Universally Accessible UIs: The Unified User Interface Development

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

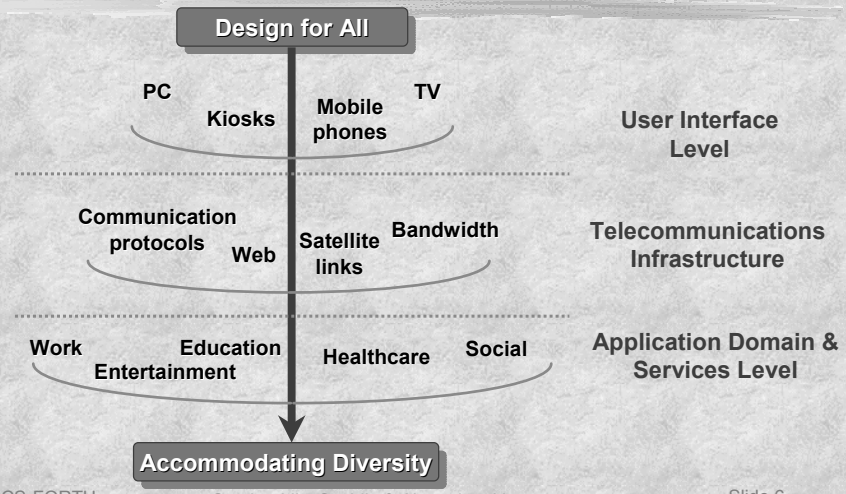
Information Society or *Digital Age*

- **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)*



HCI *for* Universal Access

- ***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*

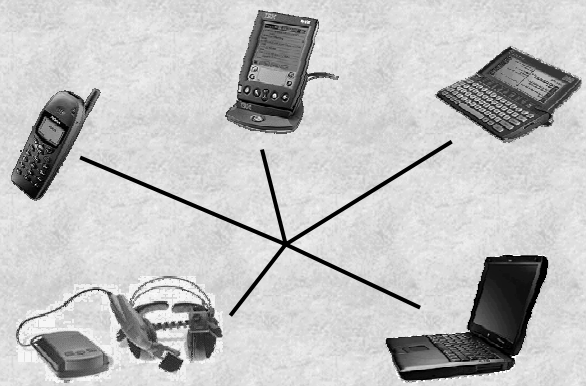
- Universal access
- ***Coping with diversity***
- Technical approaches
- Automatic user interface adaptation
- The concept of Unified User Interfaces

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

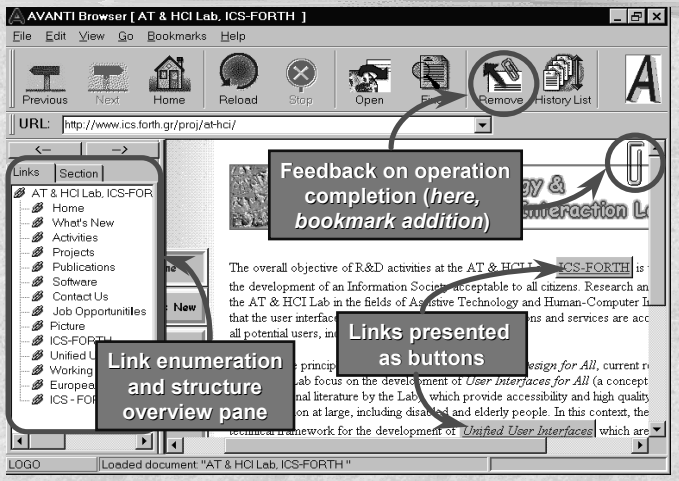
Reactive *vs* Proactive methods *(3/3)*

- Reactive
  - Relatively cheap, and may quickly provide some sort of accessible interaction
- Proactive
  - Relatively expensive initial overhead

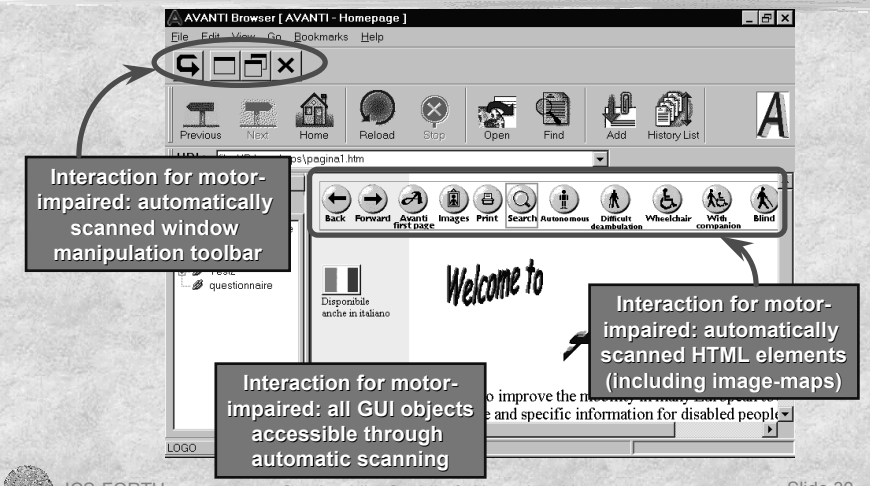
Introduction to Unified User Interfaces - *agenda*

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

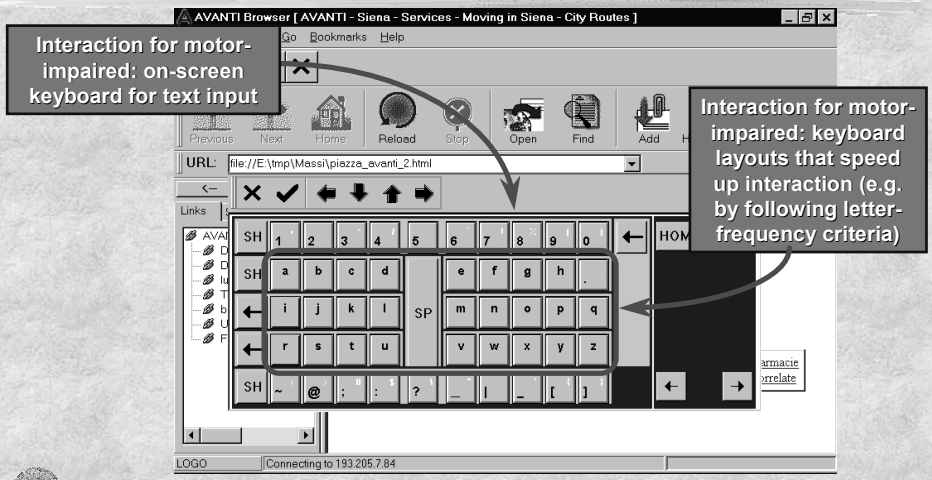
Automatic User Interface Adaptation *(1/9)*



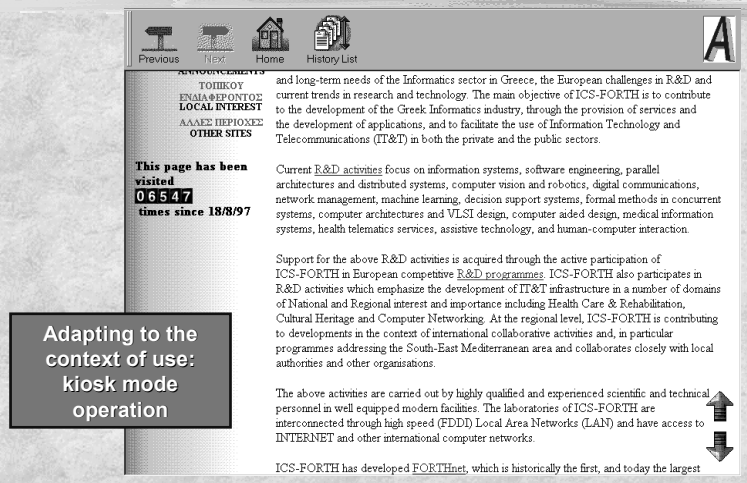
Automatic User Interface Adaptation *(2/9)*



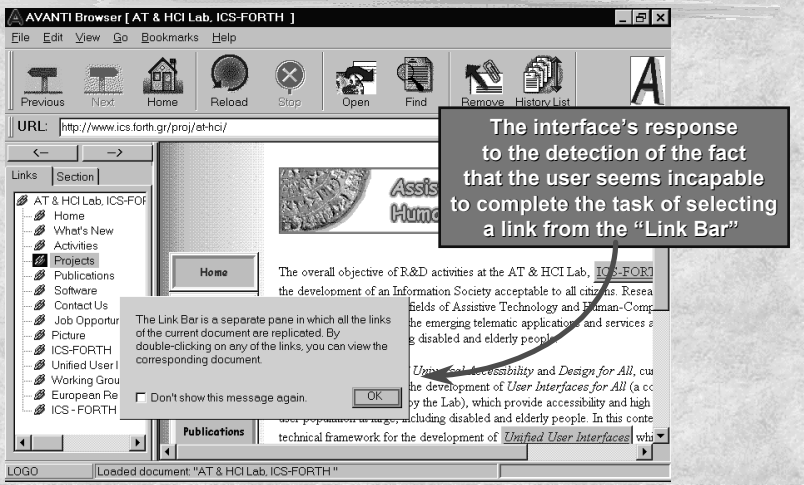
Automatic User Interface Adaptation *(3/9)*



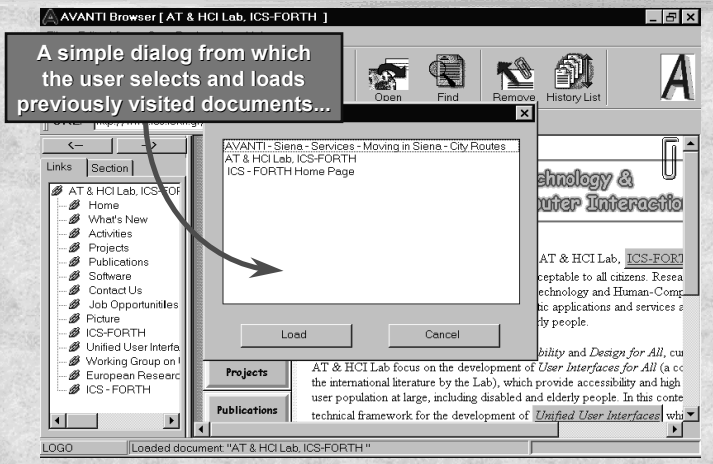
Automatic User Interface Adaptation *(4/9)*



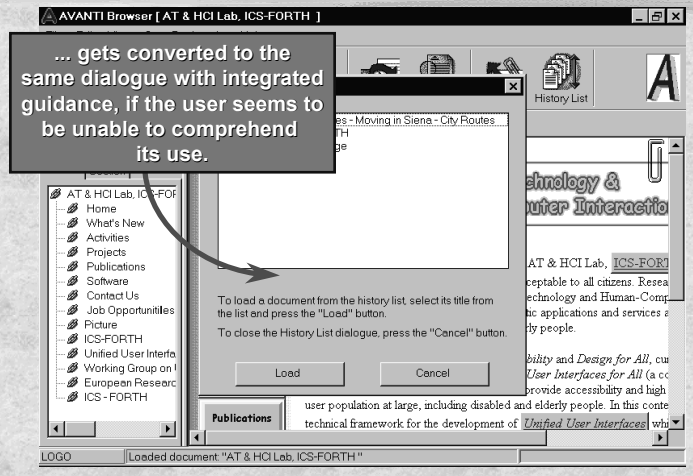
Automatic User Interface Adaptation *(5/9)*



Automatic User Interface Adaptation *(6/9)*



Automatic User Interface Adaptation *(7/9)*



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

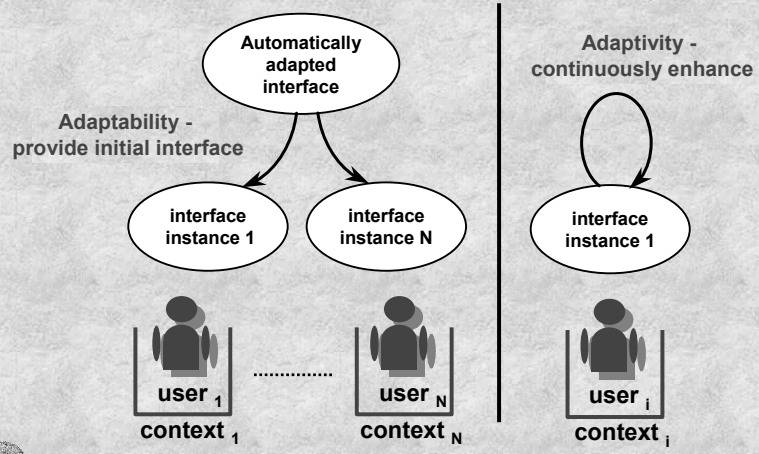
*Interface adaptation applied on the basis of **on-line** knowledge*

⇨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



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

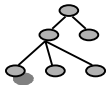
– A ***repository*** of implemented dialogue artifacts, out of which the most appropriate are selected at run-time

- ***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

semantic	internal functionality, information represented
syntactic	dialogue sequencing, syntactic rules, user tasks
constructional	
physical	devices, object attributes, interaction techniques

Polymorphism in Unified User Interfaces *(1/6)*

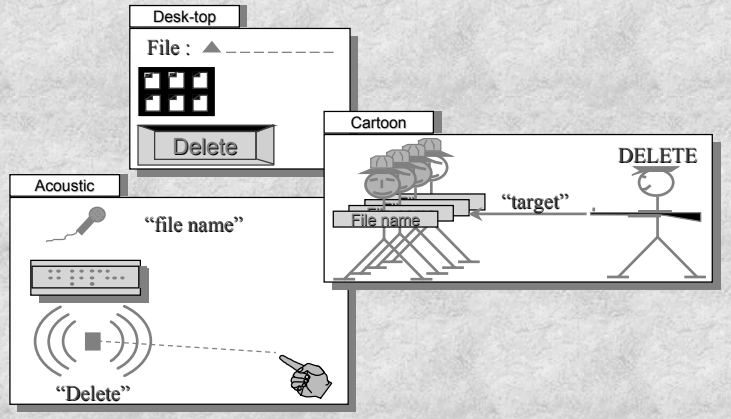
- Automatic adaptation at any level implies the ability to ***polymorphose***
  - i.e., for a given task, design alternative interactive artifacts according to different user- and usage-context- attribute values

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



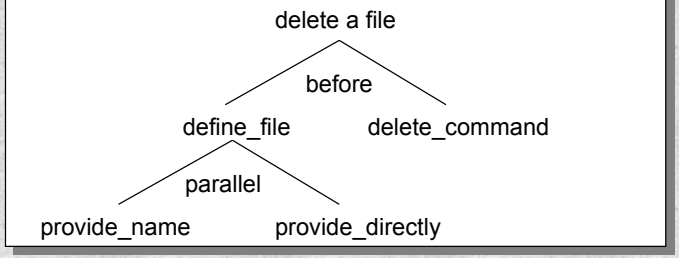
Polymorphism in Unified User Interfaces *(4/6)*

- *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

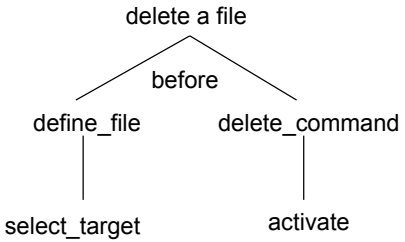
delete file = define file **before** delete command
define file = provide name **or** select directly



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



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

The Need

- For any given task, *diverse* user- / usage-context- attribute values may dictate the design of *alternative* dialogue patterns
- Hence, given a design context and task, the parameters of the design *space* may map to *more* than one interaction *artifacts*
- However, current design practices impose a *single interaction artifact* in the final outcome

Key Properties *(1/2)*

- *Polymorphic task* analysis, where any task may be decomposed into an arbitrary number of *alternative sub-hierarchies*
- *Hierarchical decomposition* of user tasks, starting from the abstract level, by incrementally specialising, in a polymorphic fashion, towards the physical level of interaction

Key properties *(2/2)*

A complete method should comprise (Moran,1996)

- –a statement of the problem
- –a device (technique, tool or representation)
- –a procedure for using the device
- –a clear set of outcomes

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*)

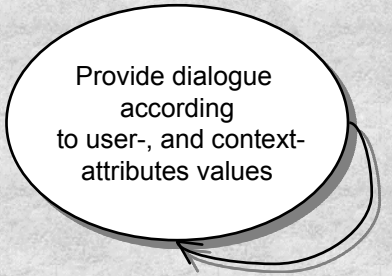
- ***Unified Interface Architecture***
- Adaptation Scenarios and control flow
- Embedding design into implementation

Revealing an *Architectural Pattern*

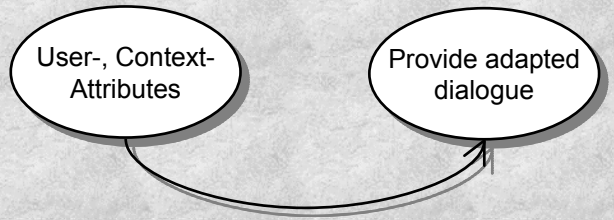
Vehicle

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

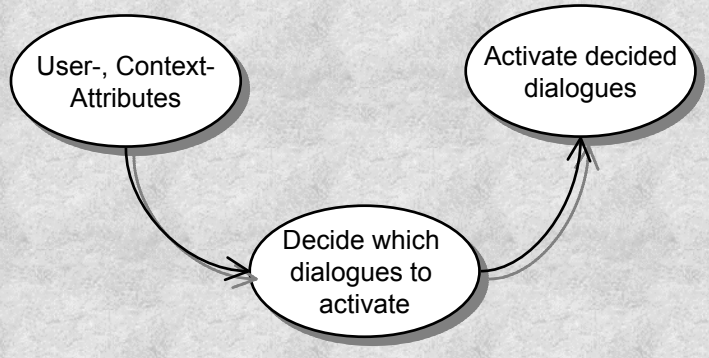
Starting point



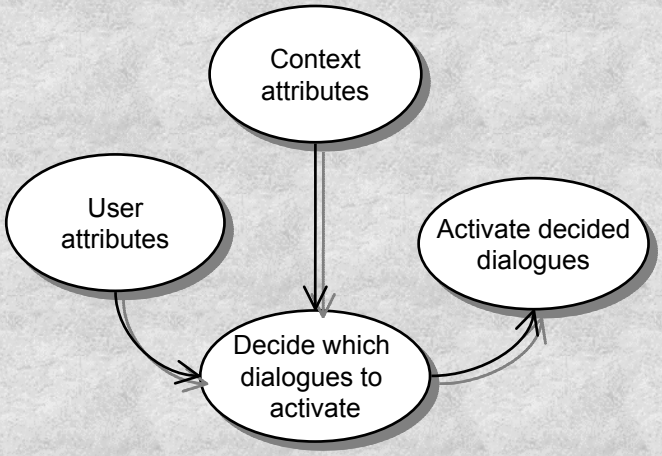
Revealing an *Architectural Pattern (1/9)*



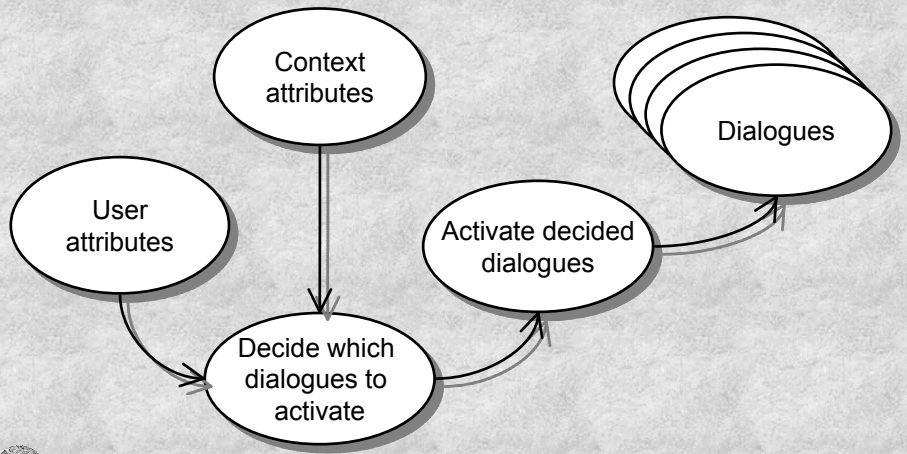
Revealing an *Architectural Pattern (2/9)*



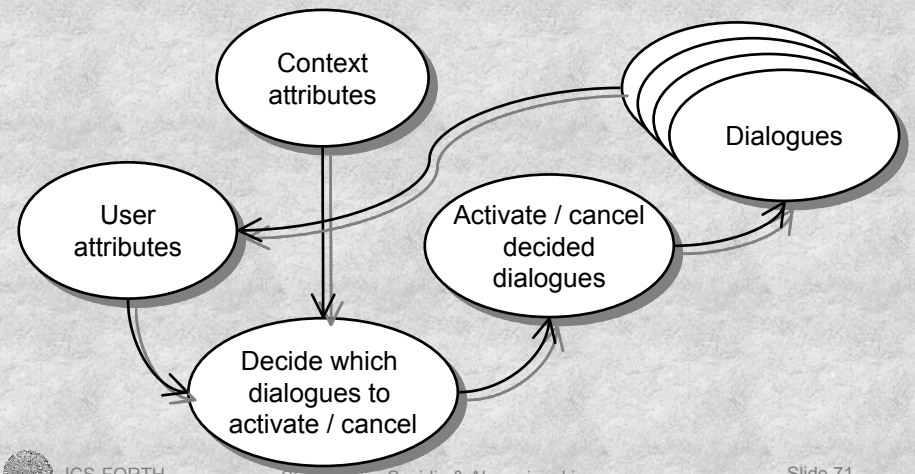
Revealing an *Architectural Pattern (3/9)*



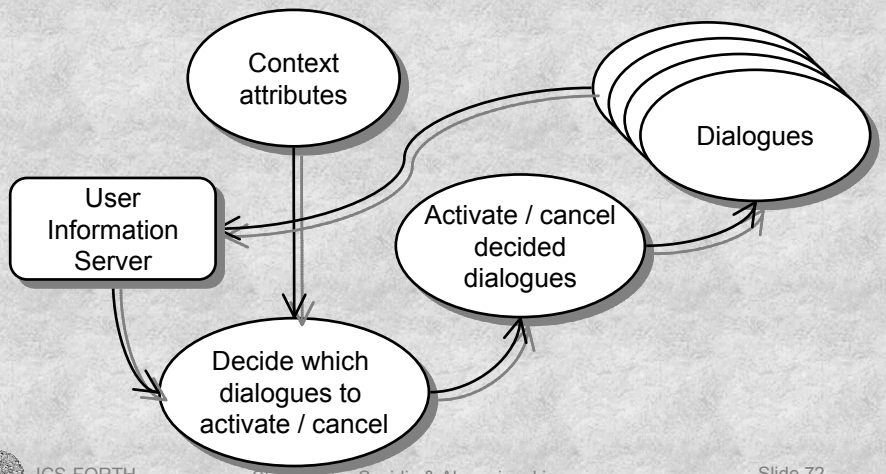
Revealing an *Architectural Pattern (4/9)*



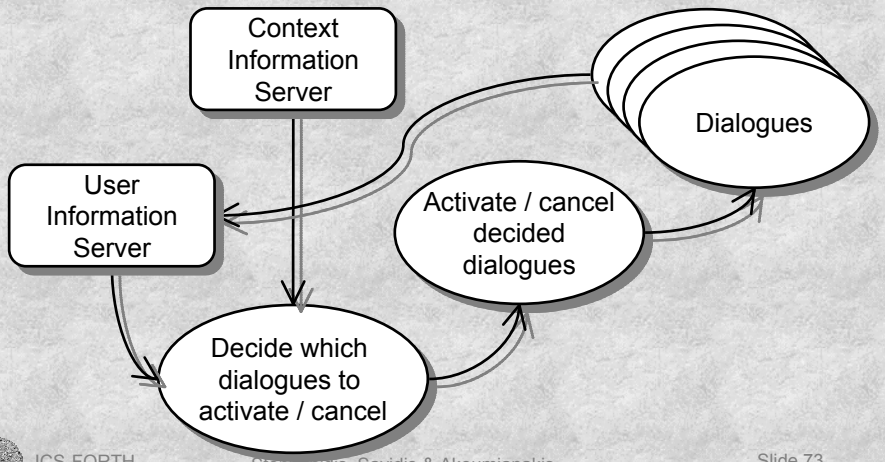
Revealing an *Architectural Pattern (5/9)*



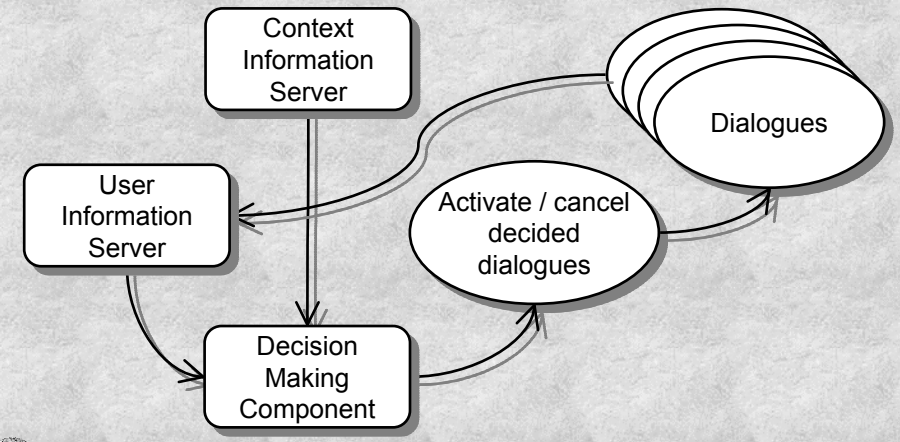
Revealing an *Architectural Pattern (6/9)*



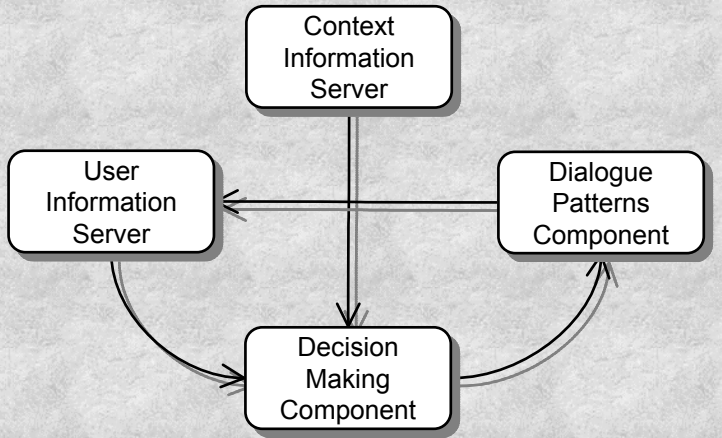
Revealing an *Architectural Pattern (7/9)*



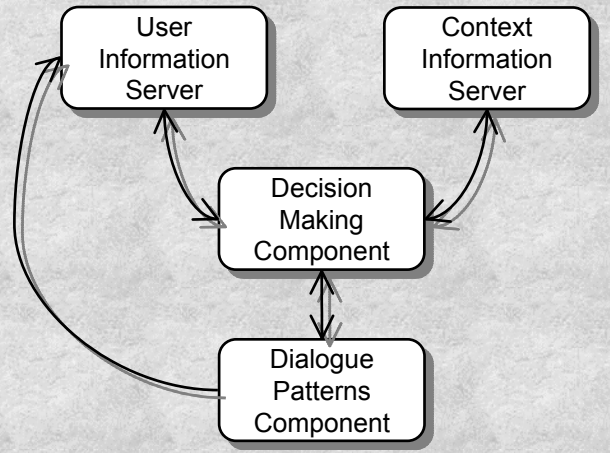
Revealing an *Architectural Pattern (8/9)*



Revealing an *Architectural Pattern (9/9)*



Unified Interface Architecture



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 (***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

Parameters	Value domain
P_1	✓
P_2	✓
⋮	⋮
P_n	✓

User profile instance

computer knowledge	expert	**frequent**	average	casual	native
Web knowledge	very good	good	average	**some**	limited	none
ability to use left hand	perfect	good	some	limited	**none**

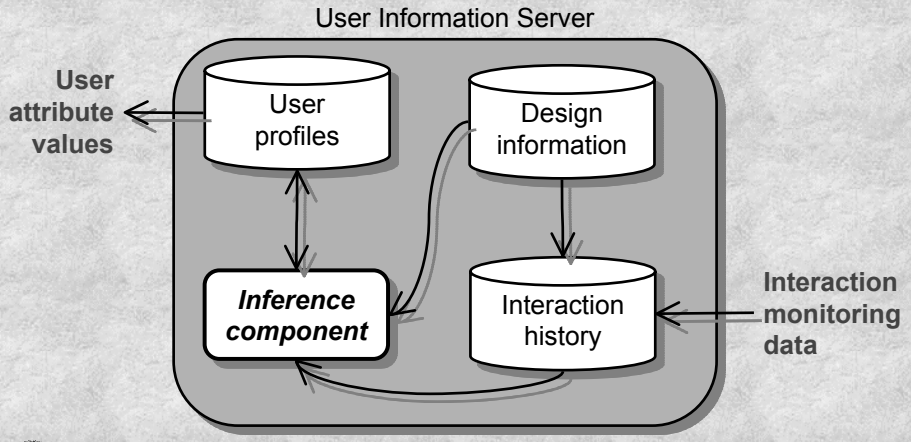
User Information Server - *Communication*

- 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

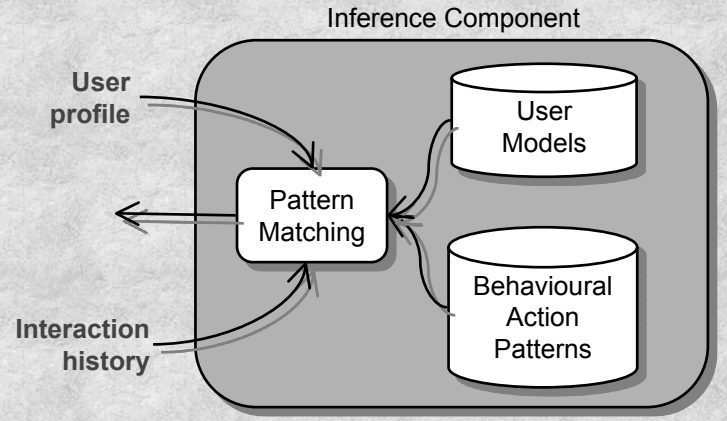
User Information Server - *Implementation (1/3)*

- 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)*



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***

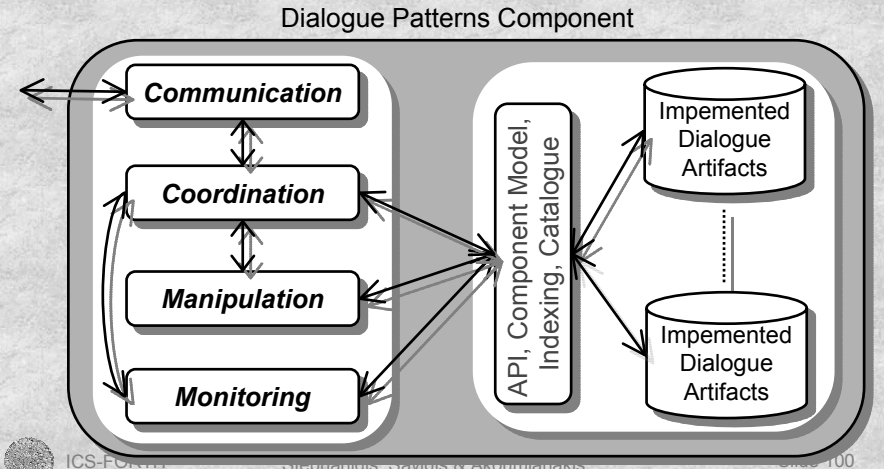
Dialogue Patterns Component - *Communication*

- 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*

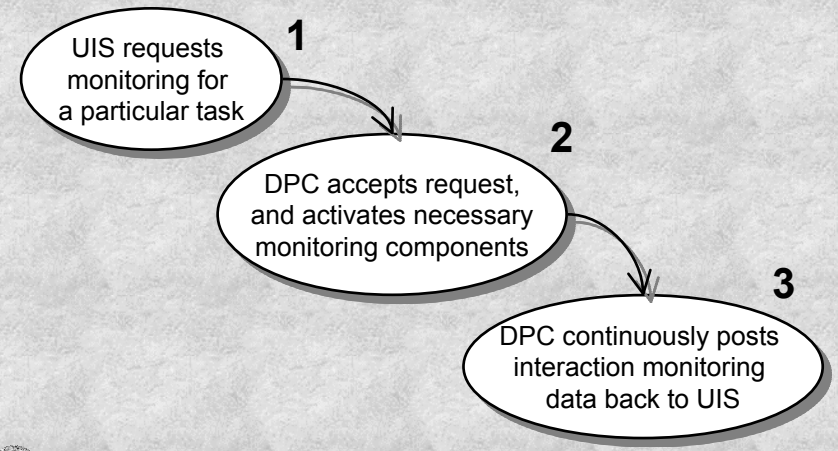
Unified Interface Engineering (*agenda*)

- Unified Interface Architecture
- ***Adaptation Scenarios and control flow***
- Embedding design into implementation

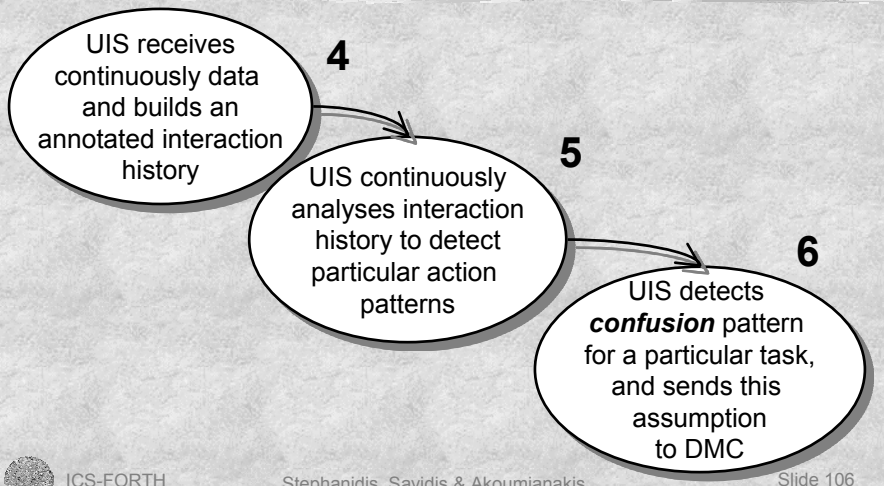
Adaptation Scenarios and Control Flow – *Example (1/5)*

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

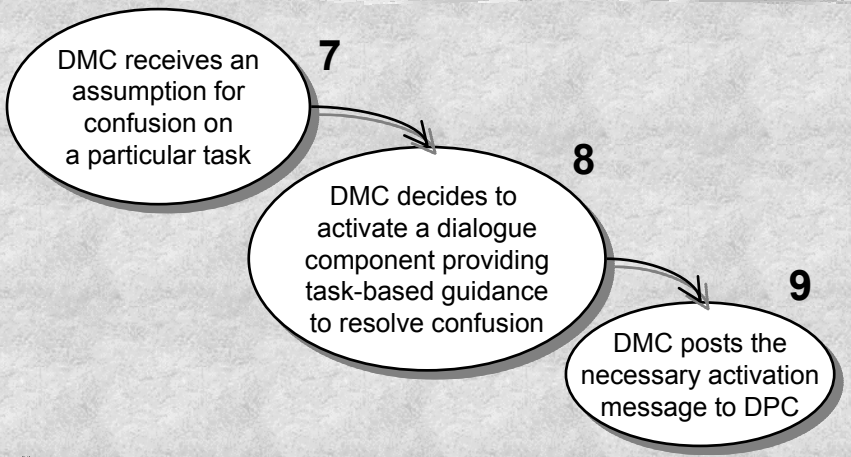
Adaptation Scenarios and Control Flow - *Example (2/5)*



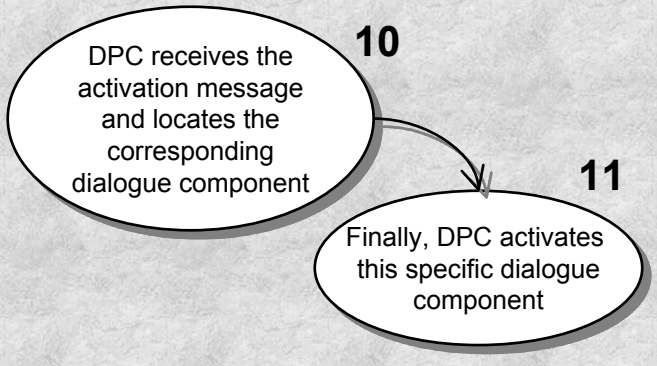
Adaptation Scenarios and Control Flow - *Example (3/5)*



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



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



Unified Interface Engineering (*agenda*)

- Unified Interface Architecture
- Adaptation Scenarios and control flow
- ***Embedding design into implementation***

Embedding Design into Implementation – *Definition (1/2)*

- The purposeful engagement of ***design information into*** the interface ***programming*** process.

Embedding Design into Implementation - *Definition (2/2)*

⇨in a way which enables the s/w implementation to reflect ***automatically design updates*** without the need of extra programming efforts

⇨in a way which promotes reusability of s/w ***by directly linking to design entities***, instead of programming concepts

Levels of Embedded Design Information *(1/3)*

- Lexical
  - interaction objects, attributes, methods, e.g. ***toolkits***
    - PushButon, Menu, CheckBox
    - x, y, width, height, fgColor, bgColor
    - Press, Select, Check / UnCheck

Levels of Embedded Design Information *(2/3)*

- Syntactic
  - User-task, user- / context- attributes, design goals, e.g. ***unified interfaces***
  - Indexing and location of dialogue artifacts based on artifact design documentation parameters
  - Automatic adaptation plays the role of a *dynamic design process*

Levels of Embedded Design Information *(3/3)*

- Semantic
  - domain entities, e.g. ***databases***
  - Database schema is both a design and an implementation entity
  - Querying is both an implementation mechanism and a user tool

Judging Interface Implementation Maturity *(1/3)*

- The smaller the ***distance between design and implementation*** models, the higher the maturity

Example-I

- Database development
  - Data models
  - Data input
  - Query models
  - Form construction

Example-II

- Unified development
  - Dialogue patterns and their relationships
  - User- / context- parameters
  - User tasks and adaptation logic

Judging Interface Implementation Maturity *(2/3)*

- The easier to locate from ***design units***, their respective ***implementation units***, and vice versa, the higher the maturity

Example

- Interface toolkits (*embedding only part of lexical level design*)
  - Objects classes and attributes
  - Methods
  - Event handlers
  - Part-of / parent-of relationships (containment versus instance hierarchies)

Judging Interface Implementation Maturity *(3/3)*

- The higher the ***capability*** of the s/w ***to cope*** effectively and efficiently ***with variations of the design parameters***, i.e. ability to adapt, the higher the maturity

Example

- Unified interface implementation
  - *Encapsulation* of design parameters (user- / context- information)
  - Interface *self-adaptation* reflecting alternative instances of design parameters
  - "*On-the-fly*" interface *assembly* process from a pool of implemented dialogue patterns

Unified Interface Engineering - *Wrap Up (1/2)*

- Distributed s/w architecture, separating:
  - Decision making
  - User- / context- information
  - Artifact implementation

Unified Interface Engineering - *Wrap Up (2/2)*

- Design-oriented interface engineering:
  - Ability to adapt to changing design parameters
  - Embedding of design information directly into the implementation

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***

Tools for Unified Interfaces - Introduction *(4/7)*

- Since designers may employ any particular interaction toolkit, the interface tool employed for implementation should enable programmers to utilise this toolkit
  - ***Toolkit integration***

Tools for Unified Interfaces - Introduction *(5/7)*

- 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

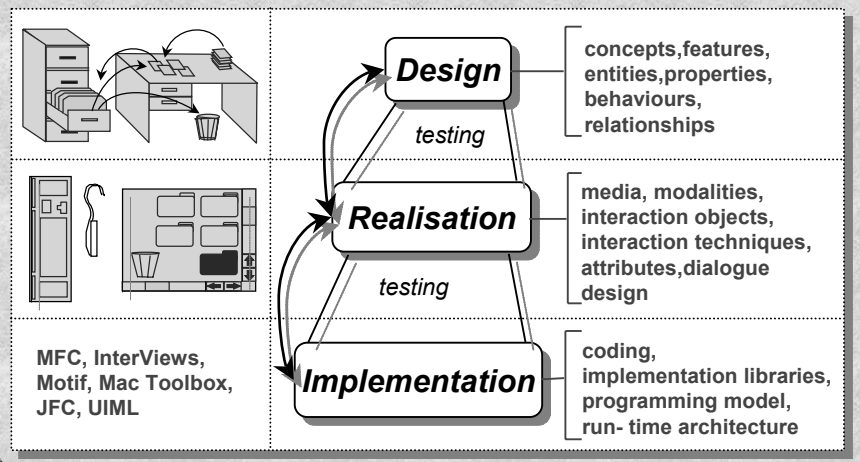
Metaphor Development - *Why ? (1/5)*

- Interaction metaphors are user-oriented
  - Design should reflect the attributes of target users
  - It is unlike 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)*



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*

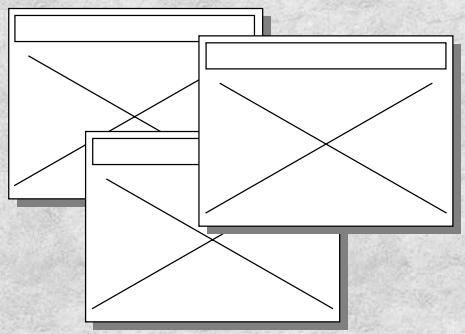
Metaphor Development - *Advantages (5/5)*

- 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*

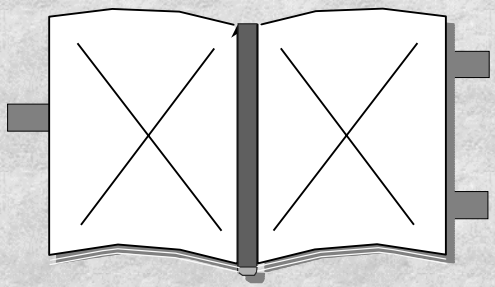
- Top-level ***container*** interaction objects play the most important role

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



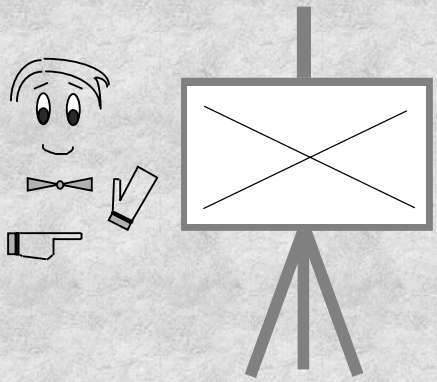
Windowing / Desk-top Metaphor ?

Top-level Containers - *Key Role (2/5)*



Books Metaphor ?

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



Teacher / Whiteboard Metaphor ?

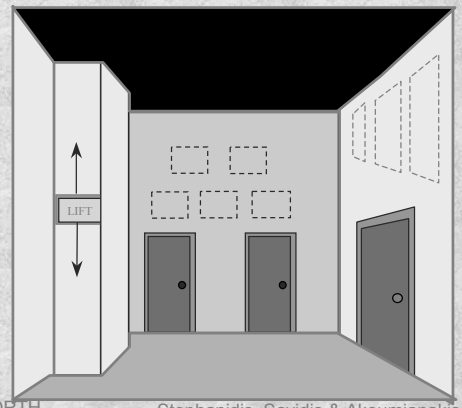
Top-level Containers - *Key Role (4/5)*

- 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 bellow

Interaction object

Group: “floor”, “ceiling” and front, left, back, right “walls”

Top Level Containers – A Step Foward

- 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 - *Functional Requirements (1/4)*

- ***Unified programming*** model for imported toolkits
  - All imported toolkits manipulated in the same way
    - Creating / destroying objects
    - Getting / setting attributes
    - Adding / removing event handlers / methods

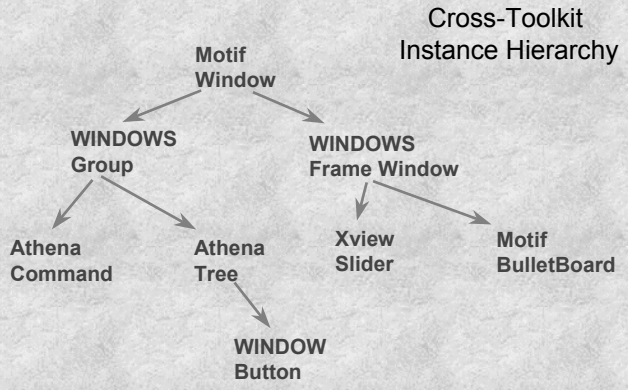
Toolkit Integration - *Functional Requirements (2/4)*

- ***Metaphor-independent*** toolkit integration model
  - Toolkits not only bounded to the windowing metaphor must be importable
    - e.g. cartoon, playground, watch-dog, whiteboard, rooms,...

Toolkit Integration - *Functional Requirements (3/4)*

- Ability for ***cross-toolkit*** object hierarchies
  - A container from one toolkit is enabled to encompass objects from other toolkits.
    - Requires ***toolkit interoperability***, currently supported only among windowing toolkits

Toolkit Integration - *Functional Requirements (4/4)*



Toolkit Integration - *Support by Existing Tools (1/3)*

- ***Multi-platform*** toolkits
  - Mainly re-implementations over multiple windowing toolkits
  - Cannot add a new toolkit; vendors should do that
  - Examples : XVT, YACL, Amulet, JFC

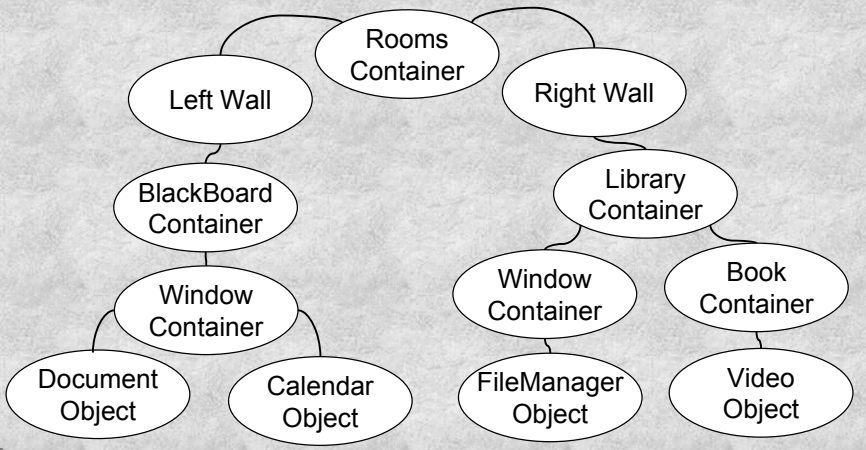
Toolkit Integration - *Support by Existing Tools (2/3)*

- Java ***Pluggable Look&Feel*** ?
  - Mainly ***parameterisation*** of visual 2D display structure
  - Limited to keyboard / mouse based interaction
  - Making a new look&feel, e.g. 3D / cartoon / non-visual rooms, ***is not*** possible through the PL&F API

Toolkit Integration - *Support by Existing Tools (3/3)*

- ***Component technologies*** (ActiveX, Java Beans) ?
  - Provide the ground for toolkit interoperability, as soon as component-ware compliant toolkits are built
  - Interesting results will appear when compliance is achieved for toolkits realising different interaction metaphors

Toolkit Integration - Cross-Metaphor Interoperability



Toolkit Integration - Cross-Toolkit Interoperability

- A very challenging research topic
  - Real-time containment and space negotiation protocols
  - Polymorphic display projection
  - Attribute transformation
  - Alternative I/O bindings supporting dynamic configuration
  - Drop-out policy support

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 - *Functional Requirements (1/4)*

- ***Unchanged programming model***
  - New attributes, methods and interaction techniques should be made available through the *original programming* approach

Toolkit Augmentation - *Functional Requirements (2/4)*

- ***Object hierarchy management and focus object manipulation***
  - Need to be able to *set* / *get* / *be-notified* regarding focus object, and also handle programmatically the interaction object *instance hierarchy*

Toolkit Augmentation - *Functional Requirements (3/4)*

- ***Programmatically extensible constructor***
  - Well documented *placeholder*, where additional interaction behaviour, attribute defaults, and hierarchy linkage can be set

Toolkit Augmentation - *Functional Requirements (4/4)*

- ***Device installation and integration layer***
  - New *I/O devices* need to be installed, while the availability of those devices should be enabled through original I/O handling facilities

Toolkit Augmentation - *Support by Existing Tools*

- Programming toolkits offer facilities for augmentation, however:
  - New object classes need to be defined (mainly through inheritance)
  - Re-compilation / re-linking of old applications is needed to gain augmented behaviour

Toolkit Augmentation - *An Example (1/6)*

- Augmented Windows MFC
- Switch-based access (binary switches)
- Lexical dialogue decomposition into two fundamental actions: **select**, **next**

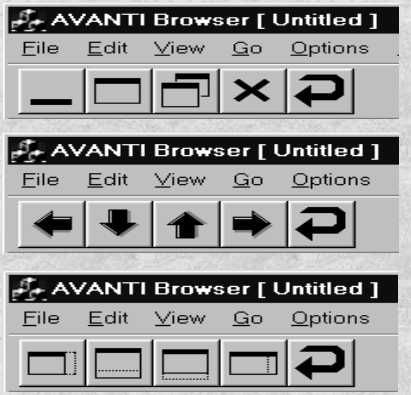
Toolkit Augmentation - *An Example (2/6)*

- 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

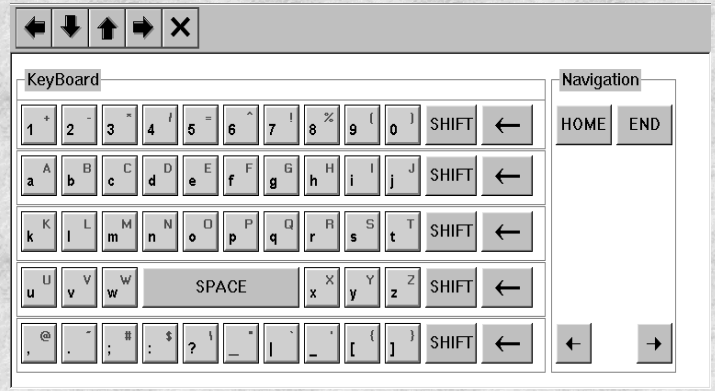
position control

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

Toolkit Expansion - *Functional Requirements*

- ***Provision of an object expansion framework, for introducing***:
  - Object class, attributes, methods, event handlers, interaction techniques, and dialogue implementation

Toolkit Expansion - *Support by Existing Tools (1/2)*

- Interface tools offer various facilities for expansion:
  - Inheritance-based
  - Template structures
  - API implementation
  - Graphical construction

Toolkit Expansion - *Support by Existing Tools (2/2)*

- The only limitation for expansion in existing tools is that:
  - ***New objects should always comply with the original toolkit metaphor***,
    - and in all known tools this is the windowing metaphor

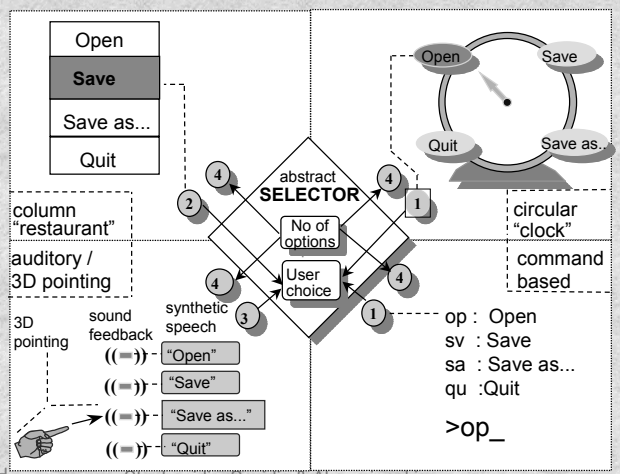
Tools for Unified Interfaces (*agenda*)

- 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*



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

Toolkit Abstraction - *Functional Requirements*

- Two categories of functional requirements: ***basic*** and ***advanced***

Toolkit Abstraction - *Basic Requirements (1/4)*

- Provide a predefined collection of abstract interaction objects
  - *closed set of abstractions*

Example

- Multiple Choice Selector
- Single Choice Selector
- Command
- On / Off State

Toolkit Abstraction - *Basic Requirements (2/4)*

- Support for a predefined mapping scheme, for each abstract object class, to various alternative physical classes
  - *bounded polymorphism*

Example

- On / Off State
  - Radio Button
  - Check Box
- Multiple Choice Selector
  - List Box
  - Combo Box

Toolkit Abstraction - *Basic Requirements (3/4)*

- Ability to select which of the alternative physical classes (in the mapping scheme) will be instantiated, for any abstract object instance
  - *controllable / conditional instantiation*

Example

```
instantiate OnOffState,
scheme RadioButton;

instantiate MultipleChoiceSelector,
scheme ListBox;

instantiate Command,
scheme default;
```

Toolkit Abstraction - *Basic Requirements (4/4)*

- Support for more than one concurrently active physical instances, for any abstract object instance
  - *plural physical instantiation*

Example

- Dual interfaces supporting concurrent visual and non-visual interaction, require abstract objects to have dual instantiation

Toolkit Abstraction - *Advanced Requirements (1/4)*

- Mechanism for the definition of new abstract object classes
  - *open set of abstractions*

Example

```
abstract      SingleChoiceSelector {
  integer     NumberOfOptions;
  integer     UserSelectedOption;
  method      Selected;
}
```

Toolkit Abstraction - *Advanced Requirements (2/4)*

- Mechanism for the definition of alternative schemes for mapping abstract object classes to physical object classes
  - *– open polymorphism*

Example

```
mapping  OnOffState for Windows {
  scheme   RadioButton { ... }
  scheme   CheckBox { ... }
  default RadioButton;
}
```


Toolkit Abstraction - *Advanced Requirements (3/4)*

- Capability to define run-time relationships between an abstract instance and its various concurrent physical instances
  - *– programmable physical mapping logic*

Example

```
always   OnOffState.state =
         RadioButton.state;

when     RadioButton.StateChanged
notify   OnOffState.StateChanged;
```

Toolkit Abstraction - *Advanced Requirements (4/4)*

- Direct programming access, through abstract object instances,to all associated concurrent physical instances
  - *physical instance resolution*

Example

```
OnOffState.Windows.x = y = 10;
OnOffState.Windows.fgColor = "blue";
OnOffState.Windows.bmp = "autofmt.bmp";

OnOffState.Hawk.audio="Click.wav";
OnOffState.Hawk.title="Auto format";
```


Toolkit Abstraction - *Support by Existing Tools (1/3)*

- Neither the basic, nor the advanced set of functional requirements are satisfied by commercial interface tools
  - multi-platform objects offer configurable / extensible "look", however, they ***are not*** abstractions

Toolkit Abstraction - *Support by Existing Tools (2/3)*

- Some research tools satisfy the basic requirements, though only for the windowing metaphor
  - they fail to supply abstractions not bounded to a single metaphor

Toolkit Abstraction - *Support by Existing Tools (3/3)*

- The advanced set of functional requirements is currently satisfied only by the I-GET UIMS
  - A 4GL interface tool designed and implemented so as to support the engineering of the Dialogue Patterns Component in unified architecture

About I-*GET (1/2)*

- An outcome of the ACCESS Project
- Public release intended for 2001
- Windows 95 / 98 / 2000 / NT
- Linux / Unix Variants
- Integrated Windows MFC, Xt/Xaw, HAWK
- Intuitive remote cross platform execution of components

About I-*GET (2/2)*

- C dialect language kernel
- Constraints, preconditions, monitors
- Dialogue control agents and ERS-based event handlers
- Functional API based on shared-space and message channels

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

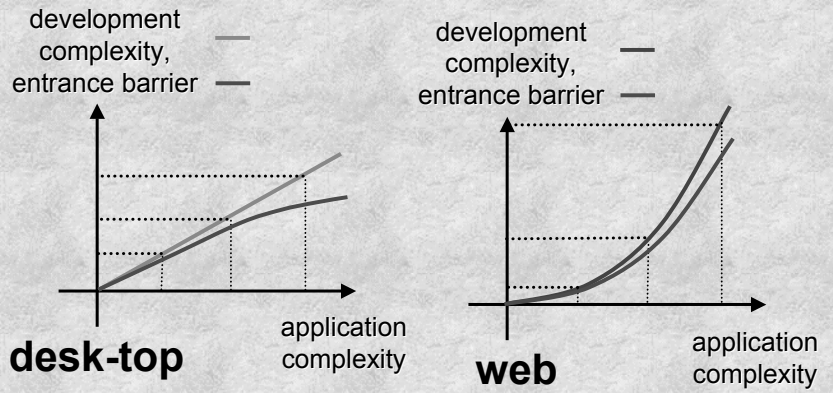
Browsers as Interface Tools *(8/10)*

- 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)*



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
- Browsers for motor-impaired users
  - AVANTI

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

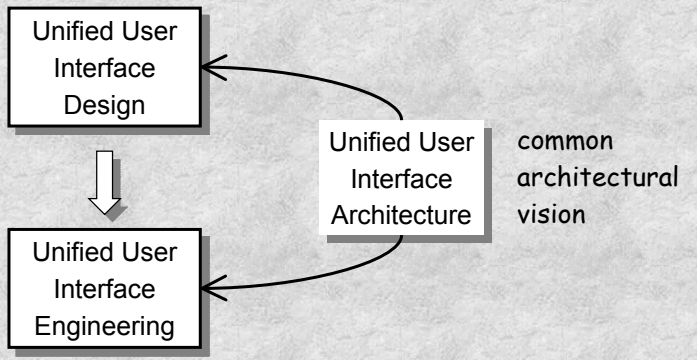
Challenges and Future Work

- Software development process
- Identifying diversity
- Diversity-based optimal design
- Computing platforms and embedded OS
- ***Concluding remarks***

Unified User Interface Development - *agenda*

- ***Overview***
- Unified User Interface design
- Unified User Interface engineering
- Tools for developing Unified User Interfaces

Unified User Interface Development - *Overview*



Unified User Interface Development - *agenda*

- Overview
- ***Unified User Interface design***
- Unified User Interface engineering
- Tools for developing Unified User Interfaces

Unified User Interface Design - *agenda*

- Motivation
- Method outline
- Concepts
- Process description
  - Designing alternative interaction artefacts
  - Re-engineering designs via abstract objects

HCI design methods

- Classification
  - micro-methods organize the overall design process e.g., Human-centered design
  - macro-methods address specific issues e.g.,
    - Requirements capture
    - Dialogue design
    - Evaluation
- Plethora of methods

A definition

- According to (Olson & Moran, 1996) a complete macro-method comprises
  - a statement of the problem
  - a device (technique, tool or representation)
  - a procedure for using the device
  - a clear set of outcomes

The Need

- Capture the global execution context of a task
- Task execution context is dictated by
  - user abilities and preferences
  - technological platform
  - context-of-use
- Current design practices impose a *single interaction artefact*
  - assumption on "average" user, desktop platform and business context of use

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 macro-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

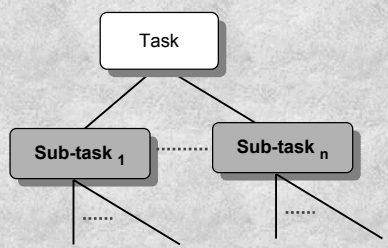
  Recorded rationale for alternative design patterns

Unified User Interface Design - *agenda*

- Need and key properties
- ***Polymorphic task hierarchies***
- Process description
  - Designing alternative interaction artefacts
  - Re-engineering designs via abstract objects

Polymorphic Task Hierarchies - *Properties*

- Hierarchical task analysis
- Polymorphism
- Task operators, including
  - sequencing
  - parallelism
  - exclusive selection
  - repetition

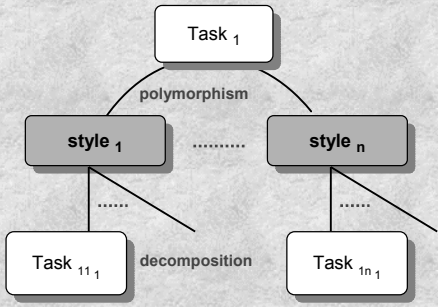


The polymorphic task hierarchy

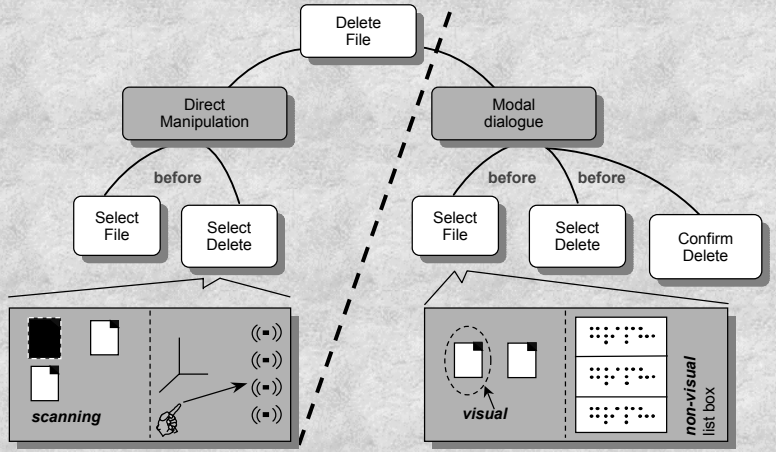
- Root represents design abstractions
- Leaf nodes represent concrete interaction components
- Polymorphic decomposition leads from abstract design pattern to a concrete artefact

Styles

- Each alternative decomposition is called a decomposition style, shortly a style
- Styles can be further analysed through any other appropriate design method
  - Heuristics, GOMS analysis, Traditional HTA, UAN, Formal specifications



An example



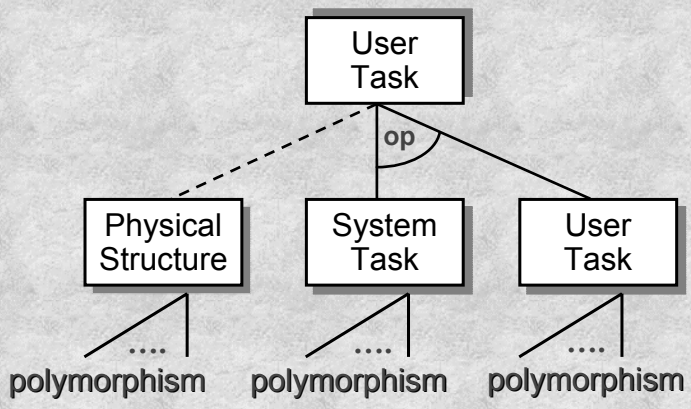
What can be polymorphosed? *(1/2)*

- Three main polymorphic artifacts
  - User tasks
  - System tasks
  - Physical (or concrete) interface elements

What can be polymorphosed? *(2/2)*

- User tasks → *the need for designing alternative paths in task accomplishment*
  what the user has to do (e.g. manipulate file)
- System tasks → *the need for designing alternative feedback methods*
  what the system has to do (e.g. feedback)
- Physical structure → *the need for designing alternative physical components in which associated user / system tasks may be performed*
  Interface components (e.g. a window) in the context of which user actions are to be performed; always associated to user- or system- tasks

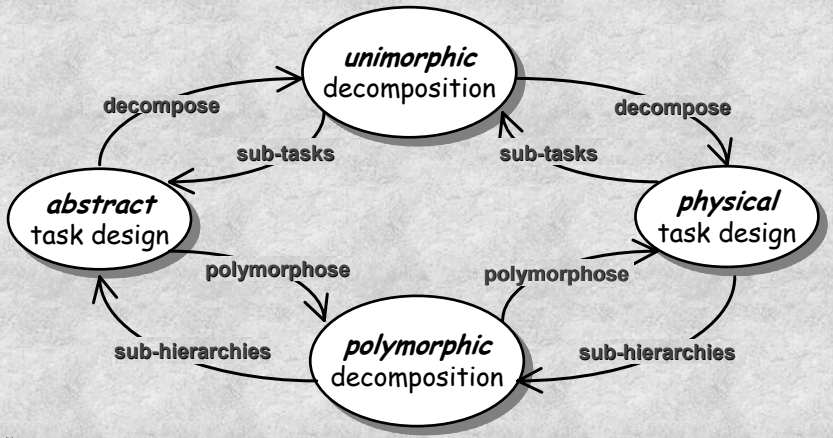
Polymorphic Task Hierarchies - *Task Categories*



Unified User Interface Design - *agenda*

- Need and key properties
- Polymorphic task hierarchies
- ***Process description***
  - Designing alternative interaction artefacts
  - Re-engineering designs via abstract objects

Polymorphic Task Decomposition- *Process Overview*



Polymorphic Task Decomposition- *Alternative paths*

- Abstract versus Physical tasks
- Unimorphic versus Polymorphic tasks
- Sub-tasks versus sub-hierarchies

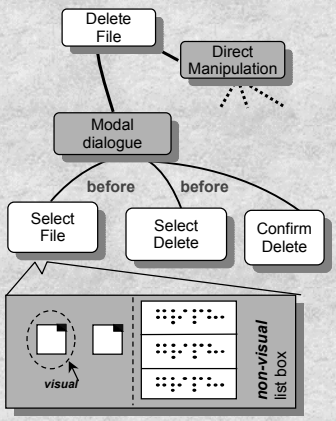
Polymorphic Task Decomposition- *Process Description (1/2)*

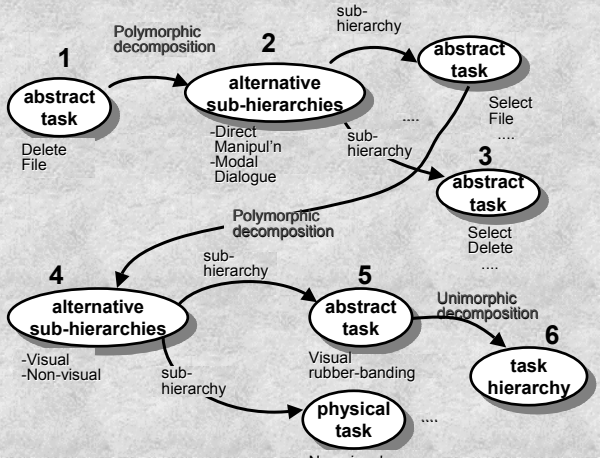
- Start from ***abstract task design*** if given task is not bound to the physical means of interaction, and then
  - ***polymorphose***, if decision parameters impose the need for alternative styles on user- / system- tasks and / or physical structure
  - ***decompose***, when alternative designs are needed for the same style

Polymorphic Task Decomposition- *Process Description (2/2)*

- Start from ***physical task design*** if given task is dependent on the physical means of interaction, and then
  - ***polymorphose***, if decision parameters impose the need for alternative styles on user- / system- tasks and / or physical structure
  - ***decompose***, when an alternative design is needed to realize the same style

Polymorphic Task Decomposition- *Process Instance Example*





Unified User Interface Design - *agenda*

- Need and key properties
- Polymorphic task hierarchies
- Process description
  - ***Designing alternative interaction artefacts (or styles)***
  - Re-engineering designs via abstract objects

Designing Alternative Interaction Artifacts - *Three Fundamental Steps*

- ***Construct*** the space of user / context attributes
- ***Identify*** when polymorphic decomposition is needed, ***then produce*** and enumerate alternative styles
- ***Record*** design rationale for each alternative style

Design parameters *(1/2)*

User attributes

- General computer expertise
- Application domain knowledge
- Work-role in an organisational context
- Motor / sensory / mental abilities
- Particular interaction preferences

Context of use

- Acoustic noise
- Light sources
- Mobility

Platform

- Processor speed, memory, secondary storage
- Peripheral equipment
- Resolution, screen physical size, graphics capabilities

Design parameters *(2/2)*

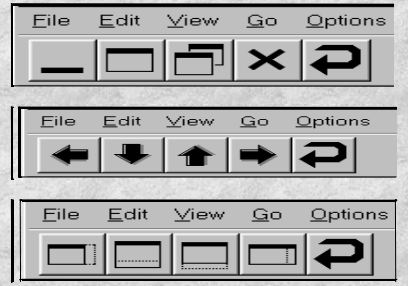
- In practice, only a sub-set of user- and context- attributes will be selected as relevant for affecting the decomposition of a particular task
- In many cases, multiple values for a particular attribute can be satisfied by the design of a single style

When to Apply Polymorphic Decomposition

- 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

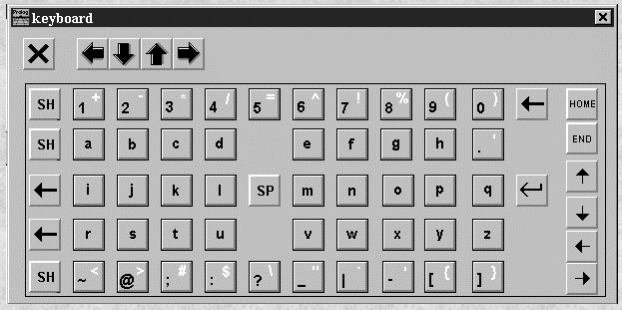
Example: Scanning style(s) *(1/2)*

- Scanning as an alternative style for window management



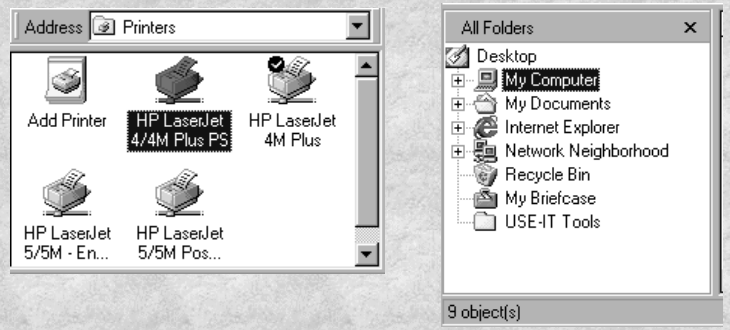
Example: Scanning style(s) *(2/2)*

- Editing using an on-screen keyboard



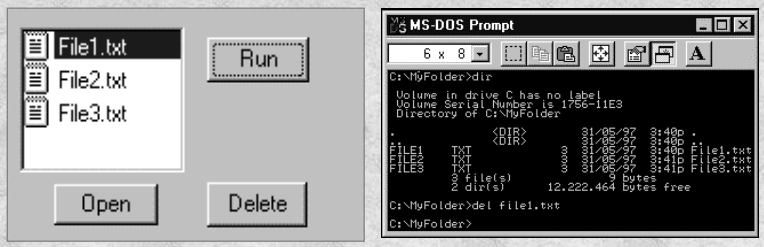
Alternative styles for deleting a file *(1/2)*

- Selecting from a list of interactive file icons or a directory



Alternative styles for deleting a file *(2/2)*

- Specifying the object and then issuing the command



Recording Design Rationale for Alternative Styles *(1/2)*

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

Recording Design Rationale for Alternative Styles *(2/2)*

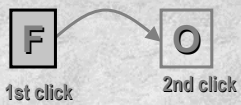
Task: ***Delete File***		
Style :	*Direct Manipulation*	*Modal Dialogue*
Targets :	Speed, naturalness, flexibility	Safety, guided steps
Parameters :	User (expert, frequent, average)	User (casual, naive)
Properties :	Object first, function next	Function first, object next
Relationships :	Exclusive	Exclusive

Analysing styles

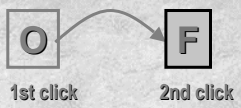
- Styles can be evaluated and compared
  - Performance measures
  - Heuristics
  - User satisfaction, etc
- Evaluation or comparison can form part of the styles design rationale
  - Representation
  - Reasoning about styles

Style analysis example

- Develop styles for command order in direct manipulation dialogues
  - Function-Object style

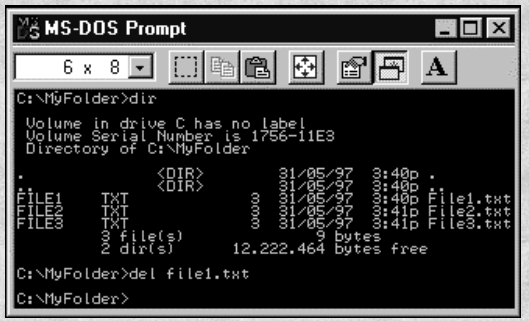


  - Object-Function style



Styles for command order *(1/2)*

- Function-Object style



Styles for command order *(2/2)*

- Object-Function Style

Style analysis

- Experiment
  - Command order as dependent variable
- Analytical criteria
  - Task completion time
  - Efficiency of task performance
  - Frequency of errors
  - Action time
  - Planning time, etc

Results

- 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

Style relationships

Exclusion

- Relates many styles together
- Only one out of those styles can be available to the user during interaction

Compatibility

- Relates many styles together
- Any, some, or all of those styles may be available to the user during interaction

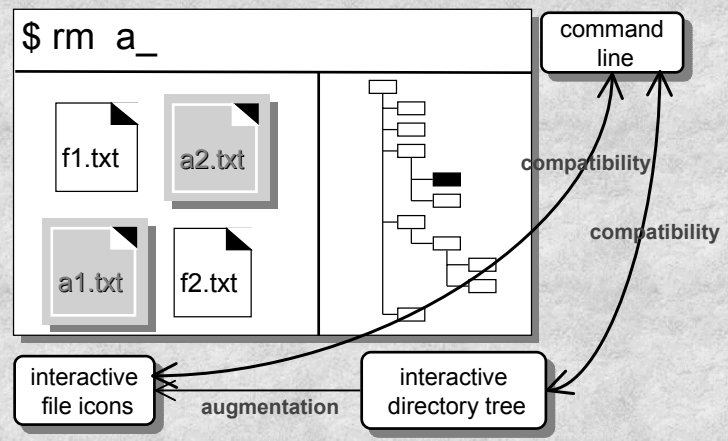
Substitution

- Relates two ordered groups of styles S1 and S2
- When S1 is available during interaction, and at some point S2 should be also made available, S1 must be closed down

Augmentation

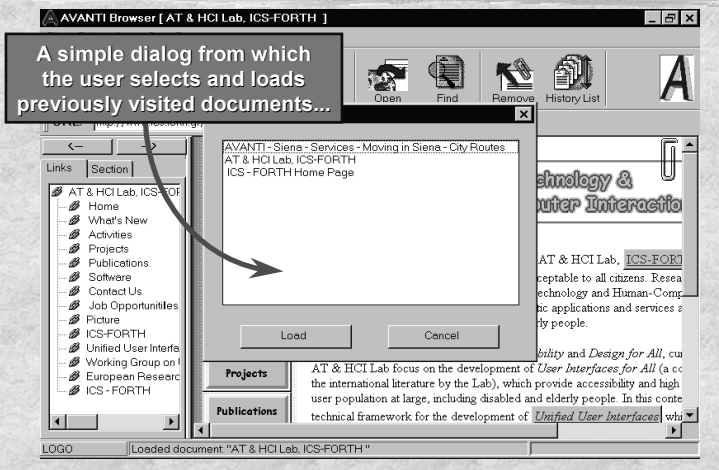
- Relates a single style S1 with a group of styles S2
- If any style belonging to S2 group is available during interaction, S1 may also become available

Style relationship examples: Compatibility



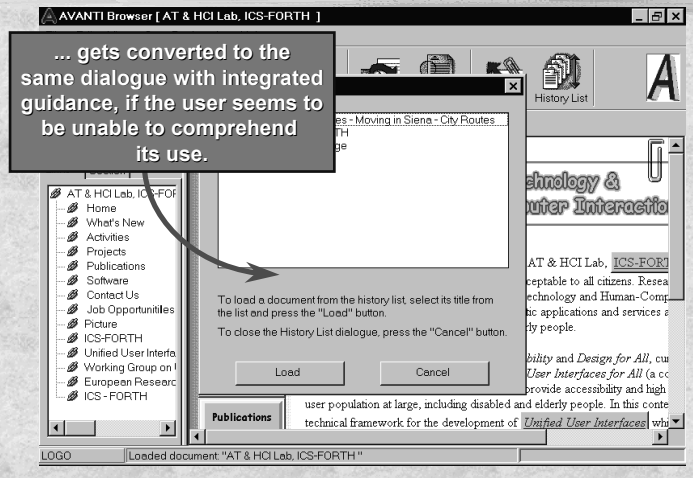
Style relationship examples: Substitution *(1/4)*

S1



Style relationship examples: Substitution *(2/4)*

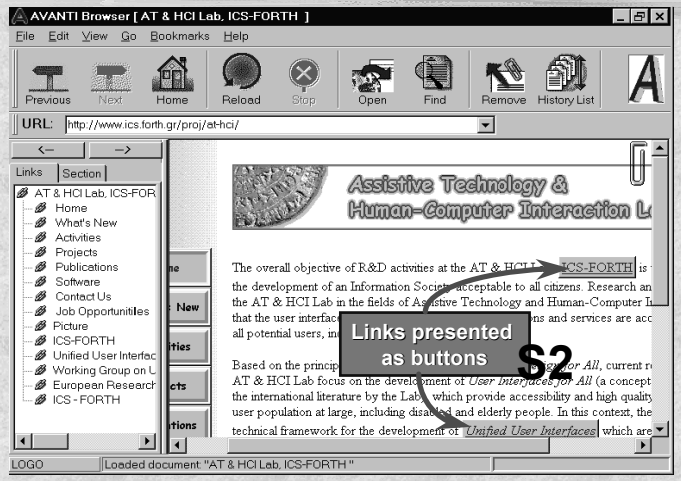
S2



Style relationship examples: Substitution *(3/4)*

<link>	**S1**. Link selection is done as far as the mouse cursor is within the text area of a link and the left mouse button is pressed.
<link>	**S2**. Link selection is done by pressing the software graphical button. *The difference with S1 is that it allows cancellation (by releasing the button while the cursor is outside the button area).*

Style relationship examples: Substitution *(4/4)*



Unified User Interface Design - *agenda*

- Need and key properties
- Polymorphic task hierarchies
- Process description
  - Designing alternative interaction artifacts
  - ***Re-engineering designs via abstract objects***

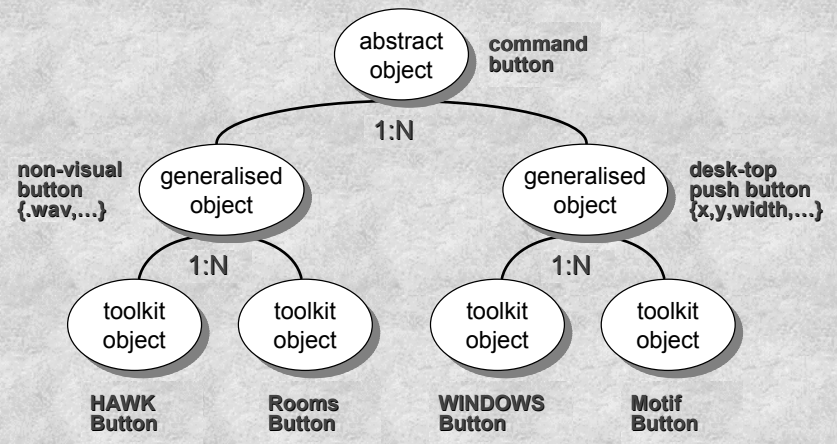
The Importance of Abstract Objects in Unified User Interface design

Capturing abstractions in Unified User Interface design is a powerful technique for creating polymorphic, highly extensible and flexible design artefacts, not tight to specific metaphors or toolkits

Abstract Objects - *Definitions*

- An abstract object is
  - completely relieved from lexical properties and
  - particular metaphoric connotations
- A generalised object exhibits
  - common lexical properties of a particular interaction object class encountered across multiple platforms
- The above distinction is important in order to avoid characterising cross-platform objects as *abstract* objects, since the former are actually *generalised* objects

Abstract Objects - *The Polymorphic Nature*

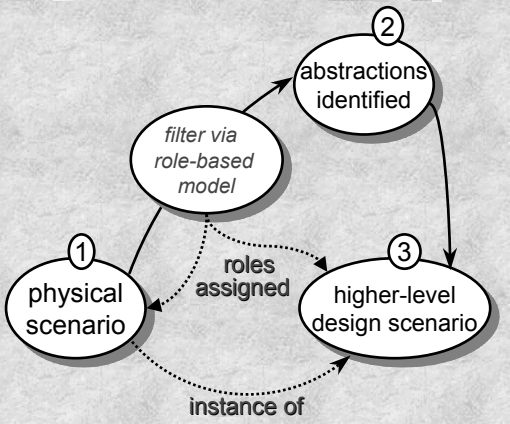


Role of Abstract Objects in Unified User Interface Design

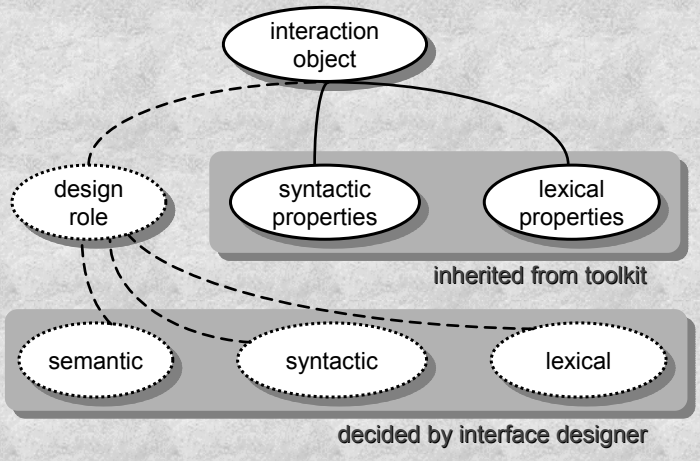
- Role-based model for re-engineering the design of interaction artifacts which identifies abstractions from physically designed artifacts
  - This is necessary since designers, especially graphic designers, primarily think in terms of interaction primitives with a concrete "look & feel"

Re-engineering Approach

- Let **S** be a physical interaction artefact
- Abstract to derive **A** on the basis of design roles
- Document **A** as the abstract design artifact, having **S** as a specific physical design instance
- Generate alternative physical artefacts on the basis of **A**



Role-based Model for Interaction Objects *(1/2)*



Role-based Model for Interaction Objects *(2/2)*

- An object is characterised by its lexical and syntactic properties (i.e. "look & feel"), inherited from the respective s/w toolkit
- In artefacts, engaged objects gain a well defined design role, being either ***semantic***, ***syntactic*** or ***lexical***

Lexical Role - *Definition*

- Lexical role: the object is employed for presentation needs
  - If such a role can be applied independently of physical realisation and overall metaphor of interaction, then an *abstract* object is identified
  - If the role is dependent on a particular metaphor (e.g. desktop), but has a cross-toolkit scope, then a *generalisation* has been captured

Lexical Role - *Example*

- A *message* object, having only one *attribute* defining the message content in textual form
  - Content could be verbal (i.e. the attribute denotes a phrase in natural language), or
  - symbolic (i.e. the attribute indicates a file storing a symbolic sequence - icon, picture, audio, video)

Syntactic Role - *Definition*

- Syntactic role: the object serves a well defined purpose in dialogue sequencing
  - If such a role can be applied independently of physical realisation and metaphor, then an abstraction is identified

Syntactic Role - *Example*

- The "**continue**", "**OK**", "**submit**" buttons, play the role of direct commands given by the user, and are appropriate for *abstraction*

but

- The "**scroll-bar**" applies only to windowing interaction, and is appropriate for *generalisation*

Semantic Role - *Definition*

- Semantic role: the interaction object interactively provides a semantic object
  - Always possible to transform the role into an abstraction

Semantic Role - *Example*

- Typical textual data fields such as "name", "zip code", etc, can be handled via the *textfield* abstraction
- Numeric data fields can be handled via the *valuator* abstraction

An example

- 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

A plausible design solution

Credit Card No: ^_____

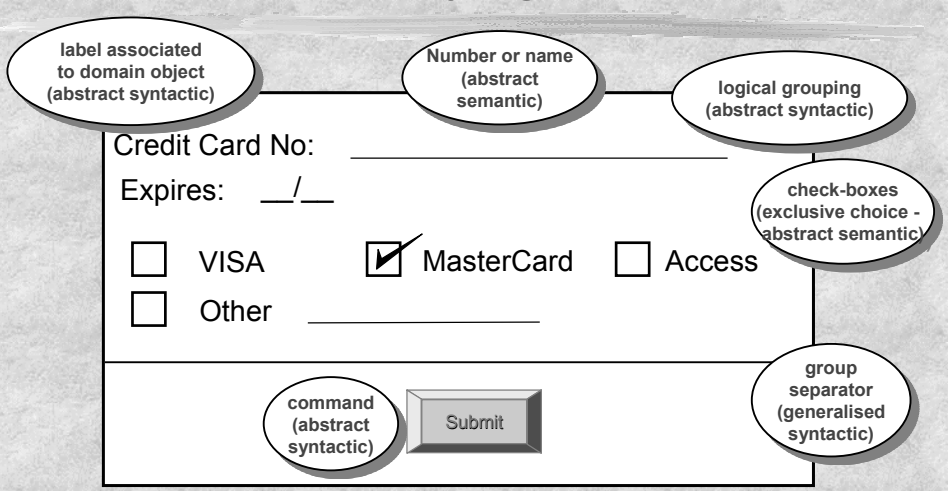
Expires: ^__/__

☐ VISA ☑ MasterCard ☐ Access

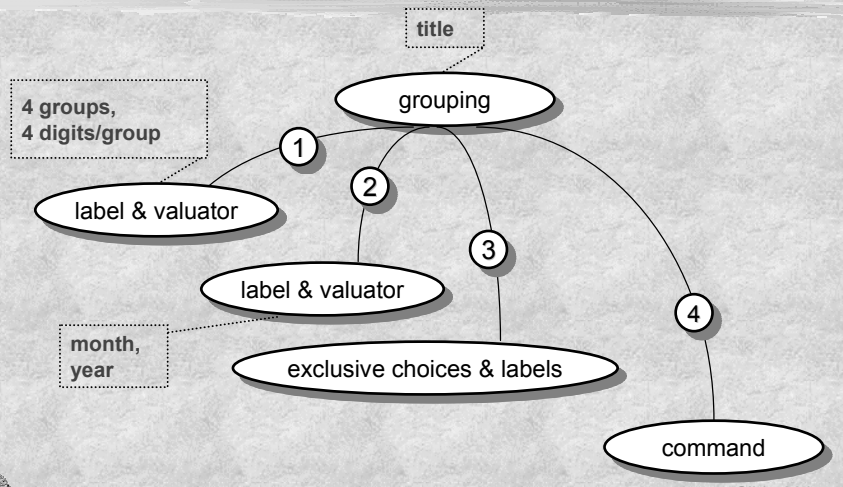
☐ Other ^_____

Submit

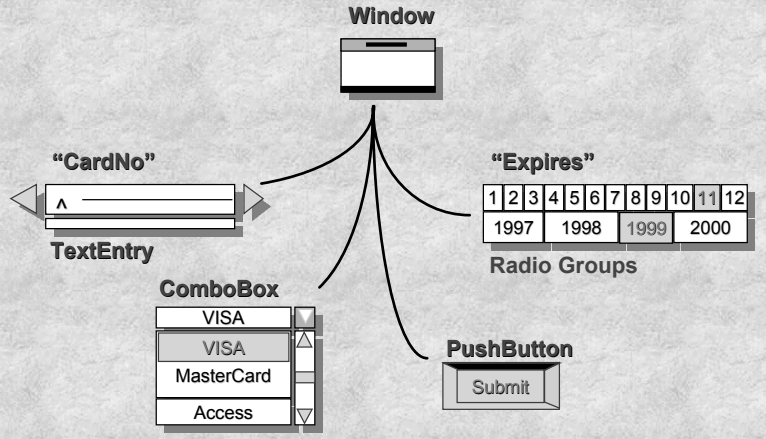
Design Re-engineering Example - *Identifying roles*



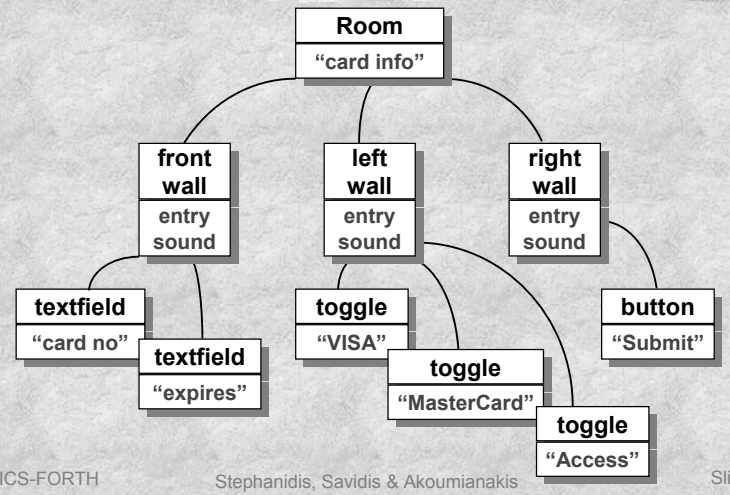
Design Re-engineering Example - *Producing Abstract Model*



Design Re-engineering Example - *Deriving Alternatives (1/2)*



Design Re-engineering Example - *Deriving Alternatives (2/2)*



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

Benefits

- A “middle-out” approach to design
- Useful for designing adaptable and adaptive user interfaces
- It links with analytical approaches to HCI design
- It facilitates design updates

Applications of the method

- In the ACCESS project
  - Design of communication aids for speech-motor and language-cognitive impaired users
  - Non-visual hypermedia for blind
- The AVANTI browser

Links with other HCI design methods

- Unified design is orthogonal to existing design techniques
  - 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)

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