Engineering Universal Access: Unified User Interfaces

UAHCI 2001 Tutorial

Constantine Stephanidis
Anthony Savidis
Demosthenes Akoumianakis

Foundation for Research and Technology – Hellas
Institute of Computer Science
Human-Computer Interaction and Assistive Technology Laboratory
Science and Technology Park of Crete
Heraklion, Crete
GR-71110, GREECE
Tel: +30-81-391741, Fax: +30-81-391740
{cs, as, demosthe}@ics.forth.gr

7 August 2001
# Table of contents

**Agenda** ii  
**Instructors’ Biographies** iii  
**Course objectives** v  
**Abstract** vi  
**Slides**  

## Introduction to Unified User Interfaces  
- Universal access  
- Coping with diversity  
- Technical approaches  
- Automatic user interface adaptation  
- The concept of Unified User Interfaces  

## Designing to cope with diversity  
- Part I: Background  
  - The design issue  
  - Understanding context  
- Part II: The unified design method  
  - Concepts  
  - Phases  
  - Techniques  
- Part III: Tools  

## Unified Interface Engineering  
- Unified Interface Architecture  
- Adaptation Scenarios and control flow  
- Unified Interface Architecture Component Analysis  

## Tools for developing Unified User interfaces  
- Introduction  
- Metaphor Development  
- Toolkit Integration  
- Toolkit Augmentation  
- Toolkit Expansion  
- Toolkit Abstraction  

## Universal Access and the Web  
- Browsers as interface tools  
- Platform diversity on the Web  
- Automatic Web-page adaptation  

## Challenges and Future Work  
- Software development process  
- Identifying diversity  
- Designing for diversity  
- Computing platforms and embedded OS  
- Concluding remarks  

---

UAHCI 2001 | Page i | Stephanidis, Savidis, Akoumianakis
## Agenda

<table>
<thead>
<tr>
<th>When</th>
<th>What</th>
</tr>
</thead>
<tbody>
<tr>
<td>08.30 – 10.00</td>
<td>Introduction: Universal Access, Copying With Diversity, Technical Approaches</td>
</tr>
<tr>
<td>10.00 – 10:30</td>
<td>Morning Break</td>
</tr>
<tr>
<td>10:30 – 12:00</td>
<td>Unified user interface design</td>
</tr>
<tr>
<td>12:00 – 13:30</td>
<td>Lunch</td>
</tr>
<tr>
<td>13:30 – 14.00</td>
<td>Tools, examples, case studies</td>
</tr>
<tr>
<td>14:00 – 15:00</td>
<td>The unified user interface engineering</td>
</tr>
<tr>
<td>15:00 – 15:30</td>
<td>Afternoon Break</td>
</tr>
<tr>
<td>15:30 – 16:30</td>
<td>The unified user interface engineering (Cont.)</td>
</tr>
<tr>
<td>16:30 – 17:00</td>
<td>Discussion and Conclusions</td>
</tr>
</tbody>
</table>
Instructors’ Biographies

Constantine Stephanidis
Constantine Stephanidis is Deputy Director and head of the Human-Computer Interaction and Assistive Technology Laboratory at the Institute of Computer Science, Foundation for Research and Technology – Hellas. He is also a member of the Faculty at the Department of Computer Science and member of the Senate of the University of Crete. For many years, he has been engaged, as Prime Investigator, in pioneering research work partly funded by the European Commission and in 1995 he introduced the concept of “User Interfaces for All” as a socio-technical goal in the context of the emerging Information Society. He has published about 200 technical papers in scientific archival journals and proceedings of international conferences related to his fields of expertise. He serves in the Editorial Board of several scientific journals and the Programme Committee of many International Conferences, and has organized international scientific conferences, workshops, seminars, and panels. Prof. Stephanidis is the Editor-in-Chief of the Springer international journal “Universal Access in the Information Society” and the Editor of the LEA book “User Interfaces for All – Concepts, Methods and Tools”. He is the Founding Chair of the International Conference “Universal Access in Human-Computer Interaction”, Founder of the ERCIM Working Group “User Interfaces for All” and General Chair of its annual Workshop, Founder and Chair of the International Scientific Forum “Towards an Information Society for All”, and the project manager of the European Commission funded Thematic Network “Information Society for All” (IS4ALL). Prof. Stephanidis is member of the Executive Committee and of the Board of Editors of the European Research Consortium for Informatics and Mathematics (ERCIM), member of the Advisory Committee of the World Wide Web Consortium (W3C), head of the W3C Office in Greece, and participant in the activities the W3C - Web Accessibility Initiative.

Anthony Savidis
Anthony Savidis has a B.Sc. in Computer Science, an M.Sc. in Information Systems and Software Engineering, and a Ph.D. in User Interface Development Tools. He is a member of the Human-Computer Interaction and Assistive Technology Laboratory at the Institute of Computer Science, Foundation for Research and Technology – Hellas since 1989, and has been involved in various European collaborative Research and Development projects including: HELIOS-HANDYNET, RACE IPSNI R1066, TIDE GUIB TP103, TIDE GUIB-II TP215, TIDE ACCESS TP1001, and ACTS AVANTI AC042. He is the chief designer / developer of the HOMER UIMS for building Dual User Interfaces, the COMONKIT interface toolkit for Rooms-based non-visual interfaces, the PIM tool for open toolkit integration, and the I-GET UIMS for implementing Unified User Interfaces, the SCANLIB switch-based augmented Windows library, and the HAWK non-visual toolkit supporting open metaphor realisation. His research interest focus on interface implementation languages, development processes and architectures, and interface toolkits for diverse users and computing platforms.

Demosthenes Akoumianakis
Demosthenes Akoumianakis received a B.Sc. (Hons) in Computing in Business, and he was the recipient of the 1st IBM Prize Award for his Final Year Undergraduate Dissertation. He also received an M.Sc. in HCI on design environments for user adaptable interfaces, and a Ph.D. in HCI on ‘Knowledgeable’ Tools for Evolving HCI Design - Theory and Practice. Since 1993, he is a member of the Human-Computer Interaction and Assistive Technology Laboratory at the Institute of Computer Science, Foundation for Research and Technology - Hellas. Demosthenes Akoumianakis has been involved in several European collaborative Research and Development projects, including TIDE-CORE TP126 and TIDE-CORE TP213, the TIDE-HEART study TP309, TIDE-ACCESS
TP1001, ACTS-AVANTI AC042, and the DE4105 -WAI (Web Accessibility Initiative). Demosthenes Akoumianakis has co-authored many publications and articles in international archival scientific journals and referred conference proceedings in the fields of Human-Computer Interaction and Assistive Technology, and has co-presented tutorials at HCI International ’97 and HCI International ’99. He is member of ACM, member of the Programme Committee and reviewer for the International Conference on Computer-Aided Design of User Interfaces (CADUI), the International Conference on Universal Access in Human-Computer Interaction (UAHCI), the International Workshop on Tools for Working with Guidelines (TFWWG2000) and the World Multi-conference on Systemics, Cybernetics and Informatics (SCI’2000/ISAS2000). Demosthenes Akoumianakis has served as the secretary of the International Scientific Forum “Towards an Information Society for All”. His current research interests focus on methodologies, design support environments and tools for universal access and adaptable and adaptive user interfaces.
Objectives of the course

Attendees of this tutorial will be introduced to:

- Universal access as the conscious and systematic effort to proactively apply principles, methods and tools for universal design, in order to develop user interfaces that are accessible and usable by anybody, anywhere, at anytime.
- The dimensions of diversity, which are intrinsic to the emerging Information Society, and create the requirement for universal access.
- The need for principled and systematic approaches towards accommodating diversity in the user- and usage- context of interactive products, applications and services.
- The concept of Unified User Interfaces and the Unified User Interface development process as an effective engineering approach integrating and applying the principles and practice of universal design for the development of universally accessible and usable user interfaces.
- Adaptation in the context of Unified User Interfaces.
- The need to employ appropriate tools and a corpus of related tool requirements.
- An analysis of the appropriateness of the World Wide Web as a platform for facilitating universal access.
Abstract

Universal access refers to the conscious and systematic effort to proactively apply principles, methods and tools for Universal Design, in order to develop user interfaces that are accessible and usable by anybody, anywhere, at anytime. The requirement for universal access stems from the growing impact of the fusion of the emerging technologies, and from the different dimensions of diversity, which are intrinsic to the Information Society. These dimensions become evident when considering the broad range of user characteristics, the changing nature of human activities, the variety of contexts of use, the increasing availability and diversification of information and knowledge sources and services, the proliferation of technological platforms, etc.

Unified User Interfaces have been conceived as a means of accommodating the interaction requirements of the broadest possible end-user population. The Unified User Interface development process conveys a new perspective on the development of user interfaces by providing a principled and systematic approach towards accommodating diversity in the user- and usage- context of interactive products, applications and services. Unified User Interfaces provide a concrete insight into how the principles of universal design can shape prevailing HCI design and development practices so that the range and scope of interactive experiences offered to the end-users are tailored to their individual requirements and expectations.

This tutorial introduces the concept of Unified User Interfaces and the Unified User Interface development process, elaborating some of its distinctive properties that render it an effective approach for the development of universally acceptable and usable user interfaces. The most important of these properties is the capability of self-adapting interactive behaviour. The tutorial unfolds the challenges of the Unified User Interface development process, comparing it with currently adopted practices to implementing interactive software, and explaining how this new process integrates and applies the principles and practice of universal design in the user interface development life-cycle.

The development of Unified User Interfaces is viewed as a multidisciplinary process, and is elaborated through examples and demonstrations. Particular emphasis is given to the need for combining a range of complementary perspectives, including design methods and techniques, component technologies, interface toolkits and programming. The Unified User Interface design method is presented as a design technique capable of capturing into a single design representation the different design patterns reflecting user- and usage-context- diversity. The Unified User Interface implementation method is presented through incrementally revealing a new distributed architectural framework, along with an engineering strategy for structuring the implementation.

The need to employ appropriate tools along the way is identified, and a corpus of tool requirements is established. Tool requirements reflect the needs for metaphor development, and toolkit integration, augmentation, abstraction and expansion. The tutorial also includes a brief discussion of the appropriateness of the World Wide Web as a platform for facilitating accessible and high-quality interaction. Finally, the tutorial identifies some challenging design and implementation issues, requiring further research efforts, and draws some conclusions regarding the contribution of the Unified User Interface Development framework towards Universal Access and Usability in the field of Human-Computer Interaction.
Engineering Universal Access: Unified User Interfaces

Constantine Stephanidis,
Anthony Savidis,
Demosthenes Akoumianakis

Human-Computer Interaction and
Assistive Technology Laboratory
@ ICS-FORTH

Tutorial agenda

- Introduction to Unified User Interfaces
- Unified User Interface Development
- Universal Access and the Web
- Challenges and Future Work
Introduction to Unified User Interfaces - agenda

- Universal access
- Coping with diversity
- Technical approaches
- Automatic user interface adaptation
- The concept of Unified User Interfaces

Information Society or Digital Age

- Interactive software applications and services for
  - Anyone - variety in user profiles
  - Anywhere and Anytime - variety in contexts of use
  - Any purpose - variety in tasks
Universal Access in the Information Society (1/2)

The right of **all** citizens to obtain and maintain **access** to a society-wide pool of **information** resources and interpersonal **communication** facilities, given the varieties of contexts of use.

Universal Access in the Information Society (2/2)

**Design for All**
- PC
- Kiosks
- Mobile phones
- TV

**User Interface Level**
- Communication protocols
- Web
- Satellite links
- Bandwidth

**Telecommunications Infrastructure**
- Work
- Education
- Entertainment
- Healthcare
- Social

**Application Domain & Services Level**

**Accommodating Diversity**
• **Accessibility**
  For each task, there is a sequence of accessible input actions and associated feedback leading to successful accomplishment

• **High-quality**
  For any individual user in a particular context of use, there is at least one path that optimally supports the accomplishment of the given task

---

**Accessibility vs Interaction Quality (1/5)**

• **Accessibility**
  – Dictates support for alternative I/O

• **Quality**
  – Dictates support for alternative designs
Accessibility vs Interaction Quality (2/5)

The same user may require different access and interaction quality attributes for performing a single task depending on the context of use.

Accessibility vs Interaction Quality (3/5)

- While driving
  - Driver is “situationally” motor- and visually-impaired
    - minimum attention, simple dialogues, speech, etc.
- In a noisy environment
  - User is “situationally” deaf
Accessibility vs Interaction Quality (4/5)

• Low interaction quality may reduce accessibility
  – What can I do?  User tasks
  – How can I do it?  Action sequences
  – What is this?  Artifact interpretation
  – Where am I?  Context clarity

Accessibility vs Interaction Quality (5/5)

– Even with a “physically” accessible interface, a particular user may be unable to carry out an interaction task
– What is “good design” for one user, may be a “bad design” for another
– Pursuing a single optimal design for everyone is a utopia
Introduction to Unified User Interfaces - agenda

- Universal access
- **Coping with diversity**
- Technical approaches
- Automatic user interface adaptation
- The concept of Unified User Interfaces

Universal Access = Coping with Diversity

- User profiles
  - Age, cultural / educational background, mental / sensory / motor skills, specific purpose of use, etc
- Contexts of use
  - Environment (e.g., noise, terminal position, lighting)
  - Technological platform (e.g., presence or absence of particular I/O devices, network bandwidth, etc)
Diversity in users

Diversity in contexts of use (1/2)
Diversity in contexts of use (2/2)

- Car
- Airplane
- Ship
- Hospital
- Factory floor
- Office
- School

Introduction to Unified User Interfaces - agenda

- Universal access
- Coping with diversity
- Technical approaches
  - Automatic user interface adaptation
  - The concept of Unified User Interfaces
Technical Approaches for Universal Access

• Reactive
  – Applying modifications and introducing add-ons over existing technology, to overcome technology-driven accessibility and interaction quality problems

• Proactive
  – Systematically catering for accessibility and interaction quality from the early phases of design and throughout the development life-cycle

Reactive methods (1/3)

Configuration of I/O
  – Binding of input sequences
    • shortcuts, accelerators
  – Device fine-tuning
    • mouse keyboard sensitivity, sticky keys
  – Display control
    • styles, colours, layout
Reactive methods (2/3)

Accessibility add-ons

– Accessibility technologies (Java / Active Accessibility)
  • SDKs to retrieve display structure, and externally manipulate interaction controls
– Alternative access systems
  • screen reader, virtual keyboard / mouse

Reactive methods (3/3)

Application of accessibility guidelines to modify existing inaccessible systems
Proactive methods (1/2)

- A new engineering paradigm
- Systems accessible by design
  - Application of accessibility guidelines
  - Appropriate development tools
    - There is a need for new commercially available toolkits supporting accessible interaction elements
  - Software I/O control in new computing platforms

Proactive methods (2/2)

- What about Java Pluggable Look&Feel?
  A generalisation API over windowing controls, enabling visual style to be altered
  - Still rectangular geometry, visual attributes, mouse & keyboard navigation, layout-based instance hierarchy
  - X Windows / Xt, X Attribute Defaults, except run-time style switching capability and audio feedback support
Reactive vs Proactive methods (1/3)

- Reactive
  - Lower interaction quality – modifications instead of alternative designs
- Proactive
  - Higher interaction quality – alternative designs adapted to user and context attributes

Reactive vs Proactive methods (2/3)

- Reactive
  - Many dialogues cannot be reproduced (appropriately modified), hence some applications or application parts can not be made accessible
- Proactive
  - All dialogue scenarios can be implemented
  - Applications are explicitly designed and developed for accessibility
Reactive vs Proactive methods (3/3)

- Reactive
  - Relatively cheap, and may quickly provide some sort of accessible interaction

- Proactive
  - Relatively expensive initial overhead

Introduction to Unified User Interfaces - agenda

- Universal access
- Coping with diversity
- Technical approaches
- **Automatic user interface adaptation**
- The concept of Unified User Interfaces
Automatic User Interface Adaptation (1/9)

Feedback on operation completion (here, bookmark addition)

Links presented as buttons

Link enumeration and structure overview pane

Automatic User Interface Adaptation (2/9)

Interaction for motor-impaired: automatically scanned window manipulation toolbar

Interaction for motor-impaired: all GUI objects accessible through automatic scanning

Interaction for motor-impaired: automatically scanned HTML elements (including image-maps)

Welcome to
Automatic User Interface Adaptation (3/9)

Interaction for motor-impaired: on-screen keyboard for text input

Interaction for motor-impaired: keyboard layouts that speed up interaction (e.g. by following letter-frequency criteria)

Automatic User Interface Adaptation (4/9)

Adapting to the context of use: kiosk mode operation

ICS-FORTH

Stephanidis, Savidis & Akoumianakis

Slide 31

ICS-FORTH

Stephanidis, Savidis & Akoumianakis

Slide 32

UAHCI 2001
Automatic User Interface Adaptation (5/9)

The interface’s response to the detection of the fact that the user seems incapable to complete the task of selecting a link from the “Link Bar.”

Automatic User Interface Adaptation (6/9)

A simple dialog from which the user selects and loads previously visited documents.
Automatic User Interface Adaptation (7/9)

... gets converted to the same dialogue with integrated guidance, if the user seems to be unable to comprehend its use.

Automatic User Interface Adaptation (8/9)

- **User awareness**
  - User-oriented information / knowledge

- **Usage-context awareness**
  - Context-oriented information / knowledge
Automatic User Interface Adaptation (9/9)

• Sources of knowledge
  – Knowledge which is made available or can be inferred prior to initiation of an interaction session (off-line)
  – Knowledge which can be only inferred by analysing interaction monitoring information (on-line)

Two Types of Interface Adaptation

• Adaptability
• Adaptivity

⇒ Differentiate according to the type of knowledge employed in performing adaptation
Adaptability - Definition

*Interface adaptation applied on the basis of *off-line* knowledge*

Applied before initiation of interaction to deliver an accessible and high-quality user interface.

Adaptability - Properties

- User- and context- attributes are considered known off-line
- An appropriate design is selected for the end-user, and the given usage-context
- Adaptability takes place before interaction is initiated
- Adaptability realises an accessible interface
Adaptivity - Definition

*Interface adaptation applied on the basis of *on-line* knowledge*

- Applied during interaction, aiming to enhance the initially delivered user interface

Adaptivity - Properties

- User- and context- attributes are dynamically inferred on-line
- The design already chosen for the end-user and the given usage-context is enhanced
- Adaptivity takes place after interaction is initiated
- Adaptivity requires an accessible interface
Adaptivity and Adaptability - Complementary Roles

Adaptability - provide initial interface

Automatically adapted interface

interface instance 1

interface instance N

user1 context 1

userN context N

Adaptivity - continuously enhance

interface instance 1

user1 context 1

Introduction to Unified User Interfaces - agenda

- Universal access
- Coping with diversity
- Technical approaches
- Automatic user interface adaptation
- The concept of Unified User Interfaces
Unified User Interfaces (1/3)

End-User view
– A user interface tailored to individual user attributes and to the particular context of use

Unified User Interfaces (2/3)

Design view
– A user interface design populated with polymorphic artifacts, i.e., encompassing alternative dialogue artifacts
  • each alternative artifact addresses specific user- and usage-context-parameter values
Unified User Interfaces (3/3)

Engineering view

- A repository of implemented dialogue artifacts, out of which the most appropriate are selected at run-time
  - decision making is needed to select the appropriate interaction artifacts given the end-user- and usage-context-attribute values

Unified User Interface - Definition

- A user interface self-adapting to user- and usage-context, encompassing
  - Alternative implemented dialogue artifacts
  - User- and context- information
  - Decision making capability selecting user- and context- appropriate dialogue artifacts
  - Interface control to apply decisions made
Levels of Adaptation in Unified User Interfaces

<table>
<thead>
<tr>
<th>Levels</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>semantic</td>
<td>internal functionality, information represented</td>
</tr>
<tr>
<td>syntactic</td>
<td>dialogue sequencing, syntactic rules, user tasks</td>
</tr>
<tr>
<td>constructional</td>
<td>devices, object attributes, interaction techniques</td>
</tr>
<tr>
<td>physical</td>
<td></td>
</tr>
</tbody>
</table>

Polymorphism in Unified User Interfaces (1/6)

- Automatic adaptation at any level implies the ability to *polymorphose*
  - i.e., for a given task, design alternative interactive artifacts according to different user- and usage-context- attribute values
Polymorphism in Unified User Interfaces (2/6)

- **Lexical polymorphism**
  - construction of the physical design domain
  - design properties space
    - I/O devices
    - interaction techniques
    - interaction objects and their attributes

Polymorphism in Unified User Interfaces (3/6)

*Example of lexical polymorphism*
Polymorphism in Unified User Interfaces (4/6)

- **Syntactic Polymorphism** (task-structure differentiation)
  - sequences of user actions
  - initiation / interim / completion feedback
  - availability of operations and interaction progress preconditions
  - multiple views and direct manipulation

Polymorphism in Unified User Interfaces (5/6)

**Example of syntactic polymorphism**

```
define_file = define file before delete command
define_file = provide name or select directly
```

```
delete_file
  before
  define_file
  delete_command
```

```
provide_name
  parallel
  provide_directly
```

Polymorphism in Unified User Interfaces (6/6)

Example of syntactic polymorphism (cont.)

```
delete file = define file before delete command
define file = select target
delete command = activate
```

```
delete a file
  before
  define_file  delete_command

select_target activate
```

Summarising on Unified User Interfaces (1/2)

- Automatic User Interface Self-Adaptation
- User- and usage-context- attribute driven
- Implies polymorphism potentially at lexical, syntactic and semantic levels
Summarising on Unified User Interfaces (2/2)

• There is a need for a new development process
  – Design process
  – Implementation architecture
  – Engineering process

• There is a need for new development tools
  – Capabilities / functionality
  – Assessment, availability, and suggestions

Designing to cope with diversity
Concepts and Principles
Plan

• **Part I: Background**
  – The design issue
  – Understanding context

• **Part II: The unified design method**
  – Concepts
  – Phases
  – Techniques

• **Part III: Tools**
The notion of context

- Context refers to the parameters which shape and influence the execution of a task
  - user condition, requirements and abilities
  - technological platform (terminal, bandwidth, etc)
  - usage context (desktop, mobile, ubiquitous, etc)

The research question

- Understanding context
  - Ethnographically-inspired inquiries
  - Scenarios

- Modelling context
  - User modelling techniques
  - Context modelling

- Accounting for context variety
  - Adaptable and adaptive interaction
**HCI and the study of context**

- Traditionally, a hard issue for HCI design
  - Assumptions about:
    - average «typical» users,
    - the device is typically a desktop PC
    - the context of use was the business environment
  - Methodological focus on:
    - productivity enhancement
    - how tasks should be carried out rather than how they are being carried out
  - Keystrokes as unit for studying interaction

**Reality is different ...**

- Diversity regarding
  - the users operating computational devices
  - the type of devices
  - the context of use
  - the type, nature and scope of tasks
  - etc
Coping with diversity

• To allow for deviation such a model of HCI introduced the notion of adaptation to cope with unforeseen events
• Originally, it was the user that was required to adapt

Coping with diversity (Cont.)

• Progressively, the device obtained the computational power to exhibit adaptations
The prevalent methods devised

- Adaptable interfaces
  - customising
  - tailoring
- User modelling
  - explicit models of user behaviour
- Adaptive interfaces
  - run-time modifications of dialogue
- Intelligent interfaces
  - systems that exhibit human-like behaviour

Adaptable interfaces: Taloring aspects of interaction
User modelling

• Explicit representation of user’s personal information such as domain knowledge, beliefs, preferences, interests, etc.

• Several approaches
  – Overlay models
  – Embedded user models
  – User modelling shells
  – User modelling servers
Intelligent user interfaces

• Systems exhibiting human-like behaviour
  – Model-based approaches
  – Agents
  – Adaptive interaction

The problem today

• The Information Society has invalidated many of the original assumptions which shaped progress in the field
  – the user is no longer a tractable element to be studied in a laboratory
  – the device is no longer the traditional PC with a keyboard, mouse and a VDU
  – the context of use is no longer bound to the business environment
Furthermore

• Interaction remains complex and multi-faceted phenomenon
• Its social dimension adds to the complexity

Consequently ...

• Keystrokes can no longer provide an informative unit for the study of interaction
• The field is in search of
  – more adequate theoretical frames of reference, thus the shift towards developmental sciences
  – richer development frameworks, thus recent proposal for tangible interfaces, embodied interfaces, disappearing computers
  – novel evaluation perspectives, thus the claims for cooperative evaluation, participatory design
The Need

- Capture global execution context of a task
- Task execution context is dictated by
  - user abilities and preferences
  - technological platform
  - context-of-use
- Lack of suitable methods

Part II: The unified design method

- Concepts
- Phases
- Techniques
What is a method?

- HCI literature reports on a plethora of methods
  - micro-level
  - macro-level
- According to (Olson and Moran, 1996) a complete method comprises
  - a statement of the problem it aims to address
  - a device (technique, tool or representation)
  - a procedure for using the device
  - a clear set of outcomes

Unified design method

- Problem statement
  - Develop a representation of the global execution context of a task
- Device
  - Polymorphic task decomposition
- Procedure
  - Enumerate - Abstract - Rationalise cycles
- Outcomes
  - Polymorphic task hierarchy
  - Styles and accompanying design rationale
Task Execution Context

- An execution context refers to how a task is to be accomplished by a user U, using an interaction device P in a specified context of use C
- Traditional design techniques assume
  - “Average” or typical user
  - Desktop platform
  - Business-oriented usage

Relaxing the assumptions …

- The “typical” user assumption
  - anybody
- The desktop platform assumption
  - anywhere
- The business-oriented use assumption
  - anytime
Implications

- A single design no longer suffices
- A task could have multiple interactive manifestations
- Design space becomes complex
  - Enumeration (of design alternatives)
  - Representation
  - Rationalization

Consequently …

- We need new design methods
  - Cope with diversity
  - Guide designers through a structured process
  - Orthogonal to existing design practices
- Unified design is a solution
Unified design is a complete micro-method

- **Problem**
  To capture and represent in a unified design-structure all the alternative dialogue artefacts

- **Device**
  Polymorphic task hierarchies

- **Process**
  Abstract task definition with incremental polymorphic physical specialisation

- **Outcome**
  Polymorphic task model
  Design artefacts
  Recorded rationale for alternative design patterns

---

Polymorphic task decomposition

- A technique which has the following properties:
  - Hierarchical structure
  - Root represents reusable design patterns
  - Non-root nodes are contextually bound instances of a design abstraction
  - Leaf nodes represent concrete interaction components

- Each alternative decomposition is called a decomposition style, shortly a style
Styles

- User interface as a composition of styles
- Styles can be analysed through any other appropriate design method
  - Heuristics,
  - GOMS analysis,
  - Traditional HTA,
  - UAN,
  - Formal specifications

An example
In other words

- Styles correspond to execution contexts
- Execution contexts are defined by the triad <User profile, Platform, Context>
- A style should be designed so as to facilitate specific task execution context(s)
- A particular style may be good enough for an execution context but totally inappropriate for another

Style: Interactive File icons
Style: Command line

C:\ del f1.txt

Style: Interactive Directory Tree
**Style relationships**

- `C:\ del f1.txt`
- `f1.txt`
- `a2.txt`
- `a1.txt`
- `f2.txt`

**Compatibility**

**Styles definition**

- New styles may be generated
  - By unfolding established patterns and implementing corresponding artefacts (c.f. styles for deleting a file)
  - By re-engineering arteacts
Artefact re-engineering

- Assumes the availability of an artefact
- Requires competence in using abstraction

For each style ...

- Develop suitable argumentation for each style
  - Why does it exist?
  - What issue does it support?
  - When should it be initiated?
  - Where is it implemented?
  - How does it compare against competing styles
Enumerate - Abstract - Rationalise

- An argumentative process:
  - Reflection on action
    - Identify existing tasks and how they are performed
    - Assess breakdowns in existing task structures
  - Creative stage
    - Set filters (or questions) which describe the breakdown
      - User questions (e.g. how is the product used by a blind user?)
      - Platform questions (e.g. how is the task executed on a portable device?)
      - Context questions (e.g. how can the task be executed away from the desktop?)
    - Develop proposals for envisioned task structures
Enumerate - Abstract - Rationalise

Techniques used

- Observation which reveals patterns and artefacts in use
- Task analysis in cases where a “system” is already available
- Envisioning and rapid prototyping to assess with users likely options
- Other formative techniques which reveal artefacts and patterns of use (e.g., scenarios)

Enumerate - Abstract - Rationalise

An example scenario

XXX has just completed an order for several pharmaceuticals items. The on-line pharmacy store requests XXX to enter the credit card number to proceed with clearance of the order and delivery of goods.
Enumerate - Abstract - Rationalise

The design task

• Design the dialogue through which a user can enter information about his/her credit card
• Information to be entered includes:
  – Card number
  – Expire data
  – Type of card
  – etc

Enumerate - Abstract - Rationalise

Identifying the issues

• Issues raised:
  – How does the user insert his/her credit card number?
  – How does the user specify the expire date?
  – How does the user indicate the type of card?
Enumerate - Abstract - Rationalise

**Describing the options**

- **O-1-1**: Type in Digit-by-Digit the number
- **O-1-2**: Speak out digit-by-digit the number
- **O-1-3**: Select digit-by-digit from a selection set

**Issues**

- **I-1**: How does the user insert the credit card number
- **I-2**: How does the user specify the expire date
- **I-3**: How does the user indicate the type of card

**Enumerate - Abstract - Rationalise**

**Exploring O-1-1**

- **O-1-1**: Type in Digit-by-Digit the number

**Issues**

- **I-1**: How does the user insert the credit card number
- **I-2**: How does the user specify the expire date
- **I-3**: How does the user indicate the type of card

Card No: ^__________
Enumerate - Abstract - Rationalise

Envisioning the artefact

- Form-based dialogue for providing credit card information
- Typical in Web documents

Credit Card No: ^Credit Card No: ^
Expires: ^__/__

- VISA
- Access
- MasterCard
- Other ^__________

Submit

Enumerate - Abstract - Rationalise

Re-engineering an artefact

- Re-engineering may be facilitated using different techniques such as
  - screening using re-engineering ‘filters’
  - abstraction
Enumerate - Abstract - Rationalise

Setting filters

- How can the task be performed by a user with gross-temporal control familiar with switch-based interaction?
- How can the task be carried out with an alternative pointing device (e.g. a stylus of a palmtop computer)?
- How can the task be performed in a public kiosk?

Enumerate - Abstract - Rationalise

Alternative option

I-1: How does the user insert the credit card number
O-1-1: Type in digit-by-digit the number
O-1-2: Speak out digit-by-digit the number
O-1-3: Select digit-by-digit from a selection set

I-2: How does the user specify the expire date

I-3: How does the user indicate the type of card

O-1-1: Type in digit-by-digit the number
O-1-2: Speak out digit-by-digit the number
O-1-3: Select digit-by-digit from a selection set

Issues

Card No: ^
Enumerate - Abstract - Rationalise

Another option

- Switch-based interaction
- Use of dedicated artefacts (e.g. visual keyboard)

A non-visual option

Issues

I-1: How does the user insert the credit card number

I-2: How does the user specify the expire date

I-3: How does the user indicate the type of card

O-1-1: Type in digit-by-digit the number

O-1-2: Speak out digit-by-digit the number

O-1-3: Select digit-by-digit from a selection set
Enumerate - Abstract - Rationalise

Using abstractions

Credit Card No: ____________
Expires: __/__

☐ VISA   ☑ MasterCard   ☐ Access
☐ Other  ____________

Submit

Enumerate - Abstract - Rationalise

Identifying roles

label associated to domain object (abstract syntactic)

Number or name (abstract semantic)

logical grouping (abstract syntactic)

check-boxes (exclusive choice - abstract semantic)

group separator (generalised syntactic)

custom text

command (abstract syntactic)

Submit
Enumerate - Abstract - Rationalise

Producing Abstract Model

- Grouping
  - Label & Valuator
  - Exclusive Choices & Labels
  - Command

- Title

4 groups, 4 digits/group

- Month, Year

Enumerate - Abstract - Rationalise

Deriving Alternatives (1/2)

- Window
  - Text Entry
    - "Card No"
  - ComboBox
    - VISA
    - MasterCard
    - Access

- Push Button
  - Submit

- Radio Groups
  - "Expires"
Enumerate - Abstract - Rationalise

Deriving Alternatives (2/2)

- Identify common design abstractions amongst design alternatives
- Model the abstractions
  - Abstract interaction elements
  - Design templates

ICS-FORTH
Stephanidis, Savidis & Akoumianakis
Slide 113
Enumerate - Abstract - Rationalise

• Develop suitable argumentation for each design options
  – Why does it exist?
  – What issue does it support?
  – When should it be initiated?
  – Where is it implemented?
  – How does it compare against competing alternatives?

• Decide on suitable representation

Enumerate - Abstract - Rationalise

*Informal definition*

• For a design variable, rationalisation involves assessing alternatives against a designated set of criteria, to select an optimal assignment, given the current state of the knowledge available.
Enumerate - Abstract - Rationalise

**Formal definition**

Let the following hold true:
- X is a design variable
- S={S₁, S₂, …, Sₙ} the set of alternative assignments
- C={C₁, C₂, …, Cₙ} the set of criteria

then

Rationalisation entails devising a technique which allows the decision maker to select for X a maximally preferred option from S, given Cᵢ ∈ C.

---

**Enumerate - Abstract - Rationalise**

**An example**

**Task to be studied**
- Develop the polymorphic task hierarchy for command order in direct manipulation user interfaces

**Approach**
1. Enumerate alternatives by observing users
2. Abstract to identify common dialogue structures & criteria of differentiation
3. Rationalise on the grounds of the identified criteria or empirical justification

**Outcomes**
- 1. Design alternatives
- 2. Polymorphic task hierarchy
- 3. Design criteria
- 4. Justification or rationale
Enumerate - Abstract - Rationalise  

*Steps to be followed*

- Steps of the designer
  - Associate the task with the abstract design pattern “Decide_command_order”
  - Declare design variable
  - Define alternatives
  - Identify criteria for polymorphosing the pattern
    - heuristics, preferences, experiment, GOMS analysis, etc
  - Collect evidence
  - Consolidate evidence to reasonable representation

---

Enumerate - Abstract - Rationalise  

*Problem description*

- Design variable
  - commandOrder
- Set of alternatives
  - {of_Syntax, fo_syntax}
- Criteria
  - task completion time, Tc
  - efficiency of task performance, E
  - frequency of errors, F
  - planning time, Pt
  - action time, At
Enumerate - Abstract - Rationalise

Enumerating alternatives

• Function-Object syntax
  – First click on function
  – Second click on object

Enumerate - Abstract - Rationalise

Enumerating alternatives (Cont.)

• Object-Function syntax
  – First click on object
  – Second click on function
Enumerate - Abstract - Rationalise

Defining plausible criteria

- Dependent variables constitute the range of possible criteria for decomposition
  - task completion time, $T_c$
  - efficiency of task performance, $E$
  - frequency of errors, $F$
  - planning time, $P_t$
  - action time, $A_t$
- Choice of criteria based on experimental results

Compiled evidence

- Hypothetical conclusions
  - Command ordering is indifferent across a range of dependent variables
    * task completion time
    * efficiency of task performance
    * frequency of errors
  - There is a preference ranking for command for
    * action time
    * planning time
Enumerate - Abstract - Rationalise

Criteria for polymorphosis

Polymorphosis can only be justified for two criteria:
- Planning time, \( P_t \)
- Action time, \( A_t \)

Representation scheme

Representational primitives
- \( A: A \in \{ A_1, \ldots, A_n \} / A_j = \text{Dependent variable} \)
- \( TC: TC \in \{ T_{C_1}, \ldots, T_{C_n} \} / T_{C_j} = \text{Application-specific task context} \)
- \( C: C \in \{ C_1, \ldots, C_n \} / C_j = \text{Independent variable} \)
- \( O: O \in \{ O_1, \ldots, O_n \} / O_j = \text{Factor level of dependent variable} \)

The design space (DS) is defined as

\[
DS = \{ x: x \in A_i(O_i) \cup \cdots \cup A_n(O_n) \}
\]

Preference and indifference expressions
- \( p(TC, A, O, O) \)
- \( i(TC, C, A, O, O) \)
Enumerate - Abstract - Rationalise

Extract from representation

/*------ Task context aggregation policy -------*/
policy(graphic_Editing, action_Time(true), command_Order)
policy(graphic_Editing, planning_Time(true), command_Order)
policy(graphic_Editing, error_Frequency(true), command_Order)
...
h(graphic_Editing, action_Time(true), command_Order, FO_syntax, OF_syntax)
h(graphic_Editing, planning_Time(true), command_Order, FO_syntax, OF_syntax)
/*------ End of task context aggregation policy -------*/

/*------ Solicited Preferences -------*/
p(graphic_Editing, command_Order, action_Time(true), FO_syntax, OF_syntax)
p(graphic_Editing, command_Order, planning_Time(true), OF_syntax, FO_syntax)
i(graphic_Editing, command_Order, task_completion_Time(true), OF_syntax, FO_syntax)
i(graphic_Editing, command_Order, error_Frequency(true), OF_syntax, FO_syntax)
...
/*------ End of Solicited Preferences -------*/

Part III: Tools

Case studies & examples
Design support

• Analysing users
  – identifying requirements
  – identifying tasks and preferred patterns
• Analysing the context of use
  – identifying the execution context of tasks
• Analysing the interaction platform
  – interaction objects and attributes
  – input/output devices
  – available/desirable interaction techniques

USE-IT: A design tool for user-adapted interactions

• Developed in the ACCESS project
• Part of the ACCESS development platform
• Intended for the designer not the end user
• A knowledge-based system
• Focus on lexical aspects of interaction
• Running under Windows
Supported high-level design tasks

- Elicitation of user parameters
  - definition of parameters and valid value range
  - populating the user taxonomy
- Description of platform resources
  - input / output devices
  - interaction object classes
- Description of task context
  - logical view of intended interaction styles

Output

- Rule base
- Lexical adaptions
- Critiquing of completeness and consistency
- Task context schemas
- Conditional rules, defaults, preferences
- Conflict detection and resolution
- etc.
USE-IT session

• Representation of the target platform
  – designate object classes and attributes
  – input / output devices

• User model
  – declare parameters and value range
  – define user abilities, preferences

• Task context schema
  – use styles of unified design to decide on task contexts
  – populate task contexts

User modelling approach

• Developing user scenarios
  – A user is to carry out a text-editing task in a graphical user interface. The user has mild motor impairments, which delimit control to gross temporal movements, exercised through contact with the fist. Fingertips cannot be reliably employed due to tremor on key-press, while movements can be performed in timed patterns and upon demand.
The user modelling tool

A new USE-IT project
Defining user parameters

A populated taxonomy
Developing a device representation

Describing interaction objects

Lexical Level Adaptation Constituents

1. Load the lexical specification
2. Compile the internal representation of the specification
3. Construct the lexical tree of the specification

Show file
Print
Exit
An example of a visual platform

Example of non-visual platform
Representation

The user model
The rules

- The rules compiled are subsequently used to:
  - select plausible devices for the user
  - select plausible interaction techniques (e.g. scanning)
  - set parameters of interaction techniques (e.g. manual versus autop scanning)
  - etc

Accessibility filters

- Employing accessibility filters
  - How is the product used by a user who possesses alternative reliable control acts (e.g., movement of one hand, movement of both hands, directed eye-gaze, head movement, movement of lower limbs, vocalisations)?
  - How is the product used by a user who possesses alternative contact site (e.g., finger tips of one hand, finger tips of both hands, fist, hand-held pointer, headstick, mouthstick, left upper part of head, right upper part of head, from top of head)?
Task context descriptions

- Identifying relevant guidelines
  - “… for a GUI to be accessible by motor impaired users, there should be a method for carrying out mouse, or other pointing device functions, with a keyboard or a keyboard emulator”
- Envisioning artefacts

Iterative prototyping

Design wisdom and Rationale.
Artefacts

• A prototype of the virtual keyboard

Artefacts (Cont.)

• Prototypes for window management
Adding justification

Assigning styles to task contexts

• Style prototypes
Declaring alternatives

Unified design method
- hierarchical task decomposition
- enumeration
- abstraction
- rationalisation

Abstract task context
(encapsulated alternatives)

Task context instances
to be populated

Populating the task context

ICS-FORTH  Stephanidis, Savidis & Akoumianakis  Slide 154
Populating the task context

Conditional rules
Checking the context schema

Running the adaptation engine
Silent critic

Generating recommendations
Formatted recommendations

Revising the original design

Unified design method
- hierarchical task decomposition
- enumeration
- abstraction
- rationalisation

Deleting through scanning options in a selection set
Design updates

Applying the recommendations

Unified interface specification

USE-IT

adapt(Object, Attribute, TaskContext, Value)

adapt(Object, Attribute, TaskContext, Value)
Concluding remarks

• USE-IT is a system that implements some of the concepts of unified design
  – Abstract interaction objects
  – Style designation through task contexts
  – Hierarchical organisation of task contexts
  – Object binding to task contexts
    • Depending on the task context the same object may exhibit alternative interactive manifestation

Concluding remarks (Cont.)

• Additional functions of USE-IT
  – Generates consistent and complete recommendations in a toolkit-specific format
  – Non-trivial attributes, e.g.,
    • feedback (initiation, interim, completion)
    • presentation policy (access, topology)
  – Semantic binding of an adaptation decision to <task context, object class, criterion, attribute>
  – Incremental design updates
    • new styles can be introduced by declaring new task contexts
  – Contextual definition of scope of adaptations e.g.,
    • object classes, devices, user categories to be addressed by the adaptation engine
Concluding remarks (Cont.)

- In the current version, USE-IT does not support
  - style relationships
    - compatibility, substitution, augmentation, etc
  - adaptations at syntax or semantic levels

Summary

- **Use when** execution context varies
  - alternatives should be enumerated for differentiating user- and context- parameters for the same tasks
- **Based on** polymorphic task hierarchies,
  - styles designated to execution contexts of a task
- **Driven by** diversity
  - user- and context- attributes, being the primary design parameters
General remarks

- Unified design introduces concepts which offer an insight to designing for diversity
  - polymorphic task hierarchies
  - styles
  - focus on artefact and supporting rationale
- Suitable for various design fashions
  - bottom up: identifying concrete artefacts and then abstracting
  - top-down: from abstractions to concrete artefacts
  - middle out: combining the above
- Application in the AVANTI browser

Unified Interface Development

*(agenda)*

- Prologue
- Unified interface design
- **Unified interface engineering**
- *Tools* for unified interfaces
Unified Interface Engineering – Outline (1/3)

• A run-time architecture for implementing interface adaptation supporting:
  – Evolution
  – Reuse
  – Distribution

Unified Interface Engineering – Outline (2/3)

• Specific scenarios describing distributed control flow and inter-component communication to accomplish adaptation
Unified Interface Engineering - Outline (3/3)

• Some techniques to bring the software implementation more close to the interface design, thus making easier to program design changes

Unified Interface Engineering (agenda)

• Unified Interface Architecture
• Adaptation Scenarios and control flow
Revealing an **Architectural Pattern**

**Vehicle**
- Divide and conquer
- Role separation
- Orthogonality
- Eliminate replication

**Starting point**
Provide dialogue according to user- and context-attributes values

---

Revealing an **Architectural Pattern**

(1/9)

User-, Context-Attributes

Provide adapted dialogue
Revealing an *Architectural Pattern* (2/9)

User-, Context-Attributes  \rightarrow Decide which dialogues to activate  \rightarrow Activate decided dialogues

Revealing an *Architectural Pattern* (3/9)

User attributes  \rightarrow Decide which dialogues to activate  \rightarrow Activate decided dialogues  \rightarrow Context attributes
Revealing an Architectural Pattern (4/9)

User attributes

Context attributes

Activate decided dialogues

Decide which dialogues to activate

Dialogues

Revealing an Architectural Pattern (5/9)

User attributes

Context attributes

Activate / cancel decided dialogues

Decide which dialogues to activate / cancel

Dialogues
Revealing an Architectural Pattern
(6/9)

Stephanidis, Savidis, Akoumianakis

Decide which dialogues to activate / cancel

Activate / cancel decided dialogues

User Information Server

Context attributes

Dialogues

Revealing an Architectural Pattern
(7/9)

Stephanidis, Savidis, Akoumianakis

Decide which dialogues to activate / cancel

Activate / cancel decided dialogues

User Information Server

Context Information Server

Dialogues
Revealing an Architectural Pattern (8/9)

Context Information Server

User Information Server

Decision Making Component

Activate / cancel decided dialogues

Dialogues

Revealing an Architectural Pattern (9/9)

Context Information Server

User Information Server

Dialogue Patterns Component

Decision Making Component

ICS-FORTH
Stephanidis, Savidis & Akoumianakis
Slide 183
Slide 184
Unified Interface Architecture

User Information Server

Context Information Server

Decision Making Component

Dialogue Patterns Component

Component Analysis

• Role and behaviour
• Content
• Communication
• Implementation
Context Information Server
- *Role and Behaviour*

• To supply *context attribute values* (machine and environment)
  – Static (non-changing during interaction, e.g. peripheral equipment)
  – Dynamic (may change during interaction, e.g. environment noise)

Context Information Server
- *Content (1/2)*

• Awareness of I/O devices and their properties
  – e.g. hand-held binary switches, speech synthesiser (English, Greek), high resolution display (mode 16bits, 1024x768, 75Hz), Pentium-III 500MHz / 2MB cache / 128 MB memory, 20 GB hard disc
Context Information Server  
- **Content (2/2)**

- Environment information (requires appropriate sensors)
  - e.g. acoustic noise, light reflection on display, presence of the user in front of the terminal, humidity, smoke detection

Context Information Server  
- **Communication (1/2)**

- Context information is supplied in the form of \((\text{attribute, value})\) pairs
  - Simple model
  - Highly generic
  - Value can be aggregate
  - e.g.  
    - (“environment noise”, “78db”)
    - (“resolution”, “1024x768”)
    - (“user presence”, “no”)
Context Information Server
- Communication (2/2)

• Send by request
  – all / some attributes with their values

• Send by modification
  – post attributes when their value changes during user interaction

Context Information Server
- Implementation

• Registry for I/O equipment
• Use of sensors for retrieving environment parameters
• Location awareness
User Information Server -
Role and Behaviour

• To supply user attribute values
  – Known off-line, before initiating interaction
  – Detected on-line, from real-time interaction-monitoring analysis
    • e.g. fatigue, loss of orientation, inability to perform the task, interaction preferences

User Information Server -
Content (1/2)

• Repository of user profiles
• Logic for interaction-monitoring analysis
User Information Server -
Content (2/2)

User profile model

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Pₙ</td>
<td></td>
</tr>
</tbody>
</table>

User profile instance

<table>
<thead>
<tr>
<th>computer knowledge</th>
<th>expert</th>
<th>frequent</th>
<th>average</th>
<th>casual</th>
<th>native</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web knowledge</td>
<td>very good</td>
<td>good</td>
<td>average</td>
<td>some</td>
<td>limited</td>
</tr>
<tr>
<td>ability to use left hand</td>
<td>perfect</td>
<td>good</td>
<td>some</td>
<td>limited</td>
<td>none</td>
</tr>
</tbody>
</table>

User Information Server -
Communication

• Send **by request**
  – all / some attributes with their values

• Send **by modification**
  – post attributes when their value changes during interaction
  – post dynamically detected attributes and their values
User Information Server - *Implementation (1/3)*

- Could employ a database to store / retrieve user profiles
- Dynamic attribute detection requires further processing

User Information Server - *Implementation (2/3)*
User Information Server -
*Implementation (3/3)*

User profile

Pattern Matching

Interaction history

Inference Component

User Models

Behavioural Action Patterns

**Decision Making Component -
Role and Behaviour**

- Matches user- and context-attribute values to the most appropriate dialogue artifacts
- Decides *why*, *when* and *how* to adapt
Decision Making Component - Content

- **Awareness** of design artifacts (e.g. named, indexed), user- / context- attributes and respective values (e.g. “age”, integer, 5...110)
- **Decision making** knowledge

Decision Making Component - Communication

- Receives user- and context- attributes (from UIS)
- Posts decisions for *activation* or *cancellation* of implemented dialogue patterns (to DPC)
Decision Making Component - Implementation (1/2)

- Knowledge-based component, playing the role of an adaptation expert
- Rule-based implementation framework may suffice

Decision Making Component - Implementation (2/2)

- If given particular user, usage context, dialogue state, and interaction history, a human designer can decide optimal adaptation
  - then we can make the machine able to adapt as well by embedding designer's decision logic
Dialogue Patterns Component -
*Role and Behaviour*

- **Applies** adaptation decisions, *making available* to the user the necessary dialogue patterns
- **Knows** where implemented dialogue patterns *reside*

Dialogue Patterns Component -
*Content*

- **Repository** of implemented dialogue patterns
  - Local / remote *address*
  - Source / binary *form*
Dialogue Patterns Component - Communication

- Receives *activation / cancellation* decisions for dialogue components
- Receives interaction *monitoring control* commands
- Posts interaction *monitoring data*

Dialogue Patterns Component - Implementation (1/2)

- *Communication* with other components
- *Coordination* of dialogue artifacts
- *Manipulation* of dialogue artifacts
- *Monitoring* of interaction within dialogue artifacts
Dialogue Patterns Component -
Implementation (2/2)

Unified Software Architecture -
Some Key Remarks (1/2)

• Comprehensive start-up cost to set-up an interactive system as a unified implementation...
• But afterwards, the incorporation of adaptation behaviour becomes a standardised convenient process
Unified Software Architecture - Some Key Remarks (2/2)

• …more dialogue artifacts
• …more interaction monitoring
• …more user attributes
• …more interaction-analysis logic
• …more context attributes
• …more adaptation-oriented logic

Unified Interface Engineering (agenda)

• Unified Interface Architecture
• Adaptation Scenarios and control flow
Adaptation Scenarios and Control Flow – Example (1/5)

- Detecting dynamically **confusion** in performing a task and providing **task-based guidance**

Adaptation Scenarios and Control Flow - Example (2/5)

1. UIS requests monitoring for a particular task
2. DPC accepts request, and activates necessary monitoring components
3. DPC continuously posts interaction monitoring data back to UIS
Adaptation Scenarios and Control
Flow - Example (3/5)

Stephanidis, Savidis, Akoumianakis
ICS-FORTH
Slide 215

UIS receives continuously data and builds an annotated interaction history.

UIS continuously analyses interaction history to detect particular action patterns.

UIS detects a confusion pattern for a particular task, and sends this assumption to DMC.

Adaptation Scenarios and Control
Flow - Example (4/5)

Stephanidis, Savidis, Akoumianakis
ICS-FORTH
Slide 216

DMC receives an assumption for confusion on a particular task.

DMC decides to activate a dialogue component providing task-based guidance to resolve confusion.

DMC posts the necessary activation message to DPC.
Adaptation Scenarios and Control
Flow - Example (5/5)

DPC receives the activation message and locates the corresponding dialogue component.

Finally, DPC activates this specific dialogue component.

Unified Interface Development (agenda)

- Prologue
- Unified interface design
- Unified interface engineering
- **Tools for unified interfaces**
Tools for Unified Interfaces

*agenda*

- **Introduction**
  - Metaphor Development
  - Toolkit Integration
  - Toolkit Augmentation
  - Toolkit Expansion
  - Toolkit Abstraction

Tools for Unified Interfaces - Introduction (1/7)

- What is the “entrance barrier” for interface tools in order to facilitate the development of universally accessible interactions? Or more specifically...
Tools for Unified Interfaces - Introduction (2/7)

• Considering the diversity in end-users and usage-contexts, which are the most important implementation ingredients for building unified interfaces?

Tools for Unified Interfaces - Introduction (3/7)

• Different users, in different contexts and situations of use, likely require different interaction metaphors
  • Metaphor development
Since designers may employ any particular interaction toolkit, the interface tool employed for implementation should enable programmers to utilise this toolkit

- **Toolkit integration**

- Support the introduction of additional interaction techniques within interaction objects, for the cases in which the built-in techniques are not sufficient

- **Toolkit augmentation**
Tools for Unified Interfaces - Introduction (6/7)

• Support the introduction of new interaction objects within toolkits, for the cases in which the originally supplied set is not sufficient
  • *Toolkit expansion*

Tools for Unified Interfaces - Introduction (7/7)

• Facilitate the manipulation of interaction objects completely relieved from physical interaction properties
  • *Toolkit abstraction*
Tools for Unified Interfaces
(agenda)

- Introduction
- **Metaphor Development**
  - Toolkit Integration
  - Toolkit Augmentation
  - Toolkit Expansion
  - Toolkit Abstraction

Metaphor Development - Why? (1/5)

- Interaction metaphors are user-oriented
  - Design should reflect the attributes of target users
  - It is unlikely that a single metaphor can be globally optimal for all end-users
Metaphor Development - Phases in Unified Paradigm (2/5)

- **Design**
- **Realisation**
- **Implementation**

Metaphor Development - Phases (3/5)

- **Design**
  - concepts, features, entities, properties, behaviours, relationships

- **Realisation**
  - media, modalities, interaction objects, interaction techniques, attributes, dialogue, design

- **Implementation**
  - coding, implementation libraries, programming model, run-time architecture

MFC, InterViews, Motif, Mac Toolbox, JFC, UIML

testing
Metaphor Development -
Advantages of the Approach (4/5)

• Multiple realisations of a single metaphor design
  – Modifications on a metaphor realisation can be applied without affecting original metaphor design

Metaphor Development -
Advantages (5/5)

• Multiple implementations of a single metaphor realisation
  – Modifications on a metaphor implementation are allowed without affecting original metaphor realisation
Design of an Interaction Metaphor
- Where to Start From

• Top-level *container* interaction objects play the most important role

Top-level Containers -
Key Role (1/5)

Windowing / Desk-top Metaphor?
Top-level Containers - 
*Key Role (2/5)*

Books Metaphor?

---

Top-level Containers - 
*Key Role (3/5)*

Teacher / Whiteboard Metaphor?
Top-level Containers -
Key Role (4/5)

• Embedded Objects do not
Affect Overall Interaction Metaphor

- Push buttons - Electric devices
- Sliders / potentiometers - Electric devices
- Check-boxes - From filling
- Menus - Restaurant
- Gauges - Electric devices

Top-level Containers -
Key Role (5/5)

Non-Visual Rooms - COMONKIT Toolkit

- Door to another room
- Leads to rooms which are one level above or below
- Interaction object
- Group: “floor”, “ceiling” and front, left, back, right “walls”
Top Level Containers – A Step Forward

- HAWK Non-Visual Toolkit providing a generic container class, supporting programmable:
  - Navigation dialogue for contained objects
  - Display / presentation policy
  - I/O device binding

HAWK Toolkit – Typical Embedded Objects

- Embedded objects are non-visual realizations of broadly used metaphoric objects:
  - Menu
  - Listbox
  - Radio button
  - Single- / multi-line editor
  - etc
HAWK Toolkit – Used in Demanding Projects

- To implement non-visual custom-made hypermedia tool in the ACCESS project
- To implement the non-visual component of the unified AVANTI browser

Tools for Unified Interfaces (agenda)

- Introduction
- Metaphor Development
- **Toolkit Integration**
  - Toolkit Augmentation
  - Toolkit Expansion
  - Toolkit Abstraction
Toolkit Integration

• As toolkits we consider software libraries providing the implementation of interaction elements
  – e.g. Windows, OSF/Motif, JFC, HAWK,...

Toolkit Integration - Definition

• The ability of interface development tools to import toolkits, thus making imported interaction elements available in the dialogue implementation process.
**Toolkit Integration - Role in Unified Development**

- Utilise elements from multiple sources (i.e. *multi-toolkit platform*)
- Supplying a *common API*, as opposed to native toolkit programming models

**Toolkit Integration - Cross-Metaphor Interoperability**

- Diagram showing various objects and containers, including:
  - Left Wall Container
  - Right Wall Container
  - BlackBoard Container
  - Window Container
  - Library Container
  - FileManager Object
  - Video Object
  - Calendar Object
  - Document Object
Tools for Unified Interfaces  
*agenda*

- Introduction
- Metaphor Development
- Toolkit Integration

- **Toolkit Augmentation**
  - Toolkit Expansion
  - Toolkit Abstraction

**Toolkit Augmentation - Definition**

- The process through which *additional* interaction techniques are *injected* into the original (*native*) interaction elements supplied by a particular toolkit
Toolkit Augmentation - 
Role in Unified Development

• Enhancing the accessibility and quality of interaction elements by augmenting with extra interaction techniques (both display, as well as input, may be affected)

Toolkit Augmentation - 
An Example (1/6)

• Augmented Windows MFC
• Switch-based access (binary switches)
• Lexical dialogue decomposition into two fundamental actions: **select**, **next**
Toolkit Augmentation - 
An Example (2/6)

- Categories of object classes subject to augmentation:
  - Top-level windows
  - Container objects
  - Text-entry objects
  - Composite objects
  - Button categories

Toolkit Augmentation - 
An Example (3/6)

- Top-level windows
  - All top-level windows have been augmented with an additional toolbar, supporting scanning interaction, providing all window management operations
Toolkit Augmentation - An Example (4/6)

- State control
- Position control
- Size manipulation

Toolkit Augmentation - An Example (5/6)

- Text-entry objects
  - Requiring text to be supplied, imposing the need for *keyboard emulation*
Toolkit Augmentation - An Example (6/6)

Tools for Unified Interfaces (agenda)

- Introduction
- Metaphor Development
- Toolkit Integration
- Toolkit Augmentation

**Toolkit Expansion**
- Toolkit Abstraction
**Toolkit Expansion - Definition**

- The construction of new interaction objects, not originally supported by toolkits

**Toolkit Expansion - Role in Unified Development**

- To implement the various artifacts in adapted interactions, developers may need to build new interaction objects
  - In this process, the development tool should provide all the adequate support
Tools for Unified Interfaces (agenda)

- Introduction
- Metaphor Development
- Toolkit Integration
- Toolkit Augmentation
- Toolkit Expansion
- Toolkit Abstraction

Toolkit Abstraction - Definition

- The provision of interaction objects entirely de-coupled from physical interaction properties
Toolkit Abstraction -
Abstract Selector Example

- Open
- Save
- Save as...
- Quit

- column "restaurant"
- auditory / 3D pointing
- auditory
- feedback

- 3D pointing
- sound feedback
- synthetic speech
- "Save as..."
- "Quit"

User choice

No of options

Toolkit Abstraction -
Role in Unified Development (1/2)

- The provision of abstract objects at the implementation layers, enables the construction of unified artifacts:
  - i.e. dialogues composed of abstract objects which can be instantiated, through programming control, to alternative physical forms
**Toolkit Abstraction - Role in Unified Development (2/2)**

- The abstraction requirement is technically considered to be the most important for unified interface development.

---

**Tutorial agenda**

- Introduction to Unified Interfaces
- Unified Interface Development
- *Universal Access and the Web*
- Challenges and Future Work
Universal Access and the Web - agenda

- Browsers as interface tools
  - Platform diversity on the Web
  - Automatic Web-page adaptation

Browsers as Interface Tools (1/10)

- Interactive components
  - what is interactively presented to the user via the Web-page

- Non-interactive components
  - functionality “behind the scenes”, not dealing with interaction or display
Browsers as Interface Tools

(2/10)

• Web-page specific, i.e. *client side*
  – HTML, CSS, XML, scripts (JavaScript, VisualBasicScript), embedded components (ActiveX, JavaBeans)
  • Varying implementation forms (script, content, programmed interactive components, style definition and use)

Browsers as Interface Tools

(3/10)

• *Server-side* functionality
  – CGI (Common Gateway Interface), ServeLets (Server-side applets), ASP (Active Server Pages)
  • Usually perform some type of data filtering, processing and retrieval, and then dynamically construct a target Web page
Browsers as Interface Tools
(4/10)

- *Client-side* code is actually the *User Interface*
- *Server-side* code is mainly the non-interactive *functional core*

Browsers as Interface Tools
(5/10)

- Browsers seem to preserve the *principle of separation*
  - A principle which was a “hot issue” for interface tools in the early 80s,...
  - subject to dispute and argumentation in the early 90s, …
  - and silently integrated within most of commercial interface tools in the late 90s
Browsers as Interface Tools
(6/10)

• Diverse development techniques
  – Declarative hypertext (HTML)
  – Scripting procedural (scripts)
  – Declarative formatting (styles)
  – Interface structural definition (forms, UIML)
  – Std programming (embedded components)
  – Semantic definitions (XML)

Browsers as Interface Tools
(7/10)

• Most of the alternative techniques are not standardised...
• There are variations per technique
  – JavaScript / VisualBasicScript
  – ActiveX / JavaBeans
  – DOM notational access
  – CSS use syntax
Browsers as Interface Tools (8/10)

• What Web mostly offered to UI developers?
  – Interactive document metaphor
  – Instant global delivery

Browsers as Interface Tools (9/10)

• Is development easier compared to traditional desk-top applications?
  – For simple things yes, for serious applications, no
  – Non-linear growth of development complexity, in relation to application complexity
  – Non-linear growth of entrance barrier in relation to application complexity
Browsers as Interface Tools
(10/10)

development complexity, entrance barrier

desk-top application complexity

web application complexity

Universal Access and the Web - agenda

- Browsers as interface tools
- *Platform diversity on the Web*
- Automatic Web-page adaptation
Platform Diversity on the Web
(1/5)

• Spread over a wide range of operating systems, with various browsers
  – PCs, MACs, Work-stations

• Porting to embedded operating systems with alternative protocols
  – WAP phones, Web-enabled devices

Platform Diversity on the Web
(2/5)

• Are all browsers on the various platforms accessible? with high quality interaction?
  – …blind user access?
  – …motor-impaired users?
  – …what about the elderly?
  – …the children?
Platform Diversity on the Web  
(3/5)

• Browsers for blind users  
  – pwWebSpeak, V-Lynx, AVANTI, NAUTILOS
• Browsers for motor-impaired users  
  – AVANTI, NAUTILOS

Platform Diversity on the Web  
(4/5)

• …or alternatively  
  – Use an alternative access system, over a typical browser  
    • Screen-reader for blind  
    • Virtual keyboards for motor-impaired.
  – W3C / WAI guidelines for Web authoring, to enable alternative access systems make a better job
Platform Diversity on the Web (5/5)

• There is no alternative browser, nor an alternative access systems for platforms such as:
  – phones
  – home appliances
    • TV, refrigerator, washing machine,...
  – office equipment
    • fax, copier, coffee machine,...

Universal Access and the Web - agenda

• Browsers as interface tools
• Platform diversity on the Web
• *Automatic Web-page adaptation*
Automatic Web-page Adaptation

• Client-side
  – Adaptation logic, constituents and control embedded within the Web-page

• Server-side
  – Adaptation logic and control residing on server side, producing adapted Web-pages

Automatic Web-page Adaptation - Client-side (1/2)

• Limited adaptation at the level of:
  – Document structure
  – Document content
  – Dialogue components
Automatic Web-page Adaptation - 
Client-side (2/2)

• Implementation mechanisms
  – Style differentiation (CSS, XML / XSL)
  – Content / dialogue differentiation (via scripting for dynamic content selection)
  – For more dynamic dialogue, embedded components must be implemented
  – User profile management requires persistent shared data and state maintenance

Automatic Web-page Adaptation - 
Server-side (1/2)

• Flexible adaptation at the level of:
  – Document *structure*
  – Document *content*
  – *Dialogue* components
Automatic Web-page Adaptation -
Server-side (2/2)

- Implementation mechanisms
  - User profile database
  - Page templates
  - Decision making
  - Dynamic page construction

Automatic Web-page Adaptation -
Wrap-Up

- Server-side adaptation is functionally superior to client-side adaptation
- While development responsibility is on document authors, re-usable services may help in making automatically adapted sites
Tutorial agenda

- Introduction to Unified Interfaces
- Unified Interface Development
- Universal Access and the Web
- Challenges and Future Work

Challenges and Future Work

- Software development process
  - Identifying diversity
  - Designing for diversity
  - Computing platforms and embedded OS
  - Concluding remarks
**Software Development Process (1/7)**

- Unified interface development is a new interface development strategy aiming to cope with diversity on users and usage-contexts
  - *There are specific technological steps which will move us closer to unified interfaces*

**Software Development Process (2/7)**

- Employment of component-ware technologies through which prefabricated dialogues are delivered
  - e.g. ActiveX, JavaBeans, OpenDOC
Software Development Process

(3/7)

• Bridges among the various component technologies enabling interoperability
  – Ability to combine dialogue components complying to different component-ware layers

Software Development Process

(4/7)

• Development of dialogue component repositories / directories supporting indexing and querying on the basis of design parameters
  – sub-task(-s)
  – user- / context- attributes
  – other
Software Development Process (5/7)

- Standardisation of user-oriented information, and production of *universal user-profile* databases
  - Legal issues are involved, so that permissions and access restrictions can be managed or regulated by users themselves

Software Development Process (6/7)

- Representation and deployment of design logic in computable forms, to enable *run-time design assembly*
  - Production of design knowledge-bases, supporting querying by criteria matching, and exploration, to enable re-usability
Software Development Process
(7/7)

- Standardisation of adaptation-oriented s/w interface reference architectures
  - Proposals for specific inter-component communication protocols and functional behaviour
    - i.e. *such as the unified architecture communication protocol*

Challenges and Future Work

- Software development process
- *Identifying diversity*
- Designing for diversity
- Computing platforms and embedded OS
- Concluding remarks
Identifying Diversity (1/2)

• How do we reveal those human-personality related parameters which are likely to affect the way interaction should be delivered?

Identifying Diversity (2/2)

• Is it possible practically, theoretically or legally to make such information available to a s/w system?
Identifying Diversity - Example

- User anxious, in a hurry, tired, does not understand the interface feedback
  - Body language analysis?
  - Heart-beat rate monitoring?
  - Facial expression analysis?
- *In this politically correct?*

Challenges and Future Work

- Software development process
- Identifying diversity
- **Designing for diversity**
  - Computing platforms and embedded OS
  - Concluding remarks
Diversity-Based Optimal Design
(1/2)

• Given individual attributes are known, how do we design in an optimal manner for those?

Diversity-Based Optimal Design
(2/2)

• How do we decide that differentiation of design artifacts is dictated when some individual attribute values differ?
Challenges and Future Work

- Software development process
- Identifying diversity
- Designing for diversity

- Computing platforms and embedded OS
- Concluding remarks

Computing Platforms and Embedded OS (1/4)

- The installation of embedded OS in various computing platforms, e.g. phones, home appliances, home electronics, public terminals, moves us away from the h/w manufactured applications and services (including the User Interface)
Computing Platforms and Embedded OS (2/4)

• Software firms are enabled to deliver competitive s/w for new computing platforms, while users have choices from a collection of alternative interactive s/w applications

Computing Platforms and Embedded OS (3/4)

• The separation between the h/w producer and the service developer opens new opportunities for interactive s/w, over a large variety of computing platforms
Computing Platforms and Embedded OS (4/4)

- An example:
  - We buy a mobile phone, and then we purchase the s/w we need:
    - a phone-book from W
    - an agenda from X
    - a Web-browser from W, and
    - a remote file-manager from Z

Challenges and Future Work

- Software development process
- Identifying diversity
- Diversity-based optimal design
- Computing platforms and embedded OS
- **Concluding remarks**
Concluding Remarks
(1/3)

• The Information Society is characterised by a considerable diversity in users and usage contexts
• Accessible and high-quality of interaction is crucial to ensure that anyone, anywhere and at anytime is enabled to use interactive s/w applications and services

Concluding Remarks
(2/3)

• Design differentiation is necessary to address diversity
  – no single design can satisfy the needs of all users in all usage-contexts
• Unified User Interface Development aims to address those challenges through a systematic interface design and engineering process, pursuing automatic user interface adaptation
Concluding Remarks (3/3)

• During the design phase, the Unified User Interface development mainly affects the artifact organisation process, rather than the artifact generation approach.
• During the engineering phase, it mainly affects the software architecture, and dialogue component organisation, rather than the dialogue implementation approach.
• These factors contribute to low deployment cost.