaluation Algorithm
Actually KWARESMI uses the default evaluation algorithm which consists of evaluating all the detected sets instances by order of priority. Defining evaluation sets and conditions as we show above should offer interesting optimization possibilities during parsing step or evaluation execution. These possibilities are under examination as we continue testing the tool over new guidelines.

4.2.4 Page Parsing and Evaluation
As it does not do any evaluation optimization for the moment, KWARESMI parses the web page then evaluates it directly following the default evaluation algorithm. Figure 6 shows a Web page and the instances of different evaluation sets.

![Figure 6. Evaluation of a web page: browser View (Left), HTML View (Middle) sets instances and evaluation result provided by KWARESMI (Right)](image)

5 Conclusion and perspectives
In this paper we presented our KWARESMI tool we are developing to support our guideline definition language (GDL) as an automatic evaluation tool of Web ergonomic guidelines. The tool and the underlying GDL would overcome main shortcomings of existing automatic evaluation tools:
- The tool should allow the dynamic evaluation of any guideline that can be expressed in GDL without any code modification of the evaluation engine.
- It should enable an evaluator to define and redefine evaluation logic of a guideline if needed, again, without the need to modify and recompile the evaluation engine.
It should enable the evaluator to realize on-demand evaluation by providing him by high control level of the evaluation process.

When applied on simple examples, the developed prototype produced promising results. With the advantage of the tool and the GDL language, there is now a possibility of introducing, managing, structuring, and evaluating guidelines at a level of abstraction that is higher than the one of HTML level. Of course, there is a long trip to transform a guideline expressed in natural language to its equivalent code working on HTML tags. This is why GDL and KWARESMI are located between to bridge the gap between a high level of abstraction (natural language) and a low level of implementation (HTML).

It is worth to note that currently implemented prototype covers guidelines evaluation in a single page only. It can not be used for guidelines that cover inter-pages relations or link-structure issues. Extending the tool toward this direction is planned after validating our results in single page evaluation.

**Acknowledgements**

This work is funded partially by the Belgian Laboratory of Computer-Human Interaction (BCHI), Information Systems Unit (ISYS), School of Management (IAG), Université catholique de Louvain (UCL, Belgium) and the Energy Commission of Syria. The first author wants to thank Christelle Farenc, Université Paul Sabatier (France), for her helpful comments during the preparation of this work.

**References**


Web Accessibility Initiative of W3C (1999). Accessible at http://www.w3c.org/WAI.

Net-Fly and ContextControl
Two Examples of Game Based Interfaces

Holger Diener, Mathias Mainka, Hagen Schumacher

Fraunhofer Institute for Computer Graphics
Joachim-Jungius-Str. 11
18059 Rostock, Germany
{Holger.Diener, Mathias.Mainka,
Hagen.Schumacher}@rostock.igd.fhg.de

Abstract. Highly complex menu and dialog structures in standard applications like word processing or spreadsheets contradict good operability and deter beginners and occasional users. In this paper, we present two examples for a new kind of user interface based on games. These game based interfaces are easier to learn and can be specialized to different tasks and contexts. We give some information about ContextControl, a widget to control existing applications with a game based interface and we discuss Net-Fly, a module for playable assistance for search engines.

1 Introduction

Former applications with command line interface were hard to use, because the user had to remember all commands to access functionality. Users even had to switch between command and writing mode using special command keys. In the early 1980s, these problems were reduced by the introduction of graphical user interfaces, where icons and menus made it easier finding commands.

Today, a similar problem comes up with graphical user interfaces. The huge amount of functionality in today’s application makes it hard to find the right menu item in highly complex menu and dialog structures. Although this contradicts good operability, it is not astonishing, because more functionality then a competitive program is always a very good sales argument.

Experts may get along with such applications, but the important groups of beginners and occasional users are deterred from such extensive applications. The user’s motivation to explore these applications is suppressed by the vast number of possibilities. Finally, users will switch to easier and simpler applications, even if those do not offer all functionality that is needed.
1.1 Game Based Interfaces

To overcome these problems we need new structures and concepts for user interfaces to rearrange the functionality of applications. Computer games may be a good source for such new approaches, because games are using new ideas to mediate their functionality. Within the scope of the project “Play the Application”, we developed new user interfaces for standard applications based on game concepts: ContextControl for an easier control of applications and Net-Fly to assist usage of search engines. These so-called Game Based Interfaces are adjusted to special tasks, easier to learn and faster to use.

In this paper, we compare games and applications with regard to software ergonomics. Afterwards we discuss the possibilities and advantages of game based interfaces illustrated by these two approaches and show other fields of application.

2 Games and Applications

The initial reason for our investigations of computer games in order to improve applications was the common assumption, that everyone can play games, that they are intuitive and easy to understand. Games seem to have all the features we are looking for in applications.

On closer examination, many games are not intuitive. For example flight simulators or real time strategy games are hard to learn and even harder to control. However, games like Tetris or Solitary are easy to understand and can be played even by beginners. Our aim was the identification of game concepts, which make these games so simple and the transfer in applications.

The quality of applications can be assessed by standards of software ergonomics like suitability for the task and suitability for use. Regarding to these definitions we analyzed user interfaces of computer games. The next paragraph shows a short overview of our results and the second paragraph gives some ideas about what concepts can be used in applications.

2.1 Computer Games

According to software ergonomics, see EN ISO 9241-10, a good program has to be suitable for the task, self-descriptive, controllable, conform to user expectations, error tolerant, suitable for individualization, and suitable for learning. Observations of games have shown that many games are self-descriptive and suitable for learning. They use stories and metaphors to describe their user interface and functionality, e.g. a sword to fight and a shield to protect.

On the other hand, games do not have to fulfill all these rules to be good games with respect to sales, reviews, and user opinions. For example, adventures often are hardly controllable or error tolerant. Players have to solve most of the puzzles in the right order and even a slight mistake can be fatal. Players naturally complain if a game is faulty, but they do not complain as long as these nuisances are part of the game. No
one would use a word processing program, if users must rewrite documents whenever they forgot to press three buttons in the right order.

We see that standards of software ergonomics have no high priority for games, because the players motivation compensates for bad user interfaces. Although winning the game is important, the result of a game matters little compared with work reports.

**Context of Use.** Another very important issue of evaluation of programs is the context in which the programs are used. If we consider games, we see that there is more or less one objective; games are designed to entertain people. In contrast to games, most applications must meet many different objectives for different users with different tasks. From this point of view, it is much harder to design a program, which satisfied a great number of users, than to design a successful game.

That does not mean, that games are easy to implement. From a technical point of view, games are very demanding. Realistic 3D graphics in real time, high quality rendering, and multi player games are very challenging tasks, but in most cases it is easier to predict what a player will do with a game than what a user will do with an application.

**Simplicity.** Intuitive software often has a simple user interface. That is particularly valid for games. Many people can play simple games like Tetris, which needs only a small number of commands, but only some people manage complex games like flight simulators. The problem is, if software consists of many functions, the user interface must present these functions and therefore cannot be simple. Otherwise, it is not necessary to present all functions, but only that part of functionality that is necessary for the current task. Our question was, how could we reorganize functionality of a demanding application, that a user can intuitively use it.

### 2.2 Application

Difficulties with software systems are classified into three groups: Difficulties with the operation of the system, difficulties with the functionality of the system, and complains about monotonous and tedious work or about stress and too much work.

Whereas the functionality of a software system can only be changed from inside the system or during implementation, both of the other problems can be reduced from outside.

Operational problems often result from complicated and overloaded user interfaces. That is where the simplicity of games could help. If we can tidy up the interface, it would be easier to work with. However, this means we have to reduce the number of commands that are accessible by this interface. Because applications must meet the demands of different users with different tasks, for beginners as well as experts, this interface has to be very flexible. The ContextControl described in paragraph 3.1 meets these requirements. With this widget we implemented a tool, which is able to manage different subsets of user interfaces. These subsets are representations of certain subsets of functionality and can be ordered according to functionality or tasks [4].
Another concept of games, stories and metaphors, can be used to reduce problems from monotonous or stressful work. Stories can mediate complicated workflows and cause a better recollection of how to use the program. Additionally, they can be used to support the visualization of great amounts of information by showing condensed facts or bringing facts in an appropriate form. Moreover, a game makes more fun with a story. The Net-Fly module presented in 3.2 uses a playful metaphor to assist search engines.

3 Examples

As mentioned above users have problems with too much information or functionality. Therefore, we need assistants to simplify what we are seeing or using. In this paragraph, we exemplary present two technologies based on computer game concepts to assist users. The ContextControl uses the principle of simplicity, as in some games, which nearly all people can play. It is designed for simplifying the control of applications. Net-Fly creates a playable interface to specialize search queries.

3.1 ContextControl – A New Interface to Control Applications

Today, applications tend to contain many more functions than there is place to present in clearly arranged menus. This results in large nested menu trees, in which beginners and occasional users are lost. We introduce a new menu access widget, more precisely a context access widget that simplifies and speed up the access to a large number of functions like the Hotbox menu of Maya [5]. We describe the design and functionality of this widget, and how it manages different working contexts to enable easy and more comfortable working with sophisticated applications. These objectives lead to the name ContextControl.

For the prototype of ContextControl we have chosen to control Microsoft Excel. The prototype is implemented with MS Visual C++ using the object library from Excel to implement several important functions. The following sections describe the usage and design of ContextControl menu regarding to easy handling and fast working and gives a short overview about planned tasks close to our menu system.

How Does It Work. Many applications use context menus to support the workflow. Such a menu is easy to use because whenever additional functions are required to the current work step, a right mouse click will make available several important functions. However, context menus offer only a few functions. Additional functionality, which belongs to a larger working context including the last five or more work steps, sometimes requires an expensive search in menus or tool bars. Naturally, tool bars could not supply the whole functionality of a high level application and searching in menus can be very time-consuming so the workflow will be broken. With respect to easy using and being a useful add-on the ContextControl menu appears by clicking right and holding down the Ctrl-key at the same time. So we do not replace the usual
context menu directly and ContextControl can be used parallel. If the user needs two or more functions in quick succession he has to go through menu bar pop up trees twice or more times. However, ContextControl will stay activated until the Ctrl-key is released. Therefore, it is possible to call a sequence of several functions.

In the basic level of the ContextControl, only few menu items and few functional buttons can be found. Clicking on menu items reveals more menu items and functional buttons just around the selected menu. These new items belong to a special working context and stay open until the menu item is pressed again. Thus, the selected menus are already open when ContextControl is called the next time. That way the number of mouse clicks were reduced. Since all buttons have a fixed location in the menu area, we prevent overlapping items and ensure that the desired item can be found fast because the location is known.

![Fig. 1. Three states of the ContextControl starting with the basic level (left), the first sub menus (middle), and all functional buttons open (right).](image)

For fast access not only the arrangement of items is important, but also where the menu appears. Like the usual context menu, our ContextControl menu appears aligned to the cursor, which could be done by two different methods. First, we always center the ContextControl with respect to the cursor. Thus, distance and direction to the desired item is always the same. The second way is to calculate the balance point of the last clicked buttons and let the ContextControl appear with cursor on that point. That way, there can be another stretch of way to the wanted item after each menu call. However, the small difference does not matter because of the special item order, which ensures that the relative position of items does not change. For the same reason we prevent the menu from partly disappearing from the screen, if the user is clicking nearby the screen boarders. Generally, the user will work in the middle of the screen. Nevertheless, if he works near the screen borders, he does not need to call the ContextControl again. This will also ensure a smooth workflow.

Often it is necessary to offer new functionality if the task or workflow changes. For this, the user can load different menu sets while he is working. If the menu is activated the user can call the menu set controller by a right mouse click in the menu area. With this controller not only menu sets can be selected, but also up to five menu states can be managed. Each menu state is a snapshot of how the menus are opened. The user can save and restore these states very fast and can easily switch between states. This
can be used to tidy up the menu or alternatively to change the working context, because every state can represent the functionality for certain work contexts.

How Does It Look. In many computer games semi-transparent control panels and flying buttons are used. Furthermore, many illustrations can be found instead of text. Small icons with a short tool tip will support memory more than only text. Of course, applications use such small images in tool bars. Thus, we let the user choose between two basic designs. The first is to leave the buttons alone without carrying window. Sometimes, this could lead to very confusing views, therefore the second design based on a semi-transparent window to give menu items more contrast to the background. Through a semi-transparent window the current work context can be kept in view while choosing the suitable menu item.

Fig. 2. The ContextControl widget used for sales figures with Microsoft Excel

Like tool bars, ContextControl uses small icons instead of short text to illustrate menu items. Using images and grouping buttons concerning a special work context are very important for recognizing the functionality behind the buttons. For example, a group of four buttons is used to justify text in most applications. Left justified, center, right justified and justification are unambiguous represented with only six lines each. To support understanding a background image on the semi-transparent window could give additional information of the work context. In addition, we can use animations for better understanding functions and workflows.

Customization and Individualization. In a separate editing mode existing menu sets can be modified or new ones can be created. It is possible to generate own menu and
sub menu items and arrange functional buttons around them. The user can assign predefined icons or personal bitmaps to every button. The according function can be chosen from a static list of several important functions of the application. Using the semi-transparent window the user can change the background color or select a background image. Additionally, he can adjust the level of transparency. Thus, he is able to adapt the menu view to the application view for an optimal contrast.

### 3.2 Net-Fly – Playable Assistance for Information Retrieval

The size of the World Wide Web as our biggest source of information makes it nearly impossible to get a complete overview of the response to any query. Of course, there are some good search engines available, which are the first place to go getting information from the Internet. Nevertheless, if we have only a vague idea of what we are searching, it is difficult to find the right search words. General search words lead to huge amounts of useless information and we will lose motivation for a more intensive search. If we choose very special search words, the search engine often responds with no result. However, if we know all essential words for this topic, why would we need a search engine?

In this paragraph we present the module Net-Fly to assist users in finding the relevant search words starting with general words. This module is implemented in Java and can be used with almost every public search engine. For better visualization and easier handling it has a game based interface, which presents the data in a playful manner to increase motivation and efficiency.

As a combination of reasonable data representation with useful intelligent assistance, Net-Fly can really improve information retrieval.

**Assistance for the Right Request.** The first step in information retrieval is to create the right query. In case of search engines this means, we have to look for the right keywords. Since words can be ambiguous, it is a good idea to combine special keywords with some words describing the desired context. For this reason query words can connected with logical operators like OR, AND or NOT.

If the user has only an imprecise idea of what he is looking for, he can start with a rather general query word to give a hint for the right ontology or context. Now the Net-Fly module does his work. It sends this start query word to some search engines and filters the texts, which belong to the most relevant response URLs. These texts represent the current knowledge of the Net-Fly module.

Now the Net-Fly module tries to give useful hints in form of words related to the initial search words to concrete the query. For this, words with low information content, like articles, pronouns and conjunction words will be filtered out of the response texts. We get a list of words, in which every word is linked with information about the website of origin and the position in the text.

There are different methods to rank the words, which the user can control over the module. Every word, stored in a hash-table, knows his frequency and his closeness to the query word or words. It is easy to set any function on this data, e.g. the words can be sorted by frequency, and therefore Net-Fly presents words with highest occurrence
at first. Another way is to start with words, which stands near to query words. Maybe
the closeness to the query implies a closer connection to the desired information.

First tests have shown that a compromise of frequency and nearness gave the best
result on average. That means words which standing frequently near to query words
get highest priority. While the user is confronted stepwise with words, he is able to
build a better or concreter query.

The Psychoanalytic Model. The psychoanalyst Sigmund Freud introduced a method
called free associations to get access to forgotten things, which we bear deep in our
mind. Freud assumed that the knowledge stored in our brain is connected through
connotations. Thoughts and memories are chains of associations of ideas. By such an
association chain, a person can remember long forgotten experiences.

Net-Fly is like a patient, who let his mind flow starting with the initial search
words. The user as an analyst has to pick out the interesting data and gives new im-
impulses for the right direction.

Presenting a Usable Response. Mostly information retrieval has to be quick and
efficient. For this reason Net-Fly filters the topmost websites from the connected
search engines regarding to the desired information. The user gets text sections with
important information from different websites. For every section the user can chose
via context menu to load the corresponding website, to get more information from this
website or delete this section from the text.

The aim is to prevent the user from useless information. The key is here to give
short helpful associations so that the user can walk the right way through the labyrinth
of information. The problem is, we do not know the destination of the way at first.
However, the longer we walk the better we get the right direction and with this direc-
tion the right way and the destination.

Game Based Interface Representation. A graphical representation makes a program
easier to understand. Pictures combined with words can be much more intuitive than
words alone. Our aim was to develop a graphical interface for Net-Fly, which allows a
playable choice of appearing words and a graphical representation of our query. As
background scenario, we have chosen a beach at the Baltic Sea where objects with
words are appearing.

Using a spyglass the user can look at objects, respectively the connotation words, at
the beach and he is able to click left or right with the spyglass on every word to create
a more precise query. Left click to include words or right click to exclude words to the
query. The query words occur as clouds at the sky. The user also can type in words
manually after clicking the sun. Clicking on a cloud, which is moving at the sky, or
clicking at an object at the beach removes these words. In this way the user has a fast
control over the current query words and connotation words.
4 Future Work

The context of work and user preferences are the most important research topics for both technologies, Net-Fly as well as ContextControl. The more we know about the user and his work the more we can conclude his most needed information or functionality. Therefore, we have to focus on analyzing the users and their work.

A basic method for the ContextControl will be an analysis of how users are working in different contexts. With these results we are able to create a base of a user interface guide system with predefined workflows. This system will guide beginners and occasional users through the menu structure of new applications step by step with increasing complexity.

The next step will be a system that can detect the current work context by observing the work. Thus, the system can adapt the user interface of the ContextControl. However, not only the current work should be taken into consideration for a user interface guide system, but also physiological user data, to make it possible to react to stress, irritation or distraction.

For the search-assistant Net-Fly the knowledge about the current working context can reduce ambiguities in the response. For this, we need some kind of semantic networks or ontology, where the relations of words depend on the current context. To generate such network, an analysis of other texts related to this context could be helpful.
5. Conclusions

We presented two examples of game based interfaces, which are designed to help users cope with complicated applications or difficult tasks. As standard applications often have complex user interface structures to represent their functionality, ContextControl is able to rearrange functionality with respect to working contexts. The user can create button groups, which are specialized for one task and switch to another group if the task changes. Therefore, users only have to deal with a small number of clearly arranged buttons and can easier access the wanted functionality. Additionally, visual and textual assistance can be integrated in this widget to guide user through their work. Simplicity and assistance are good approaches to improve standard user interfaces.

The other example, Net-Fly, shows the use of playful metaphors to introduce the functionality of an application and combines this with an associative assistance for search engines. Users can start with more general search words and can refine their search by selecting or rejecting related words given from the Net-Fly module. This approach gives beginners an easy tool to start searching the web.

The possibilities of game based interfaces are not exhausted with controlling Excel and assisting search engines. On the contrary, they can be extended to further fields of applications like education and training, edutainment or even mobile applications.

6. Acknowledgements

The approaches described in this paper are current research work within the project “Play the Application”. The project is sponsored by the INI-GraphicsNet Foundation.

References

7. Rodden, K.: About 23 million documents match your query... Relevance feedback as an information retrieval technique that can be used to make a search query more specific. Doctorial Consortium ACM ISBN l-581 13-028-7
Towards a Ubiquitous Semantics of Interaction:
phenomenology, scenarios and traces

Alan Dix
Lancaster University, Lancaster UK, alan@hcibook.com
http://www.hcibook.com/alan/

Abstract. This paper begins a process of building a semantic framework to link the many diverse interface notations that are used in more formal communities of HCI. The focus in this paper is on scenarios – single traces of user behaviour. These form a point of contact between approaches that embody very different models of interface abstractions or mechanisms. The paper looks first at discrete time models as these are more prevalent and finds that even here there are substantive issues to be addressed, especially concerning the different interpretation of timing that become apparent when you relate behaviour from different models/notations. Ubiquitous interaction, virtual reality and rich media all involve aspects of more continuous interaction and the relevant models are reviewed. Because of their closer match to the real world, they are found to differ less in terms of ontological features of behaviour.

1. Introduction
In the formal HCI literature, notations and modes of description seem to proliferate without limit, both system-oriented dialogue notations and more user-oriented goal and task descriptions. This paper aims to start a process of uncovering common semantic models for user interaction.

There are several reasons for taking on this task:
• practical – so that we can confidently use multiple notations applied to real systems and have a basis for interchange between support tools
• theoretical – so that we can give common semantics to diverse notations, and so understand their overlaps and differences
• philosophical – in grappling with these semantic roots, we begin to have a better grasp of the meaning of interaction

Furthermore, HCI is changing from the 1980's "one man and his machine" to a situation with many people and many devices embedded in a rich environment. Dourish calls this 'embodied interaction' and describes it as "interaction in real time in the real world" [16]. There is comparatively little formal work on these new devices and modes of interaction. Extending existing techniques is going to be a major challenge of the next few years and revisiting our semantic roots, one way to make sense of these multiplying computational phenomena.

So this semantics aims to be ubiquitous in that it is both applicable to:
• notations addressing different aspects of the user interface: task, goal, dialogue behaviour, system state, informal and formal
• types of interaction: GUI, wearable, mobile, ubiquitous, continuous and discrete
In saying "start a process of uncovering common semantic models" this is not because there are no existing semantic frameworks. Indeed, many formal papers have been based on semantic models rather than notations (e.g. the PIE model [11,12], the template model [36], LADA [15], status-event analysis [13,14]). In addition, some notations do have their own semantic models (e.g. trace semantics of process algebras.

However, as a discipline we have few common points to anchor our diverse activities, in contrast to, say, architecture, where the intended physical building links diverse representations (scale models, service schematics, artist's impressions, plans).

This paper proposes that such anchor points are valuable and starts an explicit process of addressing the resulting agenda.

To some extent both the original PIE model and, in a different way, the syndetic modelling framework [6] purport to be universal models of interaction. In this work the aim is to be less normative and more inclusive, setting up a framework for linking multi-paradigmatic analyses rather than proposing a single multi-faceted analysis.

This is potentially a huge task, so this paper does not attempt to address the whole question, but focuses on a phenomenological semantics of scenarios and traces of activity. Observable phenomena are often the easiest point of contact between different descriptions (as in the case of architecture). Furthermore, they are the point of contact between systems and people and so can relate user and system descriptions.

The next section revisits this rationale in greater detail, looking at why diverse notations exist (and should exist), the need for common semantics and the power and complexity of scenarios. Section 3, looks at discrete traces of activity, as is most common in more formal approaches such as STNs, process algebras, grammars and Petri nets. Section 4, considers continuous phenomena, which have more complex temporal behaviour and cannot be characterised as easily in simple state transition schemes. In both cases we will be interested in the way that different levels and different kinds of description may be related to one another, including the relationship between discrete and continuous representations. Finally, we return to the wider agenda and further areas required for a common semantic framework.

2. Rationale

2.1 Bewildering Diversity

There is something about computing that encourages the proliferation of notations and the more formal the area the more different notations we find. This is certainly true in the more formal aspects of user interaction. There are good reasons for this:

• we need to state things precisely, whether we are talking about system states or user tasks, hence notations
• we have different concerns and so want to discuss easily various significant aspects, hence diversity

However, there are downsides:

• a danger of focusing on notations themselves as significant ('I can do this with mine' papers)
• a confusing plethora of incompatible notations understood only by their particular cognoscenti

The latter problem is especially clear in systems specification notations (pick up a previous DSVIS proceedings – do any two papers share a notation?), because they
tend to be more detailed and differ in the formal expression of those details. In contrast, user-focused notations (e.g., task/goal descriptions) often have 'fuzzy' boxes at the lower levels and allow annotation for complex interactions, making them easier to understand and employ by those outside the notation's cabal.

So, despite the strengths of diversity, there are problems both within the community who accept the importance of formalisation and even more so outside.

2.2 Common Semantics

This paper aims to address these problems in part by starting the process of producing a common semantic framework for user interaction notations.

Note this is not a 'unified notation' or even 'unified method' involving multiple notations, but instead a means of understanding and relating existing and new notations. Neither does this paper contain an extensive review of UI notations, although a common semantics certainly aids the process of comparing and selecting appropriate notations.

Semantic models have proved invaluable in many areas of computing. For example, denotational semantics not only produced a common framework for expressing semantics of programming languages, but lead to a common vocabulary for describing them (binding environment, continuations etc.). Although not usually linked to the theory, we see a similar 'coming together' at a practical level when modules written in different programming languages are compiled into linkable object files – a common semantic form – allowing multi-language programming.

The need for this common semantics is evident in several areas of UI modelling. For example, CTT and ICO have been linked using scenarios [25] which effectively established a de facto common semantics between them. Also in model checking, researchers always find themselves manipulating the UI notation to translate it into the notation for the model checker – is this the 'right' manipulation, would a different 'translation' give different results? As interaction becomes more rich and diverse the need for common underpinnings will increase.

Although semantic models are in some ways more abstract than notations, they are often easier to 'ground' in reality and easier to reach common agreement. This is because notations have many concerns over and above 'meaning': tractability, clarity of expression, maintenance etc.

Although we look for a common semantic framework, this does not mean a single unified model. Instead we will find a collection of complementary descriptions that map on to one another, following the pattern of many mathematical formalisms, with several complementary descriptions for 'the same' type of thing. For example, in topology we can start off with open and closed sets or with 'neighbourhoods' of points and from each derive the other. There is even a 'pointless topology' that has neighbourhoods as the primary objects and then 'recovers' points.

This pattern of multiple related formulations exists within several specific methods. Cognitive complexity theory [20] had a production rule description of goal driven user activity and a system description and verified that the goals can be achieved using the system. This was possible because the lowest levels of goal description involved explicit user actions which could be mapped to system commands. In Abowd's thesis [4] he used both an internal CSP-like process algebra description and also an external 'set of traces' description. These were not duals of one another, but instead specified different aspects of the required behaviour. The
overall agent behaviour was constrained to satisfy both requirements. Again the linkage was possible because both descriptions share a common event vocabulary. In syndetic theory different interactors are described in their own 'microtheories' then linked by 'macrotheory' [6].

The linking between different semantic models is complicated by the fact that the concrete semantics of a description in a particular notation is governed not just by the formal semantics (where it exists) of the notation, but also the 'binding' of the constructs in the description to phenomena in the real world.

For example, consider a two state toggle (fig. 1.i) with two states, S1 and S2, and a single action, A, that moves back and forth between the states. This has a very clear formal semantics (fig. 1.ii shows one behaviour assuming start state is S1).

However, it makes a big difference whether we 'bind' this to the on-off state of a computer (S1 = off, S2 = on, A = on/off button). In this case all other activity in the computer is enabled/disabled by this state. In contrast, if could 'bind' this to a bold character format tick box in a word-processor dialogue box (S1 = bold-off, S2 = bold-on, A = click the tick box). The network is only enabled while the dialogue box is visible and may be operated concurrently with other parts of the interface.

Some of this binding may be specified within the formalism, but some level of binding is always outside the formal system itself.

![S1-A-S2-A-S1-A-S2-A-S1...](i) STN of toggle (ii) semantic behaviour

**Fig. 1.** Formal semantics of two state toggle

### 2.3 Scenarios and Phenomena

Although notations differ markedly in the way in which they describe and structure interaction, they 'come together' at the surface level of user and system actions. This is inevitable as in the end an expert in the notation should be able to look at an actual trace of observable interface behaviour and say 'yes' or 'no' that is or is not valid with respect to this specification. In addition, internal structural elements, such as states, goals, tasks, internal events, can often be associated with points or portions of the trace – 'this happens then'.

This use of scenarios as lingua franca can be seen in various places. In the CTTE tool, rich scenarios are mapped onto ConcurTaskTrees (CTT) [31], which are then used to generate formal traces of actions to validate ICO user interface specifications in PetShop [25]. Model checking is used on several UI notations including LOTOS and CTT [28,30], propositional production systems [2] and State Charts [22]. The advantage of model checkers over theorem proovers is that when they fail they give a counter example, in the form of a trace of actions – a scenario.

Looking further than UI notations and formalisms, scenarios have been used as a common focus for diverse design techniques [8], in the form of stories, incidents or snippets of interaction, they are part of the common language of more sociological descriptions of human behaviour, and several 'soft' descriptions such as patterns or cases involve some sort of scenario. So, focusing on 'what happened' or 'what might happen' also puts us in a better position to relate to this level of description.
Note that the focus on phenomenological semantics is not because it is the only level of valid description, as has become a common philosophical position in many areas of HCI. Instead, it is the focus as a practical point of 'agreement' between diverse descriptions, including those involving motivation, intentions and cognitive process of human agents and the mechanisms and abstractions of computational components ... and even descriptions which deny the validity of either!

3. Discrete Behaviour

This section is about discrete behaviour in the sense that it takes place in discrete time. Whether the value domains for phenomena are discrete or continuous is not really relevant as it makes little difference to our models. In continuous time however, we will see that these differences in value domains are more significant.

At first discrete models seem very simpler. They are certainly far more common. In Palanque and Paternò's collection [26] no paper deals with continuous phenomena and in recent DSVIS only three papers in total. However, in some ways discrete models are more complex. The real world exists in continuous time and so we have a touchstone with which to compare our semantics. With discrete behaviours we have to choose which moments we deem significant and the way in which we relate phenomena, which take place over finite times into momentary measures.

3.1 Trace as Event Stream

As a first model of discrete behaviour we'll consider a simple stream of events:

\[ \text{event 1 – event 2 – event 3 ...} \]

This is precisely the model used for grammars such as BNF [35] or TAG [34] and for process algebras such as LOTOS [27] and CSP [3]. For example, given a process:

\[ P \rightarrow a \ b \ P \mid a \ c \ P \]

Then a potential behaviour is:

\[ a \rightarrow b \rightarrow a \rightarrow b \rightarrow a \rightarrow c \rightarrow a \rightarrow b \rightarrow \ldots \]

One way to model this is as a function from some time domain Time to events:

\[ \text{trace:} \ Time \rightarrow \text{EventKind} \}

The optional value is to allow for models where the general type of event (e.g. channel in process algebras) may also allow some form of value passing. However, this is not just a feature of process algebras. If we were analysing a trace of user interface events we might say something like "the user clicked the mouse" – this is a recognisable kind of event which has an associated value "at pixel location 517, 323".

Although this 'works' as a representation of event traces, it puts too much emphasis on the arbitrary time domain. Most commonly Time is an interval of the integers, but this suggests uniformity of the time steps which is rarely the case.

An alternative is to tear apart the phenomena and regard a behaviour as a tuple:

\[ \text{Behaviour} = <\text{instances, when,what}> \]

where

\[ \begin{align*}
\text{instances:} & \quad \text{set of EventInstance} \\
\text{when:} & \quad \text{partial order EventInstance } \times \text{ EventInstance} \\
\text{what:} & \quad \text{EventInstance } \times \text{ EventKind} \{ \approx \text{Value} \}
\end{align*} \]

The EventInstance is merely a unique label allowing us to distinguish events that have similar external appearances but occur at different instants – such as the different 'a's
in the trace above. The order is the weak 'happens before' relationship. Notice how this allows us to separate out when the event happens – the order of events, from what happens – the event kind and possible value.

Both 'when' and 'what' have the obvious constraints:
\[ \text{dom(when)} = \text{range(when)} = \text{dom(what)} \]

Note the partial order rather than a total order as some notations, such as LADA [15] represent truly concurrent, as opposed to interleaved, events.

Optionally, we can then add actual times to these events
\[ \text{at: EventInstance} \]

With the temporal constraint (N.B. second order is 'when' order):
\[ \forall \ e_i1, e_i2 \in \text{instances} \cdot \text{at}(e_i1) < \text{at}(e_i2) \]

Notice this is carefully phrased we do not have a double implication. Some notations allow several simultaneous but causally ordered events – so we allow instances to be 'at' the same time, but ordered by 'when'. Note also that we can easily transform a time function trace into this more general behaviour.

Finally, although we shall use this model as a touchstone in the rest of this section it is not intended to be a final point, merely of sufficient richness to discuss various issues and flexibility to allow the necessary extensions to produce a truly ubiquitous semantics.

### 3.2 Relating Behaviours

We can relate behaviours of this kind to one to another by introducing a partial mapping between event instances. That is, Let X and Y be two behaviours and \( B_{XY} \) a binding between them:

\[ B_{XY} : \text{partial mapping} \quad X.\text{instances} \rightarrow Y.\text{instances} \]

A relationship of this kind can arise in two ways:

- a particular trace of activity is captured and the actual simultaneity of particular related events in the two models recorded automatically or annotated by hand
- a binding is defined between models, which then gives rise to an induced binding between behaviours of models

The binding mustn't introduce an inconsistent ordering:

\[ \forall \ x_i1, x_i2 \in X.\text{instances}, \ y_i1, y_i2 \in Y.\text{instances} \cdot \]
\[ B_{XY}(x_i1, y_i1) \sqsubseteq B_{XY}(x_i2, y_i2) \]

This weak ordering preservation is to allow things such as one event in one model to correspond to several in another.

However that this can lead to some 'interesting' orderings. For example:

\[ X.\text{instance} = \{a, b, c\}, \quad X.\text{when} = \{a < b, a < c, b < c\} \]
\[ Y.\text{instance} = \{p, q\}, \quad Y.\text{when} = \{\}, \quad B_{XY} = \{(a,p), (b,p), (b,q), (c,q)\} \]

Note that this binding allows us to infer that p and q in some way 'overlap'. This suggests that a full rich semantics of scenarios requires 'events' with duration and that both the 'when' relation and the binding between behaviours needs a set of linear interval relationships such as 'starts before', 'overlaps the beginning' as in [1]. We will not expand on the full semantic model for this here, however because of the separation of the instances, order and values, this extension is not problematic.
3.3 Turn-taking and States

Some models, for example the PIE and related models [12] and action–effect rules [23] have a series of user-action – system-response turns. In the PIE this is modelled as an interpretation function (I) between traces of user commands (from \( P = \text{seq} \ C \)) to system effects (E):

\[ I: \text{seq} \ C \rightarrow E \]

Fig. 2. Behaviour bindings derived from model binding

The unfolding of the system can then be thought of as a turn-taking sequence:

\[ e_0 \rightarrow c_1 \rightarrow e_1 \rightarrow c_2 \rightarrow e_2 \rightarrow c_3 \rightarrow e_3 \rightarrow c_4 \rightarrow \ldots \]

where: \( e_0 = I(\langle \rangle), \ e_1 = I(\langle c_1 \rangle), \ e_2 = I(\langle c_1, c_2 \rangle), \ldots \)

State-transition diagrams and related formalisms have states with transitions labelled by the action that caused them and any system response. That is the formal model they represent is:

\[ \text{trans: } (\text{old}) \ \text{State} \rightarrow \text{Action} \rightarrow \text{Response} \rightarrow (\text{new}) \ \text{State} \]

This gives rise to sequences of the form:

\[ s_0 \rightarrow a_1 \rightarrow r_1 \rightarrow s_1 \rightarrow a_2 \rightarrow r_2 \rightarrow s_2 \rightarrow a_3 \rightarrow r_3 \rightarrow s_3 \rightarrow \ldots \]

where: \( s_0 = \) initial state, \( r_i, s_i = \) trans (\( a_i, s_{i-1} \))

Note, a new state may be reached ‘before’ the response, but persists hence the trace order. If we look only at the externally observable phenomena, this then reduces to an action–response trace:

\[ a_1 \rightarrow r_1 \rightarrow a_2 \rightarrow r_2 \rightarrow a_3 \rightarrow r_3 \rightarrow a_4 \rightarrow r_4 \rightarrow \ldots \]

This can be viewed as an event behaviour in two ways.

The first way, which is probably the most natural semantics for the STN is to regard the action and response as being facets of the same event.

\[ \text{X.instances} = \{x_1, x_2, x_3, x_4, \ldots\} \ \text{X.when} = \{x_1 < x_2 < x_3 < x_4 < \ldots\} \]
X.what = \{ (x_1, 'user', a_1), (x_1, 'sys', r_1), (x_2, 'user', a_2), (x_2, 'sys', r_2), \ldots \}

The second is to regard the action and response as separate events, which is what we would get if we 'translated' the STN into an 'equivalent' process algebra specification.

Y.instances = \{ y_1, y_2, y_3, y_4, \ldots \}  \quad Y.when = \{ y_1 < y_2 < y_3 < y_4 < \ldots \}

Y.what = \{ (y_1, 'user', a_1), (y_2, 'sys', r_1), (y_3, 'user', a_2), (y_4, 'sys', r_2), \ldots \}

Happily these two semantics can be given a binding, so that they can be related to one another in the obvious way:

B_{XY} = \{ (x_1, y_1), (x_1, y_2), (x_2, y_3), (x_2, y_4), (x_3, y_5), (x_3, y_6), \ldots \}

If the process algebra had been derived from the STN, or if we were attempting to demonstrate equivalence, we would have some sort of binding relating the STN as a model and the process algebra as a model. In such cases the binding between behaviours would be able to be derived from the model binding (fig. 2).

3.4 Interstitial Behaviour

If we choose to retain the states in our semantics of the STN we would get a third semantics obtained from the X semantics by inserting extra instances between the action-response instances:

Z.instances = \{ z_1, z_2, z_3, z_4, \ldots \},  \quad Z.when = \{ z_1 < z_2 < z_3 < z_4 < \ldots \}

Z.what = \{ (z_1, 'state', s_0), (z_2, 'user', a_1), (z_2, 'sys', r_1), (z_3, 'state', s_1), (z_4, 'user', a_2), (z_4, 'sys', r_2), (z_5, 'state', s_2), \ldots \}

We can relate this semantics to the X semantics:

B_{XZ} = \{ (x_1, z_2), (x_2, z_4), (x_3, z_6), (x_4, z_8), \ldots \}

Notice that in the relationship between the X and Y semantics we have several instances in the Y behaviour mapping onto a single instance in the X behaviour. In this case, some of the instances in the Z behaviour have no correspondence at all in the X behaviour, they fall in the gaps between events – the interstice (fig. 3).

There are two reasons for instances in one behaviour to fall in the interstice of the other. Consider again a process algebra used to specify the interface. Some of the events would correspond to user actions and observable system responses, others would be purely internal. If we wanted to relate the full trace of events with the externally observable one, the hidden internal events would fall in the interstice.
between the user actions and system responses. These correspond to specific events that happen between the observable events, a form of trace refinement.

In contrast, states are not things that 'happen', but represent something that has a value throughout the interval between the events. That is there is a fundamental ontological distinction between the STN states and events. The former are status phenomena – things which have a continuing value over a period of time as opposed to events which occur at an instant. This status–event distinction and the importance of interstitial behaviour was originally proposed in order to deal with continuous behaviour such as mouse movement [13], but as we see, it makes a difference even to the way we regard discrete phenomena.

### 3.5 Hierarchical and Layered Models

It can be useful to consider some of the internal or structural parts of a model as 'observable' phenomena as we have shown with STN states or hidden events. Many task or cognitive models are based on hierarchies (e.g. GOMS [7], HTA [37], CTT [29]) and traces of unit actions form a bridge with internal system descriptions. It is also good to represent the higher-level tasks or goals, which require a more flexible model of instances, as discussed in section 3.2, because higher-level tasks 'happen' over a period incorporating several lower-level tasks.

![Plan 0: 1 then 2 when kettle boils 3 then 4
Plan 2: in any order 2.1, 2.2, 2.3](image)

**Fig. 4.** HTA of tea making

For example, consider the HTA in fig. 4. Consider now a possible behaviour, call it K, for this task in which the user accidentally adds tea to the pot before warming it:

- K.instances = { k1, k2, k3, k4, k5, k6, k7, ... }
- K.when = { k2 before k3, k3 before k4, ... k9 before k10, k10 before k11, ... }
- K.what = { k1 × <task',0>, k2 × <task',1>, k3 × <task',2>, ... k9 × <task',2.2>, k10 × <task',2.1>, ... }

Notice how k3 is an instance of the higher-level task "2. prepare", which persists over an interval containing k9, k10 and k11 (the sub-tasks 2.2, 2.1 and 2.3).

Slightly different considerations apply to layered architectural models such as Seeheim [33], Arch-Slinky [39] or PAC hierarchies [9].
Consider the dialogue box in fig. 5. The user interacts with the pull-down menu in series of lexical-level events, managed by the platform widget toolkit. Only at the end of this series of lexical events is there a dialogue level event 'new underline style is word'. There may then follow further lexical and dialogue events (select position, menu selection of size), before finally the user clicks on 'OK' giving rise to an application-level event and the selection format is changed (fig. 6).

Note the difference. In a task analysis the user is 'selecting the underline mode' throughout the menu interaction, but when considered in architectural terms the dialogue event happens at the end of the menu interaction. Dialogue events can be seen as coinciding or possibly being 'just after' the final lexical level event of the sequence (the mouse release). These points of synchronisation between different architectural levels are called interaction points [17]. We can regard this either as multiple behaviours with bindings between them at the interaction points, or as a single behaviour with event instances at different levels related by 'coincident with' or 'just after' relations. However, this raises another issue.

We have multiple events at an interaction point – e.g. lexical level "release mouse over 'word'" and dialogue level "underline is word". Depending on your viewpoint or concerns these may be validly regarded as coincident or as following closely after one another. The former view would make sense from a user perspective and the latter if we wanted to trace the flow of events through architectural components.

3.6 Temporal Granularity

This problem of 'coincident' vs. 'just after' when considering interaction points is because different models operate at different temporal granularities. Some relations make sense across different granularities – it would be wrong to have event A be before event B in one model and after in another. However, coincidence should
always be read as 'at the same time at this model's temporal granularity. This is one of the reasons for the weak link between clock time and order in section 3.1.

The fuzzy nature of human time has been emphasised by Payne in his study of calendar use [32]. In a previous DSVIS paper and her thesis [21], Kutar addressed the complex issue of extending temporal logics to express statements, such as 'on the same day', that have both contextual and fuzzy semantics. This work proposed some solutions but also left many open problems. Even in the simpler world of individual behaviours, problems of temporal granularity are far from trivial, but appear more tractable. Better understanding the meaning of temporal behaviours may give some insights into the more demanding world of temporal specification.

3.7 Real Time

Most models of discrete UI behaviour take place in a task-paced world timed by significant events rather than the ticking of the clock. There are exceptions to this. Temporal extensions to many notations allow one to express the time between events. This can be modelled by the 'at' time discussed in section 3.1, but this forces a global time on everything. For some models it may be more appropriate to use an enrichment of the 'when' relation between events to be able to say things like A happens sometime before B and B happens at least 3 seconds before C.

The /g116-PIE in "The myth of the infinitely fast machine" [10] embodies the external flow of time very directly in the use of 'ticks' to represent moments where there is no user activity. This can be given a semantics in terms of the behaviours in two ways. One way is to use the same 'tick' of instances where the 'what' is either empty or a special 'nothing happens' kind. Alternatively, we can only have instances for actual user actions or system effects and use the 'at' to give real times to events. These two semantics can be mapped to one another, so the choice is a matter of convenience.

4. Continuous Behaviour

We now move on to continuous behaviour. This discussion will be more limited for several reasons: partly because there are fewer models of continuous interaction and partly because the world really is continuous so there are less 'choices' about models. We will also find that many of the issues of continuous interaction have been prefigured by discrete issues, so can be dealt with quite succinctly.

4.1 Models of Continuous Interaction

Although many of the systems and interfaces studied in rich-media and in novel interfaces embody continuous real-time interaction, there are few models of this in the HCI literature. Possibly this is because of the conceptual dominance of discrete models. We start or analysis of continuous interaction by reviewing these models.

Probably the earliest continuous time models in the formal UI literature are the variants of status–event (S–E) analysis [13,14], and we have already seen a hint of this in discrete systems. This distinguishes events, which occur at specific moments of time, from status phenomena, which have (typically changing) values over a period of time. Examples of events include keystrokes, beeps, and the stroke of midnight in Cinderella. Examples of status phenomena include the current display, the location of the mouse pointer, the internal state of the computer and the weather.
Perhaps one of the most significant features of S-E is its treatment of interstitial behaviour. Whereas discrete models focus on the moments when events occur, S-E puts equal emphasis on the more fluid interaction between events. In many GUI systems this is what gives the 'feel' of interaction: dragging, scrolling etc., and in rich media this is likely to be the main purpose of interaction!

Status–event analysis is really a conceptual framework for viewing interaction, but does have several concrete models, both descriptive (variants of the PIE) and specification notations. These all have the general form of a state-transition style description of events and more continuous description of interstitial behaviour:

**action:**

user-event $\approx$ (current/history of) input-status $\approx$ state

$\circ$ response-event $\approx$ (new) state

**interstitial behaviour:**

(cURRENT/history of) input-status $\approx$ state $\circ$ output-status

The treatment of the input status at events and during interstitial behaviour distinguishes interactions of markedly different kinds: trajectory independent interactions that only depend on the current state (such as where the mouse button is when it is clicked) vs. trajectory dependent interactions where the complete status history matters (such as freehand drawing).

Another crucial aspect of S–E, not apparent in discrete system, are status-change events. These occur when a status phenomena crosses some trigger threshold, for example, a target temperature or particular time (on the clock). The nature of threshold is application dependent and may be dynamic. Furthermore there issues concerning how status–change events become system-level events: polling, active sensors, etc. At an abstract level of specification one would just say "when this happens ...", but as this becomes operationalised issues of mechanism surface.

In modelling virtual reality systems, Wüthrich [41] made use of cybernetic systems theory using formulae of the form:

$$state_t = \phi \left( t, t_0, state_{t0}, \text{inputs during } [t_0,t) \right)$$

$$output_t = \eta \left( state_t \right)$$

together with appropriate consistency conditions.

The difference between this and the S–E description is largely representational, although the 'state' is different. In the S-E description the 'state' ignores ephemeral changes during the interstices, whereas the systems theory model incorporates this (for example, that the mouse is at a particular position now). Both forms of state can be useful and can be incorporated together as alternative views or levels of description.

As a personal preference I regard the systems theory form as a potential replacement for the interstitial behaviour, but would retain the distinct action transition as this accords more closely with the way we tend to informally describe systems. However, both forms of expression should be accommodated within a ubiquitous theory.

Notice that the systems theory definition treats events on a par with status phenomena, both are merely part of an 'inputs at time t'. However, when viewed as functions of time, event phenomena have values only at a few points of time. There is a history of dealing with event phenomena in this way in applied mathematics where 'impulse' events, such as a voltage spike or collision, are modelled using Dirac delta functions (infinitely large for infinitesimally short time). In fact, when applying
systems theory to a real example Wüthrich finds it more convenient to describe event phenomena as time-stamped values.

The TACIT project [38] studied a variety of continuous time interactive systems and also considered modelling using hybrid high-level Petri-Nets [24]. In common with many models from the hybrid systems community [19], this takes a largely dualist model of the world with discrete computer systems interfacing with a continuous environment defined by differential equations. The formal expression of this involves 'continuous transitions' – a form of interstitial behaviour.

A component/interactor model based on a similar variant of hybrid Petri-nets is proposed in [40] to deal with objects in virtual environments. Its 'world object state' has both discrete and continuous output and input (fig. 7) and a threshold on continuous input, producing status-change events. Less clear from the diagram above, but evident in the Petri-net specifications, is the direct connection between continuous input and output (status–status relations) controlled by the enable/disable switch.

Note that it is quite easy to describe these models in similar terms because they sit closer to the real world than in the discrete case. For behaviour modelling we need only extend the EventInstances to include values that are functions of time (no longer 'EventInstances', just Instances). This can then represent periods where the mouse position moved over a trajectory, or the window tracked the mouse position.

4.2 Granularity and Two Timing

Granularity effects resurface in continuous time and are perhaps more perplexing. We still want to be able to talk about events: the button was clicked, the computer beeps, I had a meeting. It is right to represent these as events, but clearly when looked at closely they take some duration. We can allow many-to-one bindings just as we do in the discrete case. However, this appears a little odd where 'many' is a period with continuous motion (for example, if select menu item at one level became the actual mouse movements at another). In fact, this is not uncommon in applied mathematics where solutions of equations with slow time-varying parameters or small non-linearities are managed by imagining two timescales, one infinitely fast with respect to the other. A technique called two-timing.

A less drastic approach, which captures the finite difference in granularity, is to have bindings which both preserve order and also have bounded 'jitter'. That is, there is a maximum 'jitter' time, $\delta$, such that if event A and B are separated by time $t$ in one behaviour their separation in the other behaviour is in the range $[t-\delta,t+\delta]$.
4.3 Temporal Gestalt Phenomena

The issues of hierarchy and architectural levels are similar in the continuous case to the discrete one. However, continuous media exhibit a special case of the lexical-dialogue interaction points – temporal gestalt phenomena. Consider gesture recognition – the movement only has meaning once completed, music – the melody is only a melody whilst being played, and even graphics get meaning through movement [5]. This is rather like the fact the typed characters "d", "i", "r", return only mean "list directory" when considered as a whole, but the boundaries are clearer in the case of discrete phenomena – the gesture has no definitive start or end although an endpoint would be operationalised by actual recognition software.

5. Summary

This paper has considered the semantics of traces of user-system activities in both discrete and continuous interaction. One might have thought that this would have been a trivial task as all models in some sense are describing 'the same thing', but as we have seen the multiplicity of viewpoints and levels of analysis mean that even establishing commonality at the level of sequences of actions is surprisingly complex. The paper does not present a final or complete semantics, but does demonstrate how common behavioural semantics may be possible for a range of common notations.

We have also seen a number of problematic issues for formal treatment of scenarios, especially when we try to align scenarios corresponding to different complementary models. This is because notations differ in which events and actions they regard as atomic, or instantaneous and much of our discussion has been about how to establish bindings between these different accounts.

Another issue that has arisen in both discrete and continuous models is the ontological distinction between status and event phenomena and the important role of interstitial behaviour.

The word 'theory' comes from two Greek words thea, "outward appearance", and horao, "to look closely" [18]. This paper has been examining the surface of interaction and it is hoped that by following this through in detail we can work towards a theory of multi-paradigm interaction modelling.

Acknowledgements

This work was supported by the EPSRC funded project DIRC.

References


Architecture considerations for interoperable multi-modal assistant systems

Thomas Heider and Thomas Kirste
Fraunhofer-Institut für Graphische Datenverarbeitung Rostock
{Thomas.Heider,Thomas.Kirste}@rostock.igd.fhg.de

Abstract. Creating multimodal assistant systems supporting the intuitive interaction with technical infrastructures of the everyday life is one important goal of current HCI research. Building such systems is a substantial challenge – not only with respect to the individual concepts and algorithms that are required at the various levels of multimodal interaction processing, but also with respect to the overall system architecture. Especially, when we try to address systems that can be extended dynamically and that are not built by a single vendor.

We will look at some the challenges of creating architectures for such systems and we will outline the solution approach our research group is currently developing. The work we present is part of the EMBASSI-project, a joint project with 19 partners from industry and academia that aims at establishing an interoperable system infrastructure for multimodal and multimedia assistance systems.

1 Overview: the EMBASSI project

EMBASSI\textsuperscript{1} is a focus project supported by the German Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF) within the strategic research area Man-Technology-Interaction. With 19 partners from industry and academia and a time scope of four years, EMBASSI intends to provide an integrated approach to the development of assistants for our everyday technologies.

The primary application area for EMBASSI are technical infrastructures of the non-professional everyday life – in particular, application scenarios are being developed in the home, automotive, and public terminals environments.

EMBASSI is conceptually based on two important paradigm shifts:

- Transition from essentially unimodal, menu-based dialogue structures (with a fixed interaction vocabulary provided by the system) to polymodal, conversational dialogue structures (with an unrestricted interaction vocabulary provided by the user).
- Transition from a function-oriented interaction with devices to a goal-oriented interaction with systems.

While these paradigm shifts are being discussed in the research community for some time now, it is a substantial challenge to make these results accessible to the user of, e.g., home entertainment infrastructures. This is the goal of EMBASSI.

\textsuperscript{1} “EMBASSI” is a German acronym for “Multimodal Assistance for Infotainment & Service Infrastructures”
Building such systems is a substantial challenge – not only with respect to the individual concepts and algorithms that are required at the various levels of multimodal interaction processing, but also with respect to the overall system architecture. Especially, when we try to address systems that can be extended dynamically and that are not built by a single vendor.

In this paper, we will look the challenges of creating architectures for such systems and we will outline the solution approach we are using and developing within the EMBASSI project.

The paper is further structured as follows: Section 2 gives an overview over the challenges of self-organizing multi-modal multi-agent systems. In Section 3, we outline the architectural framework used in EMBASSI. In Section 4, we describe the underlying concepts of the middleware currently being developed in EMBASSI. We relate our work to other approaches in Section 5 and look at future work in Section 6.

2 Middleware challenges

A central requirement for an EMBASSI architecture is that it should support technical infrastructures that are built from individual components in an ad hoc fashion by the end user. This situation is for instance common in the area of home entertainment infrastructures, where users liberally mix components from different vendors. Also, it is not possible to assume a central controller – any component must be able to operate stand-alone. Furthermore, some infrastructures may change over time – due to hardware components entering or leaving the infrastructure or due to changes in the quality-of-service available for some infrastructure services, such as bandwidth in the case of wireless channels.

Therefore, such an architecture should meet the following objectives:

- Ensure independence of components,
- Allow dynamic extensibility by new components,
- Avoid central components (single point of failures, bottlenecks),
- Support a distributed implementation,
- Allow flexible re-use of components,
- Enable exchangeability of components.
- Provide transparent service arbitration.

When interacting with their personal environment, users may be driving car, enjoying a TV show at home, calling a colleague over the mobile phone, etc. These very different situations do not only influence the assistance strategies provided by the conceptual architecture’s components – they also have a strong impact on the hardware infrastructure available for implementing the assistant system. It becomes clear that a broad range of different hardware infrastructures has to be considered as implementation platform – for example:

- mobile personal communicators with wireless access to stationary servers,
- wearable computers using augmented reality displays for interaction,
- a set of networked consumer electronic components, without a central controller,
– the local information network of modern cars,
– the user’s PC at home, communicating with globally distributed information servers,
– public terminals, etc.

From these considerations, substantial challenges arise with respect to the software infrastructure that is required for implementing the conceptual architecture. It needs to support functions such as:

– Distributed implementation of components. As soon as more than one host is available (or required) for implementing the architecture, a distribution scheme must be developed. The distribution scheme may either simply allocate different functional components on different hosts (relying on the assumption that inter-component communication is less frequent than intra-component communication) or it may distribute individual components across multiple hosts (making each component virtually available everywhere, but creating challenges with respect to managing a consistent internal state). Clearly, the right choice depends on the concrete infrastructure that is available.

– Communication mechanisms. Once a distributed implementation is considered, the choice of communication concept is no longer a matter of taste. Distributed shared memories or distributed blackboards for example are a much more heavyweight communication scheme than message passing and bus architectures – but simplify communication design for knowledge based systems. Again, the right choice cannot be made without considering the concrete infrastructure, the specific communication needs of the components in question, and the distribution model.

– Ad-hoc discovery of system components. In some infrastructures, new components may join an existing system in an ad-hoc fashion. Consider, e.g., a personal communicator establishing contact to a point of sales terminal, where both components are equipped with their own version of the assistance system. Both systems must be able to integrate with each other and discover each other’s components and functionalities, in order to provide an interoperable service (such as using the mobile communicator’s input and output devices for the stationary terminal’s analysers and dialogue management).

– Ad-hoc (re-) distribution of software components. In case the infrastructure changes during system life, it may become necessary to adapt the distribution scheme towards the new resources. It may even be of interest to change the allocation of software components in an ad-hoc fashion – e.g., by using mobile agents.

3 The EMBASSI architecture

The generic architecture that we have developed within EMBASSI (Fig. 1) is a pipeline approach to the problem of mapping user utterances to environment changes. Each “level” in the architecture represents one function within this pipeline, while the level interfaces have been introduced at “meaningful” places, separating different ontologies. These “ontologies” (the sets of objects that are discussed at a level) become visible at the level interfaces. The level interfaces make up the EMBASSI protocol suite.
Each level consists of a number of processes ("components") that co-operatively implement the level’s function. Processes can be added or removed dynamically: suitable co-ordination mechanisms at each level are responsible for managing the interactions between the processes at this level. There is deliberately no central co-ordination component (see Section 4 for further details on component co-ordination).

The use of this rather fine-grained level-model in conjunction with the feature of dynamically adding or removing processes at each level allows us to create systems that can be incrementally built and extended in an ad hoc fashion, using modular components. Specifically, it allows us to build interoperable systems, where different components are provided by different vendors and where components are added and removed over time by the end-user.

Also, this allows us to collect components in a “technology toolkit”, from which specific assistant systems can be built by simply “plugging” these components together.

### 3.1 The MMI levels

An **EMBASSI** system has to accept multimodal utterances which it needs to translate into goals before it can begin to think about changing the environment. According to the architecture in Fig. 1, this translation process can be broken down into three distinct steps:

1. First we translate physical interactions into atomic interaction events (lexical level). The transformation of physical user interactions into unimodal atomic events is done by the \( I \) components (\( I = \text{Input} \)).

---

2 The reader should note that this three-level approach is a rather straightforward adoption of the **LANGUAGE** model described by Foley and Van Dam [3].
2. The stream of atomic interaction events is then sent via the Event interface to the F components (F = Filter). These components are responsible for inter- and intra-modal aggregation of atomic events into amodal sentences (syntactical level).

3. The stream of sentences arrives at the D components (D = Dialogue manager). D components are responsible for translating sentences into goals denoted by these sentences (semantical level). Also, D is responsible for managing inter-sentence relations (dialogue memory) and dialogue dynamics (such as turn-taking).

The process is reversed when producing output: D sends amodal output “sentences” to the R components (R = Renderer) which in turn map these sentences to multiple atomic output events for the available output channels, the O components.

3.2 The assistance levels

The assistance levels operate on goals which are identified by the MMI levels. The process of mapping goals to changes of the environment consists of the following steps:

1. A components (A = Assistant) take goals (which specify state changes of the environment) and try to develop strategies for fulfilling these goals (strategy level).

   These strategies are called plans.

   There is no predefined way to produce a plan. Some A components may use hard-wired plans, others could use decision trees or even complete inference systems.

2. The plans are sent to the X components (X = eXecution control), which are responsible for the (distributed) scheduling and execution of the plans. (This means, the EMBASSI-architecture advocates a two step planning policy, as described e.g. in [15], where strategy planning (A-level) and execution scheduling (X-level) are distinct processes.)

   The scheduling-components ensure the correct sequential processing of the individual steps in a plan. Also, they are responsible for managing the parallel execution of multiple plans and for the management of execution resources needed by the plan.

3. Finally, individual action requests are sent to the (abstract) devices (device control level), the G components (G = Gerät – German for “device”). The G components execute the action request by employing the physical device they control, thus changing the state of the environment and causing an effect as intended by the user.

   The Context Manager C is responsible for managing the system’s view of the world – information about the user, resource profiles, the environment, and also the availability and state of the individual EMBASSI components. Attached to the context manager, we have sensors to obtain biometrics and environmental information – such as the current position of the user or the ambient light level. The context manager supports both pull interface (queries) and a push interface (notifications) to the information it stores.

   Finally, based on a self-description deposited by a G-component in the context manager, a Generator may be able to automatically create simple A and D components for it. See [7] for further details on this concept.
3.3 Additional notes on the generic architecture

Before addressing the main point of this paper – the middleware infrastructure that is required for building dynamically extensible interactive assistance systems based on the EMBASSI architecture – a few additional aspects of this architecture should be noted:

– The above generic architecture has been instantiated for the various application scenarios investigated in EMBASSI – home control, car infotainment, and point-of-sales / point-of-information terminals. The instance architecture for home control is shown as example in Fig. 2.

– Conventional widget-based user interfaces are quite easily mapped to the EMBASSI architecture: within the Home Control scenario, they correspond to the GUI Analysis / GUI Renderer components.

– The feedback loops within an EMBASSI level – e.g., the connection between GUI Input and GUI Renderer are not explicitly shown in our architecture. Such feedback loops clearly do exist – but we have not yet ventured into designing a generic ontology for these them. Hence, they are not yet part of the EMBASSI protocol suite.

– It is of course possible to map the EMBASSI architecture to existing architecture models for man-machine interfaces such as the ARCH model [1, 9]. In ARCH, the levels that map user interaction to system actions are: Interaction Toolkit, Presentation Component, Dialogue Component, Domain Adaptor Component, and Domain-specific Component. Without going too much into the details, one possible mapping of EMBASSI to ARCH would be: I/O-Level = Interaction Toolkit, F/R-Level...
= Presentation Component, D-Level = Dialogue Component, A-Level = Domain Adaptor Component, X & G-Level = Domain-specific Components.

In this context, the reader should be aware of the fact that it is not the primary goal of EMBASSI to present yet another model for interactive architectures. Rather, we try to build an infrastructure that allows the dynamic composition of interactive systems from components in an ad hoc fashion. It is the dynamic composition functionality that is the “interesting” aspect in EMBASSI – at least at the architectural level.

Next, we will look at the middleware model that we use for implementing the dynamically extensible architecture of EMBASSI.

4 The middleware model

The goal of our middleware-work is to develop a system model that provides the essential communication patterns of a data-flow based multi-component architectures such as EMBASSI. At the same time, we also want to have an experimental platform implementing this model (a reference implementation) that allows to quickly build and empirically verify experimental applications – specifically with respect to functions such as service arbitration.

Here, the focus of empirical studies is the way systems can be build dynamically from individual components and how the mechanisms provided by the model are used for building such systems – a software engineering focus.

The model should have the following properties:

- Support data-flow based event processing topologies.
- Support conventional remote procedure calls.
- Support self-organization of system components.
- Support decentralized problem decomposition and conflict resolution (service arbitration).
- Support dynamic extension by new components.
- Support unification / partitioning of complete system topologies.

The model we have developed so far is called SODA-Pop (for: Self-Organizing Data-flow Architectures suPporting Ontology-based problem decomPosition). Following, we give a brief overview over the salient features of this model.

4.1 Component types

The SODA-Pop model [8] introduces two fundamental organization levels:

- Coarse-grained self-organization based on a data-flow partitioning.
- Fine-grained self-organization for functionally similar components based on a kind of “Pattern Matching” approach.

Consequently, a SODA-Pop system consists of two types of components:
Channels, which read a single message at time point and map them to multiple messages which are delivered to components (conceptually, without delay). Channels have no memory, may be distributed, and they have to accept every message. Channels provide for spatial distribution of a single event to multiple transducers.

The interface buses of the EMBASSI architecture are channels.

Transducers, which read one or more messages during a time interval and map them to one (or more) output messages. Transducers are not distributed, they may have a memory and they do not have to accept every message. Transducers provide for temporal aggregation of multiple events to a single output. Note that a transducer may have multiple input and output channels (\(m : n\), rather than just \(1 : 1\)). The \(I, F, \ldots\) components of EMBASSI are transducers.

The criterion for discriminating between transducers and channels is the amount of memory they may employ for processing a message – i.e., the complexity they create when trying to implement them in a distributed fashion: Channels may use no memory. This requirement clearly makes sense when considering that we may want to use channels as “cutting points” for distributing a system: Implementing distributed shared memory is expensive. Communication primitives for potentially distributed systems therefore should not provide such a facility “for free”. In addition, the “‘No Memory” Constraint provides a hard criterion for discriminating between the functions a channel is allowed to provide and functions that require a transducer.

Finally, it becomes obvious that persistence functionality (such as provided by blackboard-based communication infrastructures, e.g. LINDA [4] or FLiSiDE [13]) shall not be part of a channel, as persistence clearly violates the concept of memory-free channels.

4.2 Channels & systems

Channels accept (and deliver) messages of a certain type \(t\). Transducers map messages from a type \(t\) to a type \(t'\). A system is defined by a set of channels and a set of transducers connecting these channels. So, a system is a graph where channels represent points (nodes) and transducers represent edges\(^3\). Channels and transducers are equally important in defining a system – a minimal complete EMBASSI system for example consists of 10 channels and 10 transducers (9 and 9 if sensors are omitted).

Channels are identified via Channel Descriptors. Conceptually, channel descriptors encode the channel’s ontology (the meaning of the messages), so that transducers can be automatically connected to channels that speak the languages they understand.

4.3 Communication patterns

The middleware for multimodal event processing and multi agent approaches should support at least the following two communication patterns:

- **Events** that travel in a data-flow fashion through the different transducers. When an event \(e\) is posted by a transducer \(t\), it \((t)\) does not expect a reply. Rather it expects

\(^3\) Rather: a multigraph, because we may have several edges connecting the same two nodes.
that other system components (i.e., the called transducer) know how to continue with processing the event.

- **RPCs** that resemble normal remote procedure calls. When a RPC is called by a transducer, it expects a result. Here, the calling transducer determines the further processing of the result.

Events and RPCs describe different routing semantics with respect to result processing. When considering the EMBASSI architecture, the flow from I to G is a typical event processing pipeline, where at each level we have a set of transducers that cooperate in order to translate an event received at the input (upper) level into an event posted at the output (lower) level.

Event- and RPC-like result routing semantics correspond to different types of channels, a transducer may subscribe to. Event- and RPC-Channels are the two basic channel types provided by SODA-Pop.

With respect to events, there is one important additional requirement: In the normal course of action, events are delivered by a *push* mechanism, initiated by the producer. However, there are also situations when the consumers need to *pull* events – here, event delivery is initiated by the consumers. One specific instance of this pull situation arises when the transducers receiving an event need to *ask back* to the producing level for further information that may be needed to understand the event (e.g.: $D$ may ask back to $F$ for further multimodal event information it may need to disambiguate a given user utterance). So each event channel implicitly contains an inverse RPC channel on which an event-pull can be performed.

### 4.4 Subscriptions

Events and RPCs are (in general) posted without specific addressing information: in a dynamic system, a sender never can be sure, which receivers are currently able to process a message. It is up to the channel on which the message is posted to identify a suitable message decomposition and receiver set (service arbitration).

A channel basically consists of a pipe into which event generators push messages (events or RPCs) which are then transmitted to the consumers (transducers) subscribing this channel. When subscribing to a channel, an event consumer declares:

- the set of messages it is able to process,
- how well it is suited for processing a certain message,
- whether it is able to run in parallel to other message consumers on the same message,
- whether it is able to cooperate with other consumers in processing the message.

These aspects are described by the subscribing consumer’s *utility*. A *utility* is a function that maps a message to a *utility value*, which encodes the subscribers’ handling capabilities for the specific message. A transducer’s utility may depend on the transducer’s state.

The definition for Utility values in SODA-Pop is:

---

4 The current version of SODA-Pop is defined in Haskell [12], the current “standard” functional language.
type Quality = Int -- just as example
data UtVal = NotApplicable -- Can’t handle msg
| Exclusive Quality -- Expect to handle it exclusive
| Nonexclusive Quality -- Don’t mind if others are involved
| Cooperative [(Quality,Msg)] -- Can do some parts, but need help

And a simple transducer that is able to handle only a single kind of message \(m_0\) might provide a utility function such as

\[
\text{isForMe} :: \text{Msg} \rightarrow \text{UtVal} \\
isForMe m | m == m_0 = \text{Nonexclusive} 0 \quad \text{-- if } m \text{ is } m_0 \\
| \text{True} = \text{NotApplicable} \quad \text{-- otherwise}
\]

The Cooperative value needs further explanation: with this utility value, a transducer may return a list of partial messages it is able to handle, together with a quality value for each sub-message. This gives the Channel the opportunity to select the best tradeoff for decomposing a message across multiple transducers\(^5\).

### 4.5 Message handling

**Receiver selection & message decomposition.** When a channel processes a message, it evaluates the subscribing consumers’ handling capabilities and then decides which consumers will effectively receive the message (receiver set). Also, the channel may decide to decompose the message into multiple (presumably simpler) messages which can be handled better by the subscribing consumers. (Obviously, the consumers then solve the original message in cooperation.) The basic process of message handling is shown in Fig. 3.

*How* a channel determines the effective message decomposition and how it chooses the set of receiving consumers is defined by the channel’s *decomposition strategy*.

Both the transducers’ utility and the channel’s strategy are eventually based on the channel’s ontology – the semantics of the messages that are communicated across the channel.

For some channels, the concept of cooperative message processing may already be a part of the channel’s ontology. This means that the channel’s language contains a means for embedding synchronization statements into a (presumably compound) message – such as “wait for completion of sub-request \(i\)” and “announce completion of sub-request \(j\)”. The channel’s strategy then embeds suitable synchronization statements into the messages it creates for the receiver set. Corresponding announcements are to be exchanged over a synchronization channel that needs to be established between the receiver set. This mechanism is used in EMBASSI for distributing the execution of strategies computed by the \(A\) level across multiple scheduling components at the \(X\)-level. (Note in this context that *temporal scheduling* is *not* a channel function as it clearly requires memory in the channel.\(^5\))

**Reply recombination.** A channel’s strategy may also describe the method for how to assemble the reply messages created by cooperating (or parallel) message consumers

\(^5\) This is a rather experimental feature.
into a single aggregated reply. This strategy describes how to wait for the different transducers that have received (partial) messages and what algorithm to use for aggregating these replies. The most simple approach is to just return the first reply that is received. Another simple approach is to wait for all results, put them into a list, and return this list.

Unfortunately, it requires memory to perform this reply recombination: the component responsible for recombination has to remember from whom it can expect replies and which replies it already has received. Therefore, this component can not be handled by a channel. Instead, the channel creates an implicit transducer that performs the reply recombination strategy. By factoring reply recombination out of the channel, the design choice of where to do recombination in a distributed environment (at the receiver side? at the processing side?) becomes explicit, while at the same time keeping channel functionality lean: The channel may decide where to place the recombination transducer – but it does not have to implement its memory functionality.

Note that by putting decomposition and recombination into the channel rather than leaving this to the requesting component, we ensure that message decomposition and reply recombination is transparent to a component. This has two effects:

- Component designers are relieved from the task of doing receiver selection and reply recombination, this greatly simplifies implementation.
- The danger of misbehaved components that always select the same kind of receivers (i.e., only receivers from the same vendor . . . ) is minimized.
5 Related work

SODA-Pop is not the first concept for addressing the problem of dynamic, self-organizing systems. Other approaches are for example HAVi [5] and Jini [17], the Galaxy Communicator Architecture [14, 11], and SRI’s OAA (Open Agent Architecture) [16, 10], where specifically Galaxy and OAA intend to provide architectures for multi-agent systems supporting multi-modal interaction.

Compared to the state of the art, the pattern-matching approach in SODA-Pop itself is not new. Comparable concepts are provided by Galaxy, by SRI’s OAA, as well as by earlier work on Prolog [2] and the Pattern-Matching Lambda Calculus [18]. Here, SODA-Pop simply intends to provide a certain refinement at the conceptual level by replacing language-specific syntactic pattern-matching functionality (such as the Prolog-based pattern matching of OAA) by a language-independent facility based on utility value computation functions that are provided by transducers.

The important differences of SODA-Pop to the above approaches are

- SODA-Pop uses a two-stage approach to system decomposition and self organization. Coarse-grained structuring is provided by defining channels, fine-grained structure is supported by “pattern matching”.
- SODA-Pop supports data-flow architectures by providing event channels besides conventional RPC channels.

The combination of these two approaches is an important extension over the above systems.

HAVi, Jini, OAA, and Galaxy all provide a single mechanism for message routing. In HAVi and Jini, we have a simple event subscription mechanism on a global bus. Furthermore, Havi and Jini both do not provide transparent service service arbitration. OAA basically provides a single SODA-Pop RPC channel with a Prolog-based decomposition and recombination strategy. Galaxy provides a centralized hub-component, which uses routing rules for modeling how messages are transferred between different system components. Galaxy too can be modeled by a single SODA-Pop RPC channel that uses a decomposition approach built on top of Galaxy’s frame language.

On the other hand, both Galaxy and OAA could be used to model SODA-Pop simply by representing channels with message tags. (Galaxy and OAA both use heavy-weight routing components that incorporate arbitrary memory and are therefore not suited for a distributed implementation – but this is a different issue.)

So the question is not so much which approach is more powerful, but rather: which approach provides those abstractions that best help to structure a system architecture. Specifically, SODA-Pop aims at supporting systems that are created by multiple (e.g., 19) partners in parallel.

To our experience it is dangerous to provide only a single granularity for decomposing a complex system structure such as EMBASSI. The single granularity necessarily has to be fine in order to provide the required flexibility. When trying to fix the overall structure of the system, such a fine granularity provides too much detail and quickly leads to a proliferation of interfaces that are shared by only a few components. This danger specifically exists, when the interface discussion is carried out by several project...
partners in parallel. However, the proliferation of interfaces is a Bad Thing, because it obstructs the interoperability of system components – a prime goal of EMBASSI.

The SODA-Pop approach provides abstractions that allow a top-down structuring of the system (channels) as well as a bottom-up structuring (within-channel decomposition). In addition, it explicitly includes a data-flow based mechanism for constructing systems out of components, based on SODA-Pop Event Channels. As a design paradigm, the SODA-Pop approach has already been used successfully in implementing the EMBASSI demonstrator systems.

6 Summary and outlook

6.1 What has been achieved so far

In this paper, we have outlined the architecture of a multi-agent system that supports multimodal interaction with technical infrastructures of the everyday life – the EMBASSI architecture. Furthermore, we have outlined the underlying middleware mechanisms, the SODA-Pop model, that provides the essential communication patterns of a data-flow based multi-component architectures such as EMBASSI.

The SODA-Pop model defined so far contains the following properties:
- Support data-flow based event processing topologies.
- Support conventional remote procedure calls.
- Support self-organization of system components.
- Support decentralized problem decomposition and conflict resolution (transparent service arbitration).
- Support dynamic extension by new components.

The aspect of dynamic unification / partitioning of complete system topologies has not yet been integrated, but should be comparatively straightforward based on the current definitions.

The SODA-Pop infrastructure is currently implemented using the functional language Haskell [12]. This implementation serves as a proof of concept of SODA-Pop, as well as a testbed for experimenting with different data-flow topologies and alternative channel ontologies (e.g., decomposition and recombination strategies).

EMBASSI system prototypes have been built for the application areas of consumer electronics & home control, car infotainment, and point-of-sales / point-of-information terminal access. The complete system has been on display at different conferences and fairs, such as the International Telecommunication Exhibition (IFA 2001) in Berlin.

6.2 Next steps

Currently, we work on making SODA-Pop available in a more conventional language (\textit{i.e.}, Java). The goal is here to use SODA-Pop in EMBASSI not only as research prototype for experimenting with self-organizing systems and as design paradigm, but also to use it as foundation for the next EMBASSI incarnation.

\footnote{Systems with a similar scope as EMBASSI are known to the authors that implement well above 100 interfaces, based on a single structuring mechanism.}

\footnote{The source code of SODA-Pop is included in [8].}
Also, work on SODA-Pop is far from complete. Amongst others, the following aspects remain to be investigated in the future:

**Temporal patterns.** The definition of SODA-Pop currently is more or less elaborate with respect to decomposing an event into a set of sub-events that is to be distributed to a set of receivers (spatial decomposition). However, there is currently no comparable mechanism for describing the aggregation of several events into one compound event – as is required for a simplified definition of transducers that are doing temporal aggregation. The concept of recombination strategies is just a first step in this direction. Most notably, temporal aggregation could be described by such things as state machines, petri nets, or by parsers.

**QoS guarantees.** Currently, SODA-Pop provides no mechanisms for specifying and verifying QoS properties such as:

- **Local QoS:** e.g., a reply with a certain minimum precision is guaranteed to arrive within a given time interval. Channels provide currently no mechanism for describing QoS guarantees. The minimum would be that an answer – without a guarantee on its precision – is being made available after a given time. This could be achieved by incorporating strategies such as successive refinement into reply recombination.
- **Global QoS:** e.g., the current set of transducers and channels fulfills a certain data-flow topology (i.e., for each required system function, at least one transducer is available).
  
  There is currently no mechanism defined for making global statements about a set of channels and transducers. These statements could contain both constraints on the topology of the channel/transducer network as well as constraints on their temporal behavior. Although technologies for specifying and verifying such properties exist (e.g., temporal logic, petri nets, process calculi, . . .), it has not yet been investigated, which of these technologies suits best the needs of SODA-Pop and how they can be integrated into an essentially decentralized system concept.

**Concrete channel ontologies.** Finally, the definition of concrete channel ontologies for the EMBASSI infrastructure is an important item of future work – after all, the need for transparent and self-organizing service arbitration in EMBASSI has been one of the main motivations for developing SODA-Pop. The focus here is currently the D → A channel: how do we automatically select between different A components that all claim to be able to solve a specific user goal detected by a D component?

## 7 Acknowledgements

The work underlying this project has been funded by the German Ministry of Education and Research under the grant signature 01 IL 904 G 0. The content of this publication is under sole responsibility of the authors.

---

8 A recombination strategy exactly produces such a temporal aggregation.
References

Model-based approach to control over concurrency in interactive CSCW applications

Yannick Bouillon¹, Fabrice Wendling²

¹ Atlantide, Centre Affaire Oberthur, 74D Rue de Paris, 35000 Rennes, France
   yannick.bouillon@ago.fr

² Laboratoire de Traitement du Signal et de l’Image, INSERM - Université de Rennes 1,
   Campus de Beaulieu, 35042 Rennes Cedex, France
   fabrice.wendling@univ-rennes1.fr

Abstract. In this paper, the problem of event collision in CSCW interactive applications is addressed and a model-based approach (Petri Net) is proposed to control over concurrency. The approach is used in the development of two telemedicine applications which use and reliability level are enhanced. This approach may be applied to a large set of interactive collaborative applications as well as in the control of user interfaces.

1 Introduction

Computer-supported synchronous cooperative work applications (CSSCW) are a particular class of distributed interactive multiuser systems that offer users the opportunity to work together and remotely, at the same time (synchronous work) and on the same multimedia data (text, video, signal, etc.). Thanks to recent advances in the field of telecommunications, disciplines like medicine, teaching and research (from a user point of view), have been actively taking part into CSSCW applications. From a theoretical viewpoint, these applications must address several issues, mainly state consistency, synchronization, participant awareness, latecomers management and event collision [1]. State consistency, also called dynamic uniformity [2], is maintained when any event (user-action) performed by any client is performed at all clients. This study mainly focuses on event collision that remains an open problem in CSSCW applications. It occurs when two or more client applications become concurrent in accessing the same shared resource at the same time. Concurrency may take several forms depending on the type of collaborative application. Consequently, it must be dealt with at the application level to avoid catastrophic consequences such as deadlock situations, well known in the domain of distributed systems. They may occur in synchronous collaborative architectures when two or more users are simultaneously performing an action on their computer interface. Such situations have already been identified by several authors working on CSCW. Readers may refer to Greenberg & Marwood [3] or Munson & Dewan [4] for a presentation of the problem of concurrency. Among proposed solutions, most are based on user- or object-locking mechanisms [5][6] (typically “master/slave”) that do not favor the natural use of...
collaborative software applications. This paper presents a new model-based approach (Petri Net, or PN) to control over concurrency and solve the problem of event collision in CSSCW interactive applications, without using restrictive locking techniques. In this approach, a first PN model of a supervisor-based collaborative architecture is first studied on a theoretical basis using standard PNs analysis methods (structural and enumerative). From derived theoretical results, a second model in which deadlocks are avoided and state consistency is maintained is proposed. This model is further used in the development of two telemedicine applications.

2. Methods

2.1 PN Supervisor-based architecture model

For many years, PNs formalisms [7][8] have been used to model and analyze distributed systems. In the present work, we used PNs to model the behavior of a supervisor-based architecture in which the role of the supervisor application is to maintain state consistency and synchronization between all client applications: any action performed on one application participating the collaborative session is broadcasted, via the supervisor, to all other applications.

A first model of event broadcasting was proposed and studied under various conditions. To formally analyze structural and dynamical properties of this initial model, structural analysis was first applied to identify sequences of transitions bringing the PN back to its initial state (re-initiability). Second, enumerative analysis was applied to identify potential situations of deadlock. Structural analysis shows that the model is re-initialized after each external event (user-action) with identical update of both client applications (“synchronous work”). Enumerative analysis provides additional information. Indeed, the covering graph highlights the fact that particular sequences of transitions drive the model into markings corresponding to deadlock situations (no transitions to be fired in these markings). These deadlocks occur when a new message, encoding a new user-action, is sent to the supervisor application (shared resource) before a previous message is totally processed by this same supervisor application. A typical problem of concurrency is raised here. These theoretical results led us to propose a new PN model able to avoid deadlocks in the presence of event collisions. This second model is based on a general principle: when a user-action is performed in client application \( C_k \), a preliminary communication is established between \( C_k \) and the supervisor application. If the supervisor application is “available”, then the user-action becomes effective. Then, it consequently implies identical update of all the client applications \( C_i, i=1..n \), before reinitializing the global PN in its initial marking. The only hypothesis introduced in the problem is that the probability to see two messages encoding concurrent user-actions to reach the supervisor application exactly at the exact same time is null.

Structural and enumerative analyses have then been conducted on the new model. For simplicity, results obtained from enumerative analysis are only summarized here. Readers may refer to [9] for detailed calculations:

- no deadlock situation appears on the covering graph derived from model analysis,
- there always exists a sequence of transitions driving the model to its initial state with identical update of all client applications, even when user-actions concurrency occurs,
- all concurrent user-actions are detected and processed by the cancellation mechanism in the PN that models the supervisor application.

2.2 Development of a model-based architecture for CSSCW applications

The theoretical study of the extended model was used in the development of fault-tolerant CSSCW software applications. Here, the main difficulty was to preserve the model properties within the software collaborative application. Using object-oriented programming, we developed two classes respectively named *Place* and *Transition* that adopt the same semantics as in PNs and we used instances of these two classes to implement the PNs that will control supervisor and client applications. The obtained model-based distributed architecture is dynamic: the number of interconnected PNs can change as client applications connect or disconnect during the collaborative session. Its two main functions are: i) to manage the communications between client and supervisor applications and ii) to manage the evolution inside client and supervisor applications by controlling the sequential execution of their methods.

![Fig 1. PN model-based distributed architecture for the support of interactive CSSCW applications.](image)

3. Results and conclusion

The whole methodology was used in the design of two collaborative software applications in the field of telemedicine. The first one is dedicated to biomedical electrophysiological signal visualization and processing [10], the second one is aimed at conjointly visualizing and working on video documents (applications in video-monitoring of epilepsy and gastroenterology). Each application software is controlled by the underlying distributed model-based architecture (fig.1). For the evaluation and for the validation of the proposed approach, we first used the initial model. As predicted, when user actions are performed simultaneously, deadlocks are observed. When the initial model is replaced by the extended one, we observed that both
software collaborative applications are able to deal with simultaneous actions. Moreover, state consistency is maintained. From a theoretical viewpoint, PNs are well suited to the modeling of CSSCW architectures. Indeed, models consist in interconnected PNs that perfectly reflect the distributed nature of CSSCW architectures, allowing processes involved in the collaborative session to be represented and traced as external events (user actions) occur. From the user viewpoint, working with synchronous interactive reliable collaborative software is an outstanding advantage, especially in the field of medicine. In addition, the presented approach is transparent for users who can remotely work and interact in a more natural way, without any restrictive locking mechanisms. In a more general way, this approach is independent from the operating system, the communication network or the programming language. It could be easily applied to a large field of applications that meet the same issues: network interactive gaming, future mobile GPRS or UMTS embedded interactive applications, for which the specification, the design and the verification [11] of interactive user interfaces as well as the control of concurrent user-actions are of crucial importance.

References

A-Priori-Usability Testing of Ubiquitous-Computing Application Designs

Chris Stary
University of Linz, Department of Business Information Systems
Communications Engineering, Freistädterstraße 315
A-4040 Linz, Austria
christian.stary@jku.at

Abstract. This paper shows an analytical approach to test usability-engineering principles, such as task conformance. It can be applied at design time and operates on integrated design representations. This type of representation is required to keep the application context that is required for automated testing. The approach is exemplified through TADEUS which is a model-based approach that allows for seamless development and consequently, context-sensitive representations.

1 Objectives and Concept

Ubiquitous-computing applications are software applications that are not only environments augmented with computational features [6] but also applications that can be run using various interaction devices and styles (e.g., [1], p. 45). Considering the usability of these applications as a key factor of future information-technology developments methodologies, methods, and tools have to support the coupling of the logic of applications with different interaction devices, styles and modalities. For ubiquitous-computing applications not only a single (stationary) user interface has to be developed, but a diverse set of devices and related styles of interaction have to be made available for a (functional) software solution. A typical example for this setting is the access to public information systems via mobile devices (WAP (Wireless Application Protocol)-based cell phones, palmtops, PDAs (Personal Digital Assistants), tablet PCs, handhelds etc.) as well as stationary user-interface software, such as kiosk browsers at public places (airports, cine-plexes, malls, railway stations etc.). Hence, the same data (content) and major functions for navigation, search, and content manipulation become available through various presentation facilities and interaction features.

This kind of openness while preserving functional consistence does not only require the provision of different coaldalities of information and corresponding forms of presentation, such as text and audio streams, but also different ways of navigation and manipulation. For instance, WAP is designed to scale down well on simple devices, such as the mobile phone with less memory and display capacity. Due to the nature of
the devices and their context of use, it is not recommended to transfer Web application
directly to WAP applications without further analyses. For WAP applications it is
highly recommended to use menus as much as possible to save the user from using the
limited keypad facilities. In addition, any text should be short and consistent in its
structure etc. (cf. [2]).

Since it is the set of users that decide primarily via the user interface whether a
product is successful or not, the user-perceived quality (in terms of usability princi-
pies, such as adaptability) has to be ensured through development techniques and
tools. The latter have to comprise the measurement of usability principles at design
time to avoid time- and cost-consuming rework of coded user interfaces based on the
results of a posteriori evaluation.

Since ubiquitous-computing applications are polymorph - different users utilise dif-
ferent devices and interaction styles to access common information and functions op-
erating on and processing that information - the development procedure implies a con-
struction process that allows dynamic adaptability to various devices and interaction
styles to access and manipulate information. To this end, it is important that not only
the needs of the possible user population (context of use), but also the technological
capabilities to present and manipulate information are taken into account in the early
design phases of new products and services, and thus, become explicit part of design
representations. Such representations might be processed to indicate whether the tasks
and/or users of an application has been designed for can be supported or not. Hence,
analytic a-priori-usability testing aims for processing design representations to avoid
lengthy rework of already programmed ubiquitous-computing applications.

2 Analytic Usability Testing

Following the tradition of model-based design representations (cf. [4]) several models
are required to define (executable) specifications of interactive software:

- a task model - that part of the situational setting (organisation, public domain etc.) the interactive computer system is developed for
- the user model – the set of user roles to be supported, both, with respect to
functional roles (e.g. search for information), and with respect to individual
interaction preferences and skills
- a problem domain (data) model – the data and procedures required for task
accomplishment
- the interaction (domain) model – providing those modalities that might be
used for interactive task accomplishment.

Of particular interest for ubiquitous computing is the separation of the interaction
model from the problem domain (data) model, as an application should be separated
from the set of possible interaction devices and interaction styles to allow the dynamic
assignment of several devices and styles to application functions. The separation of
the task and user model from the data and interaction model is also of particular im-
portance, since it enables to implement the operational definition of task conformance
and adaptability, and thus, analytic evaluation.

The crucial part in order to achieve automated analytic evaluation of design repre-
sentations is to define operational metrics, i.e. to define algorithms which check
whether usability criteria are met. In the following two examples, one for task conformance, and one for adaptability, are given.

**Task conformance** is defined as follows: “A dialogue supports task conformance, if it supports the user in the effective and efficient completion of the task. The dialogue presents the user only those concepts which are related to the task” [3] (part 10). It can be measured in an analytical way through two metrics: completeness with respect to task accomplishment (*effectiveness*), and *efficiency*. The first metrics means that at the system level there have to be both, a path to be followed for user control (i.e. navigation procedures), and mechanisms for data presentation and manipulation, in order to accomplish a given task. The second metrics is required to meet the principle task conformance in an optimised way. High efficiency in task accomplishment can only be achieved, in case the users are not burdened with navigation tasks and are enabled to concentrate on the content to be handled in the course of task accomplishment.

In order to minimise the mental load the navigation paths have to be minimal. At the specification level an algorithm has to check whether the shortest path in the application model for a given task, device and style of interaction has been specified, in order to accomplish the control and data-manipulation tasks. For an implementation of these metrics on object-oriented design schemes see [5].

In ISO 9241, Part 10 'Dialogue systems are said to support suitability for individualization if the system is constructed to allow for adaptation to the user's individual needs and skills for a given task.' [3] Hence, in general, interactive applications have to be adaptable in a variety of ways: (i) to the organisation of tasks; (ii) to different user roles; (iii) to various interaction styles, and (iv) to assignments. Adaptability means to provide more than a single option, and to be able to switch between the various options for each of the issues (i) – (iv).

Adaptability with respect to the organisation of tasks means that a particular task might be accomplished in several ways. Hence, a specification enabling flexibility with respect to tasks contains more than a single path in a design representation within a task to deliver a certain output given a certain input. Flexibility with respect to user roles means that a user might have several roles and even switch between them, eventually leading to different perspectives on the same task and data. Adaptability with respect to interaction devices and styles, i.e. the major concern for ubiquitous computing support, does not only require the provision of several devices or styles based on a common set of interaction elements, but also the availability of more than a single way to accomplish interaction tasks at the user interface, for instance direct manipulation (drag & drop) and menu-based window management, within a particular style of interaction. The latter can be checked at the behaviour-specification level, namely through checking whether a particular operation, such as closing a window, can be performed along different state transitions. For instance, closing a window can be enabled through a menu entry or a button located in the window bar.

Adaptability of assignments involves (i) – (iii) as follows: In case several tasks are assigned to a particular user or a user changes roles (in order to accomplish a certain task) assignments of and between tasks and roles have to be changed. In case a user wants to switch between modalities or to change interaction styles (e.g., when leaving the office and switching to a computer-augmented environment), the assignment of interaction styles to data and/or tasks have to be modified. Changing assignments re
quires links between the different entities that are activated or de-activated at a certain point of time. It requires the existence of assignment links as well as their dynamic manipulation. Both can be provided, either through additional semantic relationships in design representations, such as ‘handles’ between roles and tasks (linking different models), or the runtime environment of the prototyping engine as in TADEUS [4].

In order to ensure adaptability (i) the designer has to set up the proper design space, and (ii) modifications have to occur in line with the semantics of the design space. The first objective can be met through providing relationships between design items, both, within a model (e.g., ‘before’ in TADEUS), and between models (e.g., ‘handles’ in TADEUS). The first enables flexibility with each of the perspectives mentioned before, e.g., organisation of tasks, whereas the second flexible mutual tuning of components, in particular the assignment of different styles of interaction to tasks, roles, and problem domain data.

The second objective can be met allowing to manipulate relationships according to the syntax and semantics of the representation scheme (i.e. specification language), and providing algorithms to check the correct use of the language as well as the changes related to the manipulation of relationships. For instance, the ‘before’ relationship in TADEUS can only be set between sub tasks in the task model. A dedicated algorithm for ‘before’ has to process the design representation to ensure consistent and correct use of the relationship. In case the relationship is modified, e.g., ‘before’ set between other sub tasks, the corresponding restrictions have to be lifted, and new ones, according to the semantics of the relationship are enforced, however, using the same type of algorithm as before (for ‘before’).

References