SINTEF

SINTEF ICT

Address: NO-7465 Trondheim, NORWAY
Location: Forskningsveien 1
Telephone: +47 22 06 73 00
Fax: +47 22 06 73 50

Enterprise No.: NO 948 007 029 MVA

SINTEF REPORT

TITLE

State of the Art in Form-based Mobile User Interface Design and Configuration.

AUTHOR(S)

Rolf Kenneth Rolfsen

CLIENT(S)

REPORT NO.	CLASSIFICATION	CLIENTS REF.	
SINTEF A2300	Open		
CLASS. THIS PAGE	ISBN	PROJECT NO.	NO. OF PAGES/APPENDICES
Open	978-82-14-04070-8	90B235	42
ELECTRONIC FILE CODE		PROJECT MANAGER (NAME, SIGN.)	CHECKED BY (NAME, SIGN.)
SOTA Forms based Mobile User Interfaces.doc		Erik G. Nilsson	Erik G. Nilsson
FILE CODE	DATE	APPROVED BY (NAME, POSITION, SIGN.)	
	2007-03-31	Bjørn Skjellaug, Research Manager	

ABSTRACT

This report is based on a State of the Art analysis of existing literature in the field of form based mobile user interface design and configuration.

In the academic literature we do not find many specific design-patterns or guidelines for form-based mobile user interfaces, and it also seams like the many commercial solutions within the field of automatic generation of UI based on proprietary databases and data models are not so easily accessible.

Our state of the art analysis is mainly based on academic papers presented at conferences or in journals, and not so much on online white papers from various commercial actors and products.

We see that it is a large academic activity within the field of model-based UI, i.e. to use different descriptive and abstract descriptions to design, generate and maintain (multi-device) applications and user interfaces.

Many model-based tools and languages specifically address the issue of creating user interfaces targeted at multiple devices. Most take the same basic approach: The designer or developer describes a user interface using one or more abstract models, either directly in the textual modeling language or by using a GUI tool. These models are then rendered into a concrete user interface, either during development or at runtime using a supporting renderer.

KEYWORDS	ENGLISH	NORWEGIAN
GROUP 1	ICT	IKT
GROUP 2	Human Computer Interaction	Menneske-maskin-interaksjon
SELECTED BY AUTHOR	User Interface Design	Brukergrensesnittdesign
	Form-based mobile UI	Formularbasert mobilt brukergrensesnitt
	Model-based mobile UI	Modelbasert mobilt brukergrensesnitt

TABLE OF CONTENTS

1 Introduction

This report is based on a State of the Art analysis of existing literature in the field of form-based mobile user interface design and configuration.

The motivation is that we in the R&D-project FLAMINKO will describe a methodology for how-to develop product or enterprise standards for form-based mobile user-interfaces (UI).

Mobile computing poses a series of unique challenges for user interface design and development: user interfaces must now accommodate the capabilities of various access devices and be suitable for different contexts of use, while preserving consistency and usability.

Screen size and interaction mechanisms are the two main challenges regarding mobile user interface. These challenges are especially applicative for form-based mobile UI. When it comes to screen sizes, forms are hard to scale and less usable for solutions based on scrolling.
When it comes to automatic generation of form-based mobile UI, we find both layout and functionality challenges. A promising solution to handle different and tailor-made layout is to make a stronger separation of content and appearance.

Although professional mobile solutions are at most transaction-oriented, built on top of a database with a form-based UI., we do not find many specific design-patterns or guidelines for form-based mobile user interfaces in the academic literature, and it also seams like the many commercial solutions within the field of automatic generation of UI based on proprietary databases and data models are not so easily accessible.

Our state of the art analysis is mainly based on academic papers presented at conferences or in journals, and not so much on white papers from various commercial actors and products.

We see that it is a large academic activity within the field of model-based UI, i.e. to use different descriptive and abstract descriptions to design, generate and maintain (multi-device) applications and user interfaces.

Many model-based tools and languages specifically address the issue of creating user interfaces targeted at multiple devices. Most take the same basic approach: The designer or developer describes a user interface using one or more abstract models, either directly in the textual modeling language or by using a GUI tool. These models are then rendered into a concrete user interface, either during development or at runtime using a supporting renderer. Sometimes, the models are annotated to give hints to the renderer about how to display the user interface for a particular device.

2 Problem arena: Form-based mobile UI

Form-based UIs do not offer sophisticated selection and navigation techniques to select objects and context-sensitive menus to manipulate them. But even though the currently UI trend is object oriented, the form-based interface will remain with us in the future (Draheim and Weber 2003) (Coldewey and Krüge 1997). Despite all the benefits of object-oriented user interfaces, there are still domains that call for a form-based user interface. Business information systems that support fast processing of few, well-defined use cases are typical examples. The UI as a hole can be a mixture of object-oriented navigation and selection and form-based registration and submission.

Though most modern user interfaces provide object-oriented user interfaces, the need for traditional form-based interfaces persists for several reasons: (Coldewey and Krüge 1997)

- *Often, skilled users need to process well-defined use cases as fast as possible.* The users know what data the system needs and have all the data at hand; they need a user interface, which reflects their work flow and needs as few key strokes as possible. "Media switching" between keyboard and mouse slows down their work.
- *Untrained or semi-skilled users often need step-by-step assistance while performing their tasks.* For instance, an ATM supports only one or two use cases (getting money and displaying account information) but should serve customers of all skill-levels. Because form-based UIs direct the workflow, the user does not necessarily need to know the next step to go.
- *The system might not have enough resources to support an object-oriented UI.* These can cause significant load to a database server while retrieving information of which only a small fraction may be relevant to the user.
- *The existing hardware infrastructure may not support an object-oriented UI.* You need a high-resolution screen and a pointing device to use an object-oriented UI effectively.

One view of form-based UI is that it is based on a paper form metaphor (Draheim and Weber 2003), which corresponds to the statement that "*a form is a document with blanks for inserting information*" (Hekmatpour and Ince 1986). Draheim and Weber claims that many of today's enterprise application use submit/response style as a particular form-based interface type. Form-based interfaces collect user input in the style of input forms. Input forms can be understood as a metaphor, namely as a direct translation of paper forms into human-computer interfaces. Input forms are composite structures of input elements. Submit/response style interfaces is their term for form-based interfaces with a special screen update policy. On a submit/response style interface all screen updates have to be triggered by a dedicated user action. The user actively submits data or a query, instead of watching a constantly updated view. (Draheim and Weber 2003)

The usage of the system is intuitive, since it is guided by the paper form metaphor. The difference between temporary input and submission, or "sending" fits to the business semantics: the two classes of interactions correspond to the work intensive preparation of a contract by page interaction and the punctual and atomic interactions of the "serious" kind, the agreement on a contract by submission of a form. (Draheim and Weber 2003)

Draheim and Weber observe that an important class of systems that use form-based UI is systems with ultra-thin clients. Ultra-thin clients are used for creating an interface tier that does not contain business logic in itself. Ultra-thin clients cache the user interaction on one page in the client layer. Systems with ultra-thin clients are typically multi-user systems. Ultra-thin clients fit neatly into transactional system architectures. (Draheim and Weber 2003)

Normally form-based UI is viewed as a data-oriented UI, like the paper form metaphor (Draheim and Weber 2003) or as a database front end. But the UI can also be viewed as a more task-oriented UI, like forms organized in dialog sequences (Coldewey and Krüge 1997) or wizards that are based on a restricted and rule-based task patterns or work flows.

Resource-based UI is another task- and object-oriented UI approach (Eva 2002). The objective of developing a resource-based design approach is to use resources to make the designer think explicitly about user actions. A resource-based UI is less concern about the workflow of actions, but rather how make the resources (actions) available in a usable way for the user.

A form-based UI is less flexible than an object-oriented UI (Coldewey and Krüge 1997), since the user interface determines the work flow, and any unforeseen use case is hard to handle, if it is manageable at all. New cases will usually require changes of the user interface. You therefore have to take care during analysis and design to make the form-based UI extensible, which is harder than with an object-oriented UI.

3 Problem area: Different architectures

The user interface aspects of an application should be separated from the business/domain logic and preferably pluggable and reusable in several applications (Coldewey and Krüge 1997; Eva 2002; Iverson 2004).

3.1 User Interface Layer Patter

Separate user interface issues from domain level aspects and put them into two different subsystems. Make sure the domain level subsystem has no notion of the user interface part, thus forming a layered architecture.

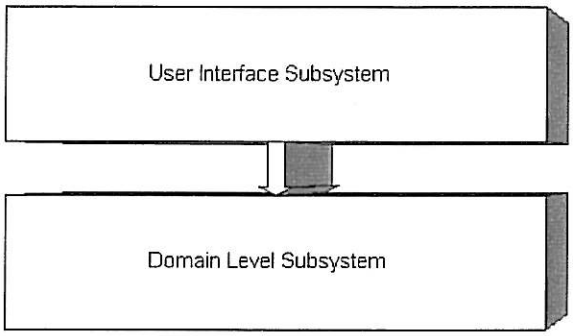
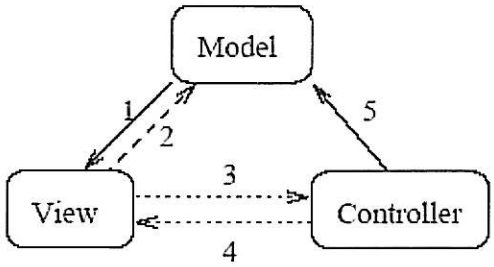


Figure 1: Layered architecture: Separate user interface layer and domain layer

3.2 MVC (Model-View-Control pattern)

The model-view-controller (MVC) pattern describes a modularization of functionality in interactive software applications in which the user interacts directly with a controller, that in turn interprets the user's actions to modify both the application's internal data model and the representation of that model on the computer screen (the view) (Iverson 2004).



1: The model sends a notification of change to the view
2: The view calls the methods of the model
3: The view calls the basic functions in the controller. Note that you should keep this dependency minimal in case you want to swap to another controller
4, 5: The controller adjusts the model according to user input, and contact the view to interpret what user action means

Figure 2: The basic links of communication between a model, a view and a controller (Eva 2002)

3.3 DKC (Data-Knowledge-Context model)

The Data-Knowledge-Context (DKC) model is a layered application development model and system architecture that separates concerns between a general operating system layer (the Data layer) and two semantically rich layers above it (the Knowledge and Context layers). It separates these concerns based on persistent vs. ephemeral storage and explicit vs. implicit semantics. Data layer is a generic data storage infrastructure. The Knowledge layer is a persistent store of explicitly semantic knowledge intended to provide the facilities for storing and accessing semantically rich depictions and interpretations of the world. The Context layer is the ephemeral, task and user-oriented interface to this world of data and knowledge (Iverson 2004).

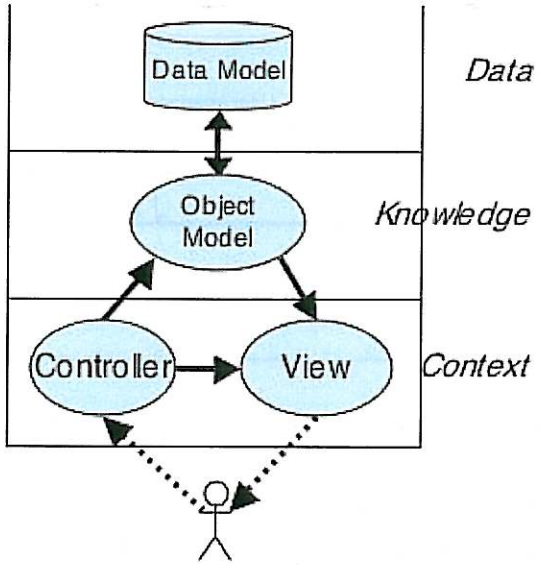


Figure 3: MVC mapping to DKC layered model

4 Problem area: Patterns for form-based mobile user interface

"Patterns are abstract, core solutions to problems that recur in different contexts but encounter the same 'forces' each time."(Graham 2003)

4.1 Pattern-based development

"Design patterns … are well known uses of successful patterns in computing. … patterns document problems and its correspondent best solutions. Pattern languages are excellent tools for expressing the concepts involved in a given domain. Therefore, they constitute a valuable way to express distilled experience from real life." (Molina, Meli´a et al. 2002)

4.2 Form-based UI pattern

(Coldewey and Krüge 1997) have developed a pattern language to support the development business information systems with a form-based UI. The language consists of three parts (Figure 4):

- *Architectural patterns* describe classes and subsystems of the architecture that do not depend on a specific platform.
- *Conceptual patterns* describe analysis concepts that help to organize the architecture.
- The *Platform Patterns* encapsulate different hardware and operating system aspects.

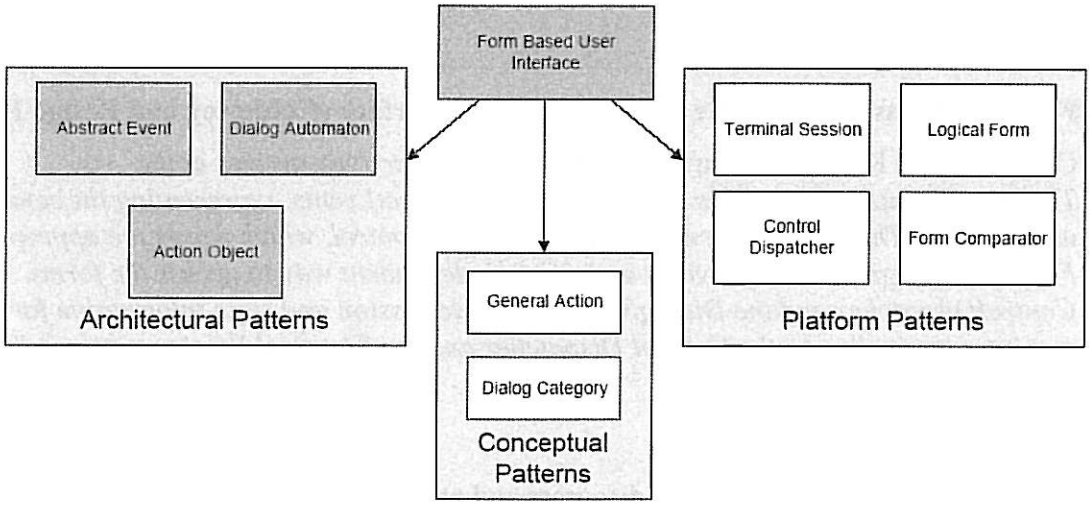


Figure 4: The patterns of the Form-Based User Interface language (Coldewey and Krüge 1997)

According to (Coldewey and Krüge 1997) a information system based on form-based UI should be based on a layered architecture that separates the user interface and the domain kernel (Figure 5). They partition the user interface into five subsystems. Control Dispatching and I/O Wrappers encapsulate the operating system. These two subsystems are specific to the platform in use. Event Abstraction, Dialog Control and Domain Kernel Interface represent the logical part of the user interface. They are independent of any specific implementation platform.

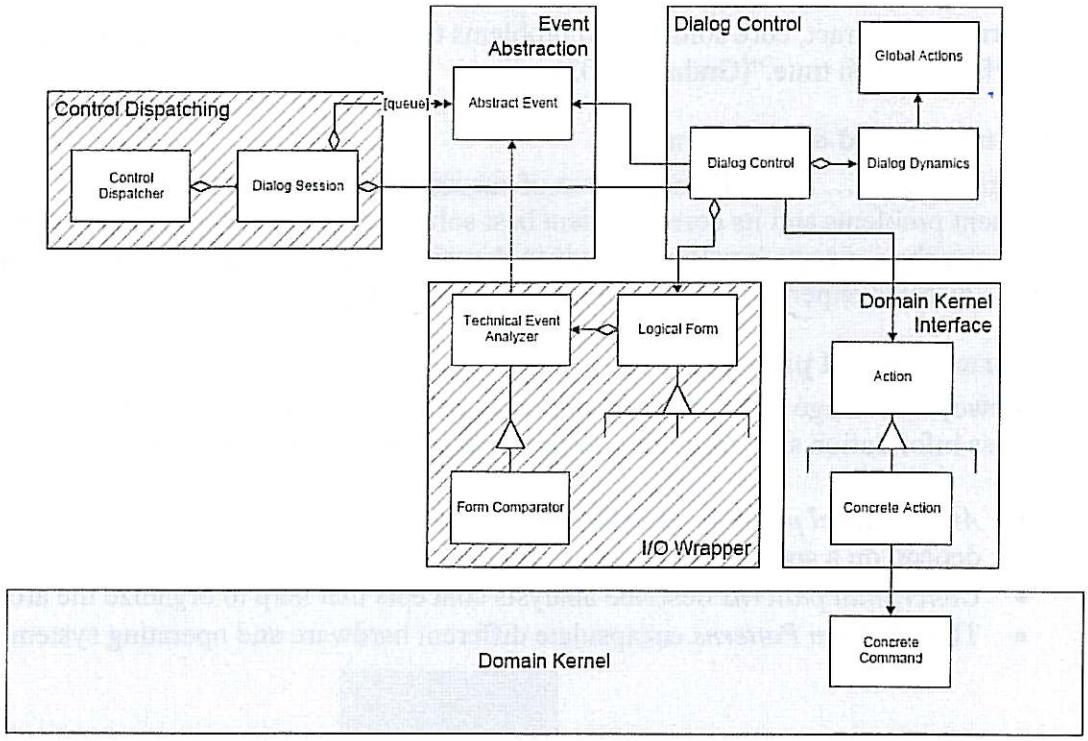


Figure 5: Class design of the form based user interface (Coldewey and Krüge 1997)

Coldeway and Krüger explain: *"Logically, any user activity arrives at the TechnicalEventAnalyzer. This class generates AbstractEvents, representing the semantic of the user activity. This event is forwarded to the DialogControl, which raises the appropriate Action. Finally the LogicalForm provides a platform independent way to access the forms. The ControlDispatcher and the DialogSession provide session and state information for transaction monitors, according to the Control Dispatcher and the Terminal Session patterns."* (Coldewey and Krüge 1997)

4.3 UI Pattern Collections

(van Welie and Trætteberg 2000) discusses and presents twenty interaction patterns in user interfaces. The patterns take an end-user perspective and usability is the essential design quality.

In the following subsection we present online collections of UI Patterns.

4.3.1 Pattern in Interaction Design (PID) [1]

This pattern cataloging tool is basically just a website. It publishes patterns in groups based on three application domains: Web Design patterns, GUI Design patterns, and Mobile UI Design patterns. All the patterns within this tool can be browsed and searched using basic internet tools. User can read the detail of each pattern and add their comments to the bottom of any pattern. There are links to related patterns, cited literature and other resources supporting a pattern using standard hyperlinks. There is no pattern submission facility for visitors to the site.

4.3.2 Common Ground (CG) [2]

This tool is composed of a number of web pages, which organize and categorize patterns using questions that address specific design problems, to assist users in choosing the patterns they wish

[1] Pattern in Interaction Design: http://www.welie.com/patterns/
[2] Common Ground: http://www.mit.edu/~jtidwell/interaction_patterns.html

to view. All the patterns in this tool have the same structure including a request from the author to send comments via email. Some additional notes have been appended to some patterns. This tool relies on internet browser facilities to search for specific patterns and standard hyperlinks are used to navigate to related patterns.

4.3.3 UI Patterns and Techniques (UIT)[3]

The author of these patterns implies that they are just a re-work of the patterns presented in the Common Ground tool. Although a different presentation template has been used, the main difference is in the way the collection of patterns has been grouped. The patterns are ordered based on common design problems such as: "Organizing the Content" and "Getting Around". Internet tools provide the browsing and searching facilities for the repository.

4.3.4 User Interface Design Patterns (UIDP)[4]

This tool is similar to the previous three in that it is a basic website. The distinguishing feature is that the set of UI design patterns reflect recurring design problems based on the user's goals.

4.3.5 Some online guidelines for mobile computing

- Palm OS Design Guidelines[5]
- Openwave GSM Guidelines[6]
- Openwave Top 10 Usability Guidelines for WAP Applications[7]
- Blackberry and RIM wireless handheld UI Developers Guide[8]
- Nokia Mobile User Interface Guidelines[9]
- W3C Mobile Web Best Practices 1.0 User Interface Guidelines[10]
- W3C Ubiquitous Web Applications[11]
- DotMobi Mobile Web Developer's Guide[12]

(Gong and Tarasewich 2004) discusses the characteristics and limitations of current mobile device interfaces, especially compared to the desktop environment. Based on existing interface guidelines they propose a set of practical design guidelines for mobile device interfaces.

4.4 Pattern Design Tools

In this section we present a collection of Pattern Design Tools.

4.4.1 GUIDE

GUIDE is a tool to help developers create more usable interfaces (Henninger 2001). It has a searchable repository of usability guidelines, patterns and cases, which are examples or "context-specific instances" of patterns or guidelines. A rule-base uses questions to lead the developers through a set of design decisions based on the projects characteristics. This process builds a navigation hierarchy to visualize the project's characteristics linked to recommended patterns and

---

[3] UI Patterns and Techniques (UIT): http://time-tripper.com/uipatterns/
[4] User Interface Design Patterns (UIDP): http://www.cs.helsinki.fi/u/salaakso/patterns/Continuous-Filter.html
[5] Palm OS Design Guidelines: http://www.access-company.com/developers/#979973
[6] Openwave GSM Guidelines: http://developer.openwave.com/dvl/support/index.htm#styleguides
[7] Openwave Top 10 Usability Guidelines for WAP Applications: http://developer.openwave.com/dvl/support/documentation/guides_and_references/wap_usability/
[8] Blackberry and RIM wireless handheld UI Developers Guide (PDF): http://www.blackberry.com/developers/na/c_plus/doc/sdk21/ui_engine_api_21.pdf
[9] Nokia Mobile User Interface Guidelines: http://www.forum.nokia.com/main/technical_services/usability/ui___ue_guidelines.html
[10] W3C Mobile Web Best Practices 1.0 User Interface Guidelines: http://www.w3.org/TR/mobile-bp/
[11] W3C Ubiquitous Web Applications: http://www.w3.org/2007/uwa/
[12] DotMobi Mobile Web Developer's Guide: http://pc.dev.mobi/?q=node/197

guidelines that may provide solutions. This is an interactive process and designers can submit recommendations to the system librarian to update the knowledge base thereby improving the design advice given to the next project. This tool only uses patterns rather than providing facilities for managing pattern collections.

4.4.2 Damask[13]

Damask is a pattern-based design tool for early-stage design and prototyping of multi-device user interfaces(Lin and Landay 2002; Lin 2005). UI designers can use it to create storyboard prototypes. It contains a catalog of patterns from which a designer selects patterns as they build their design. The designer then customizes the patterns to match the problem domain. Each design pattern includes many different examples illustrating how to apply the pattern. As well, each pattern provides a general solution for each device (PC, cell phone, PDA) supported. When a storyboard sketch for one interface has been completed the system can create an equivalent UI design for other devices. Damask automatically updates the examples section of a pattern when new designs are created. Designers can also create new patterns to add to the repository but the solution must be illustrated using Denim's sketching language.

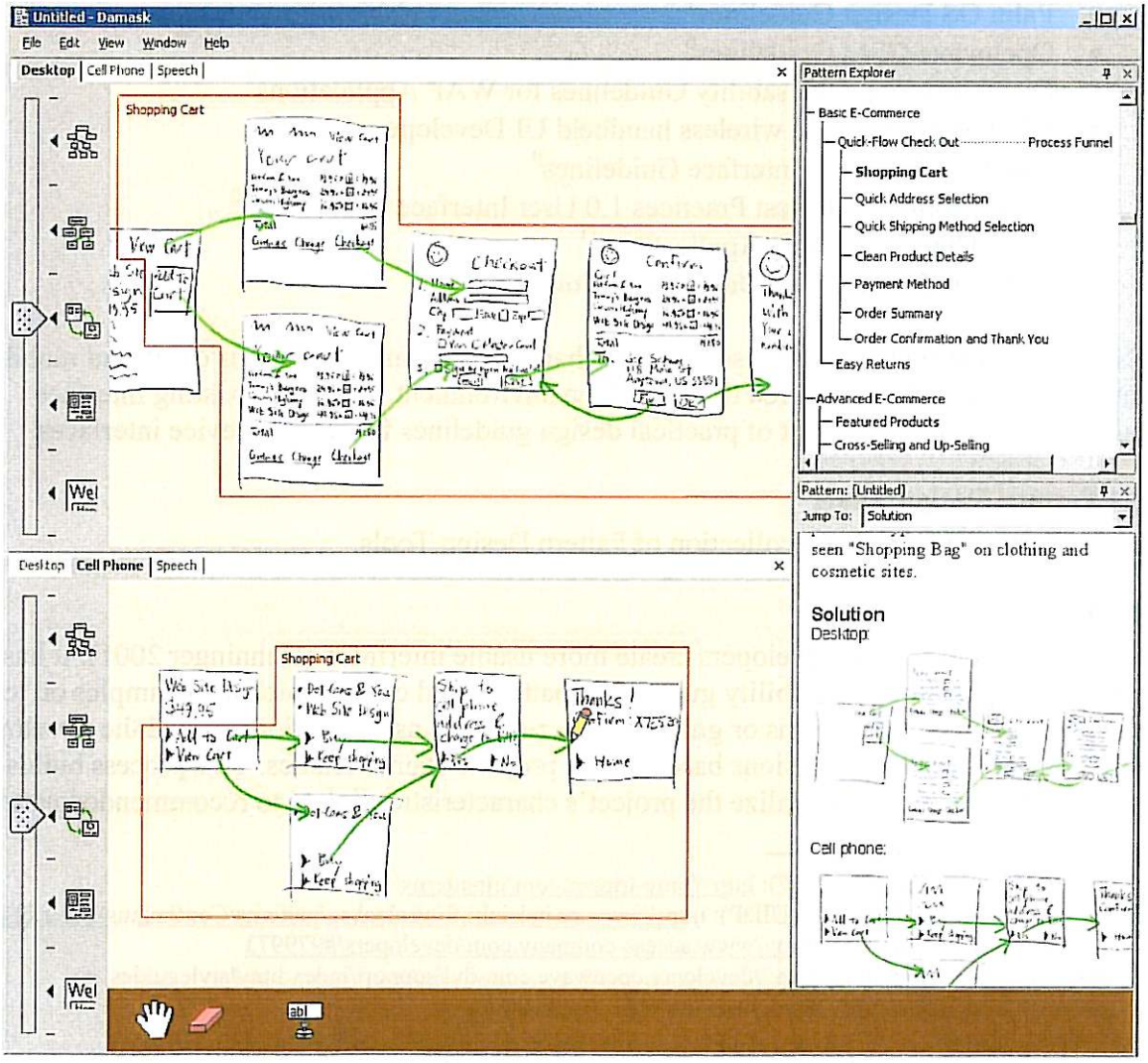


Figure 6: Damasku's user interface

[13] Damask: http://guir.berkeley.edu/projects/damask/

4.4.3 OlivaNova Modeler (ONM) [14]

ONM is a CASE tool which includes a pattern-based UI design component. A model of the UI can be developed based on their UI pattern language which includes patterns at different levels in the design process. The conceptual UI patterns have to be written using their pattern composition language which enables the automatic validation of potential designs. A transformation engine is used to translate the specification into implementation code in the selected programming language and for the specified environment, such as Web, Desktop or PDA.

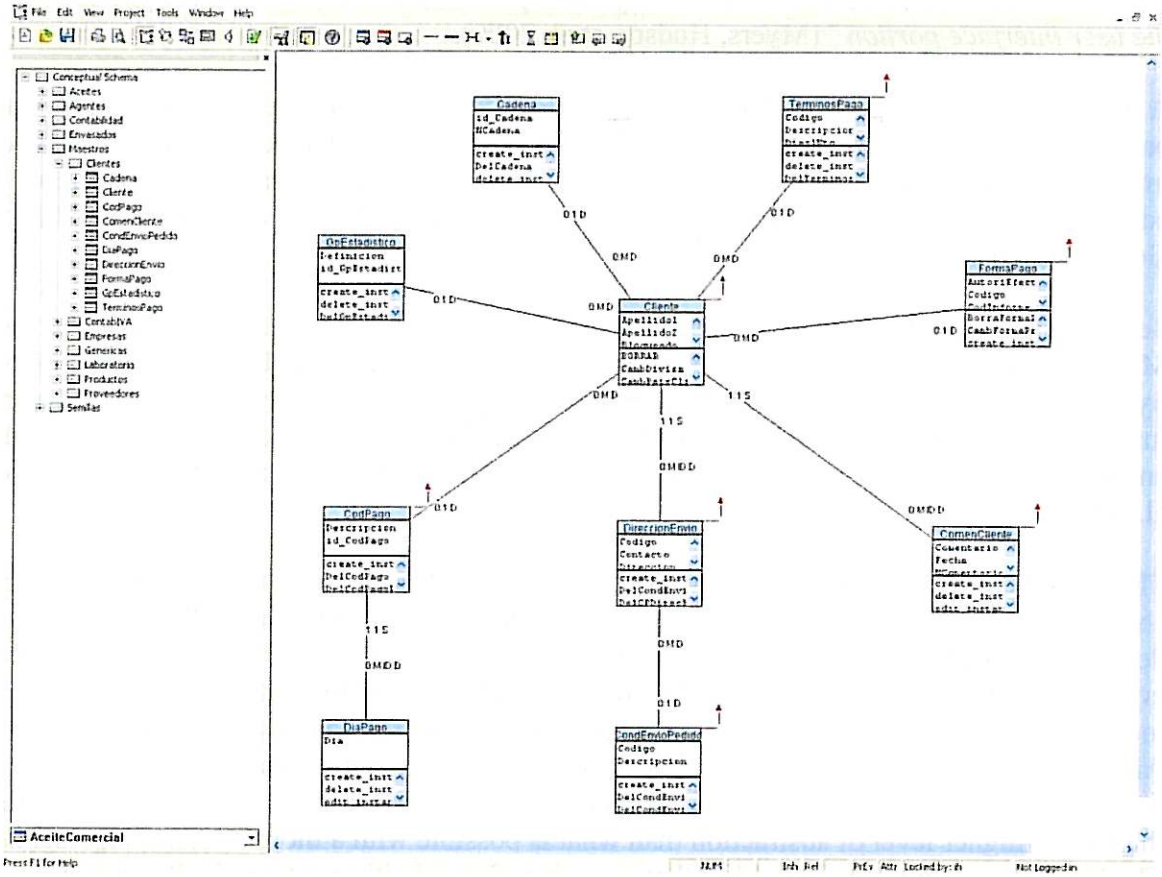


Figure 7: OlivaNova Modeler

[14] OlivaNova Modeler (ONM): http://www.sosyinc.com/products/modeler.asp

5 Problem arena: Model-based form-based mobile UI

User interface is a very important part in software development, and regarding multi-device user interface traditional code-based development is very expensive, both for the development of the first version of the user interfaces and for their maintenance (Florins 2006). Others have calculated the among of UI development: *"An average of 48% of the code of applications is devoted to user interface, and about 50% of the implementation time is devoted to implementing the user interface portion"* (Myers, Hudson et al. 2000).

A dominate approach within UI development is to automate or semi-automate the development, based on more or less abstract and descriptive descriptions. It is clamed that automating user interface improves the quality of developed interfaces and makes the creation of interfaces more economical and maintainable (Eisenstein, Vanderdonckt et al. 2001; Gajos and Weld 2004; Gomaa, Salah et al. 2005).

Several approaches have been proposed to automate the creation of user interfaces. User Interface Management Systems (UIMS) were first proposed with an analogy to Database Managements Systems (DBMS). Model-based user interface development environments (MBUIDEs) were introduced later to overcome problems faced with UIMSs. They are tools that support the design and development of user interfaces through the use of abstract interface models. (Gomaa, Salah et al. 2005).

The central component in a model-based tool is the model that is used to represent the UI in an abstract fashion. (Szekely, Sukaviriya et al. 1996) defines an interface model as high-level declarative specification of some single coherent aspect of a user interface, such as its appearance, layout characteristics and dynamic behavior. While (Puerta and Eisenstein 2002) defines an interface model as a computational representation of all the relevant aspects of a user interface. And different types of models have been used in different systems including task models, dialogue models, user models, domain models, and application models. All these models represent the UI at a higher level of abstraction than what is possible with a more concrete representation. The UI developer built these models that were transformed either automatically or semi-automatically to generate the final UI. The generation of the interface was often done via an automated tool using very complex processing. Most of these tools included a set of rules for generation of the interface components as well as for specifying interface style. Thus, extending a model-based development for multiple platforms would require adding new rules to the generation tool for each new interface style.

Within model-based UI research, there has been some interest in mobile user interfaces the last years (Nilsson, Floch et al. 2006). According to Nisson, Floch et al. the focus in most of this work is on using models at design time for specifying either purely mobile UIs or having models that act as specifications across mobile and stationary UIs, so called multiple UIs. Multiple-User Interface (MUI) refers to an interactive system that provides access to information and services using different computing platforms (Seffah, Forbrig et al. 2004).

Although model-based tools provide many advantages over other user interface development tools, there seems to be a consensus among the user interface experts that the model-based UI tools existing today have considerable shortages. The existing approaches of designing a single user interface using one platform do not adequately address the challenges of *diversity*, *cross-platform consistency*, *universal accessibility* and *integration* (Seffah, Forbrig et al. 2004). Nilsson identifies that model-based UI languages and tools today are *inefficient* when it comes to the among of platform specific models in a multiple UI setting, that they generate UI that *lack*

usability, and that they are fairly *restrictive* with regards to which type of user interfaces that may be expressed (Nilsson 2002). Although a common restriction is to form-based database application, we find few contributions regarding design-patterns and guidelines for form-based UI (see chapter 4). It is also claimed that the functionality in automatic generated solution is limited to generic functions like create, search/navigate, save, and delete.

Szekely (1996) identifies five approaches that model-based UI tools have taken: automatic interface design, specification-based model-based interface development environments, help generation, tools to help designers create models, and design critics and advisors.

5.1.1 Automatic interface design tools

Automatic interface design tools, like JANUS[15] and TRIDENT, strive to automatically create the user interface of an application, given a task or domain model of the application.

As Szekely describes (Szekely, 1996), an automatic design tool typically takes the following steps to generate a user interface:

1. Determine the presentation units. The tool figures out the windows that will be used and the contents of those windows.
2. Determine the navigation between presentation units. The tool constructs a graph of presentation units that defines which presentation units can be reached from other units.
3. For each presentation unit, determine the abstract interaction objects, which define the behavior for each element in a presentation unit in an abstract manner, for example, "select one from a list."
4. Map abstract interaction objects into concrete interaction objects, which are actual widgets available in a toolkit.
5. Determine the window layout, in other words, where the widgets are placed in the window.

The first three steps build the abstract UI specification, and the last two build the concrete interface.

As Szekely discusses, each of these steps is difficult to automate, especially steps 1 and 3, which require a deep understanding of the user's tasks. For example, it is hard for a tool to tell whether a set of data is better displayed as a table or as a graphical display like a map. Consequently, designers have not accepted the tools since it is harder to create a model and guess what the tool will generate than it is to design it directly themselves. Some tools, such as Tadeus (Task-based development of user interface) (Schlungbaum and Elwert 1996), explicitly involve the designer in each step instead of trying to do each step automatically, as a way to address this problem.

Bischop has identified eight challenges for software engineering (broader scope than just model-based engineering, but relevant) regarding multi-platform User Interface construction (Bishop 2006):

1. Indexing and navigating through the control library.
2. Encouraging standard and device independent positioning techniques for controls
3. Reducing the complexity of the event-handling paradigm
4. Extracting better error messages during reflective parsing
5. Establishing and developing core sets of widgets
6. Teaching principles of GUI programming in a notation-independent way.
7. Adapting GUIs to a variety of output devices with different operating systems and GUI libraries.

[15] JANUS Systems: http://www.janusys.com/

8. Making tangible user interfaces conform to high-level standards of portability from the start.

5.1.2 Specification-based model-based interface development environments

Unlike automatic interface design tools, specification-based model-based interface development environments (MBIDEs) do not try to automatically generate a user interface from task or domain models. Instead, designers directly create and interact with task, domain, or presentation models, which the MBIDE then uses to generate a final UI. Letting designers directly interact with models enables them to more easily specify a design, change it, and retarget it, and so on.

5.1.3 Types of models

Within the UI modeling research tradition, a number of model types are exploited. Some of the models are used as input to the UI modeling activities; others are the results of the UI modeling.

Most work on model based UI development utilizes at least one – preferably a number of – the interaction models to describe the appearance and behavior of the UI to be developed.

In spite of these differences, the architecture exploited by most languages and tools similar. Figure 8 shows how most tools work (Nilsson's modification of (Szekely 1996)).

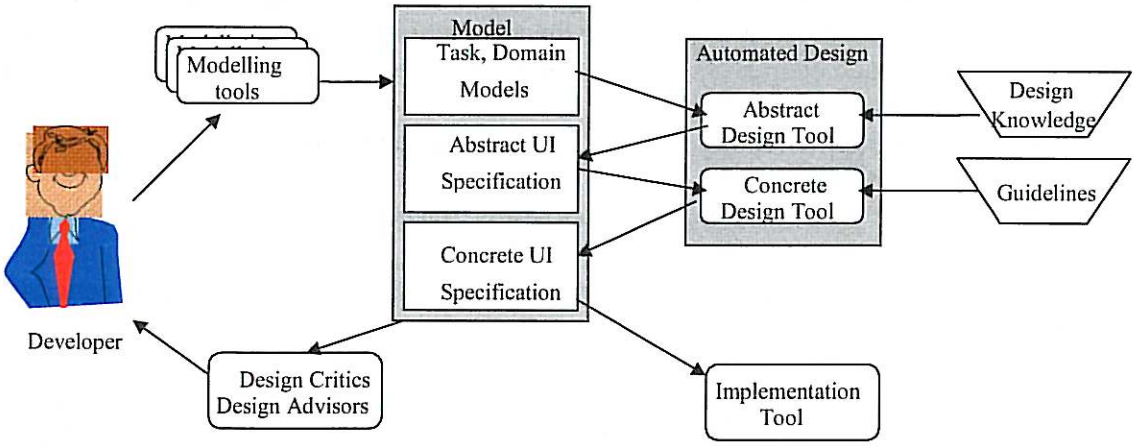


Figure 8: How most tools work (Nilsson, 2006 – Appendix B)

Nilsson says that is convenient to make a distinction between the models that are made for other purposes than describing a UI, and the models that primarily describe different issues regarding the UI. The domain and task models are usually in the first group, while the others are in the second.

(Vanderdonckt and Puerta 1999) lists nine different (kinds of) models that may be used in interface design, as shown in Figure 9.

Kind	Content	Representation
Task	what the user does or wants to do and why	There are two main kinds of task models: hierarchical: order/ sequence-oriented models and process/data flow models. The former stresses the constraints on the sequence of subtasks in a hierarchy, the latter how data is used in tasks for control and processing. Task models are used both for describing *current practice*, to judge potential for improvements, and for *envisioning* or *prescribing future* tasks for which a new user interface is to be designed.
Domain/data:	concepts, object and operations etc. describing the domain	Tasks use and operate within and on the domain. Main formalisms are entities with attributes and relations, and objects with attributes, relations and operations and aggregation hierarchies.
Dialogue/ conversation	the structure of dialogue between human and computer.	Behaviour-oriented abstractions like interactors, Petri Nets, flow charts, UML Activity and Sequence Diagrams, and state transition diagrams are used.
Concrete interaction	details of interaction, usually split into presentation and behaviour as follows:	
- Presentation	the structure of output and mapping to data and operations	Usually represented by generic but concrete dialogue elements and graphics
- Behaviour	means of inputting data and invoking operations	Describes how the dialogue is driven by the user's (physical) interaction and includes the link between presentation and dialogue, such as clicking a button to pop up a viewer.
Control	the application's services that task performance rely on	Usually a list of functions or object operations that may be invoked, and their pre- and post-conditions. Can be used as functional requirements for the application.
Platform	input and output capabilities of the device	Attributes and qualities may be defined, which can be referenced by interaction object mapping rules.
Environment	the physical and cultural context of interaction	Informal descriptions
User	characteristics and abilities of end-user not captured by other models	Classification hierarchy of user stereotypes, containing attributes and qualities that can be reference by e.g. mapping rules.

Figure 9: Kinds of user interface models (copied from (Trætteberg 2002))

The two mappings, from tasks to abstract dialogue to concrete dialogue, as well as evaluation, are the main steps in a model-based and iterative design process (Trætteberg 2002). In addition to task and dialogue models, these three steps rely on additional supporting models that capture knowledge of both the problem and solution domain.

Trætteberg explains the indicated steps in the table; the main steps from tasks and domain models, via abstract interaction or dialogue, to concrete interaction, are supported by models about the application core, target platform, working environment and user characteristics. The actual content and requirements of all these models depend on how they are used within the design process, and hence are somewhat ambiguous. For instance, a task model targeted at automatic generation of interface architecture will need to be more formal and complete than one focusing on mutual understanding among human participants in a design group. The focus of a method will

also implicitly depend on the tradition, engineering or cognitive, within which it was conceived. Although there may be consensus concerning the relevant models, the details of a method may vary considerably (Trætteberg 2002).

5.2 Some contributions

In this section we give some pointers to selected contributions within the field of model-based user interface. Many of the contributions are based on Szekely's modelling architecture (Szekely 1996), like the CAMELEON framework.

5.2.1 CAMELEON framework[16] and MDA approach

Models structured in four abstraction levels (CAMELEON framework)

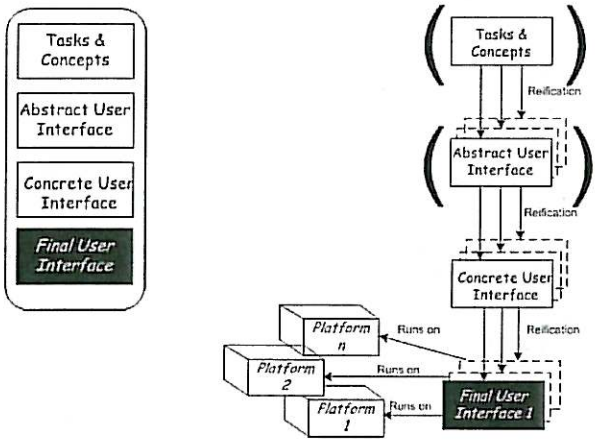


Figure 10: CAMELEON framework

Three design levels:

- *Task and Concepts* is described through domain models and task models.
- *Abstract User Interface* defines the interaction spaces, e.g. the distribution of task among windows and pages.
- *Concrete User Interface* describes the UI in terms of presentation objects such as windows, buttons, check boxes and images and their layout relationships.

TERESA (see section 5.5.5 at page 34) and UsiXML (see section 5.4.7 at page 29) are based on the CAMELEON framework, and (Vanderdonckt 2005) illustrates the use of CAMELEON framework with UsiXML corresponds to a model-driven architecture approach (MDA[17]).

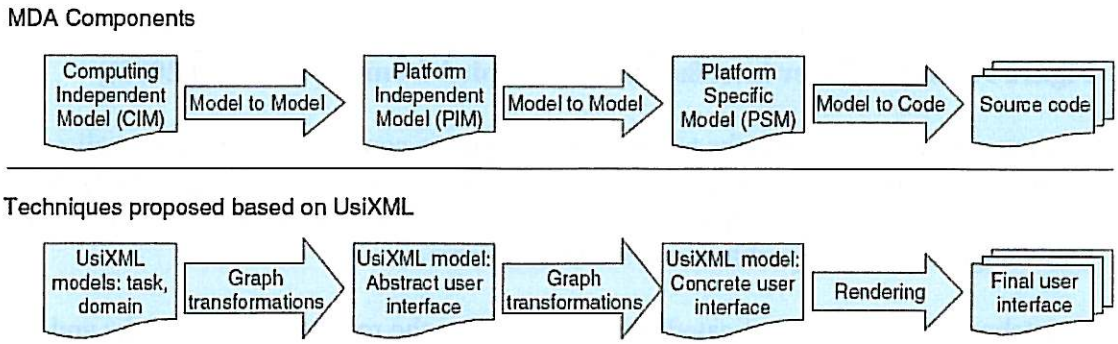


Figure 11: The distribution of UsiXML models according to the MDA classification (Vanderdonckt 2005).

[16] CAMELEON project: http://giove.cnuce.cnr.it/cameleon.html
[17] MDA: http://www.omg.org/mda/

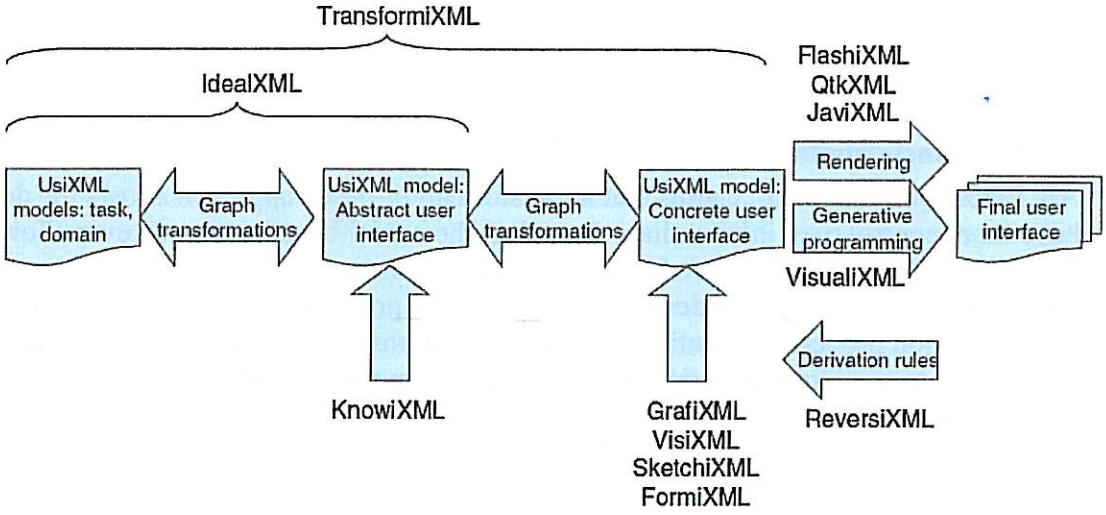


Figure 12: The suite of UsiXML tools structured according to the MDA classification (Vanderdonckt 2005).

- TransformiXML[18] is a Java-based application that is responsible for defining, storing, manipulating, and executing productions contained in graph grammars to support graph transformations (model-to-model transformations)
- IdealXML[19] is a Java-based application containing the graphical editor for the task model, the domain model, and the abstract model. It can also establish any mapping between these models either manually (by direct manipulation) or semi-automatically (by calling TransformiXML).
- KnowiXML[20] consists of an expert system based on Protégé that automatically produces several abstract UIs from a task and a domain models for various contexts.
- GrafiXML[21] is the most elaborate UsiXML high-fidelity editor with editing of the concrete UI, the context model and the relationships between. It is able to automatically generate UI code in HTML, XHTML, XUL and Java thanks to a series of plug-ins.
- VisiXML[22] is a Microsoft Visio plug-in for drawing in mid-fidelity graphical UIs, that is UIs consisting exclusively of graphical CIOs. It then exports the UI definition in UsiXML at the CUI level to be edited by GrafiXML or another editor.
- SketchiXML[23] consists of a Java low-fidelity tool for sketching a UI for multiple users, multiple platforms (e.g., a Web browser, a PDA), and multiple contexts of use (Coyette and Vanderdonckt 2005). It is implemented on top of Jack agent system.
- FormiXML is a Java editor dedicated to interactive forms with a smart system of copy/paste techniques to support reusability of components. It automatically generates the complete UI code for Java/Swing.
- Several renderers are currently being implemented: FlashiXML[24] opens a CUI UsiXML file and renders it in Flash, QtkiXML[25] in the Tcl/Tk environment, and JaviXML for Java.

[18] TransformiXML: http://www.usixml.org/v2/index.php5?module=menu&page=34
[19] IdealXML: http://www.usixml.org/v2/index.php5?module=menu&page=15
[20] KnowiXML: http://www.usixml.org/v2/index.php5?module=menu&page=19
[21] GrafiXML: http://www.usixml.org/v2/index.php5?module=menu&page=12
[22] VisiXML: http://www.usixml.org/v2/index.php5?module=menu&page=13
[23] SketchiXML: http://www.usixml.org/v2/index.php5?module=menu&page=14
[24] FlashiXML: http://www.usixml.org/v2/index.php5?module=menu&page=24
[25] QtkiXML: http://www.usixml.org/v2/index.php5?module=menu&page=25

- VisualiXML personalizes a UI and produces one thanks to generative programming techniques for Visual C++ V6.0.
- ReversiXML[26] opens a HTML file and reverse engineers it into UsiXML at both the CUI and AUI levels.

5.2.2 Transformation-based approach

Ali, Pérez-Quiñones et al. claims that a transformation-based approach allows the developer to have more control over intermediate steps than the model-based approach ever allowed. Furthermore they say their transformation approach encompass simpler processing, thus making it easier for the developer to understand the generation process, and potentially even extend the transformation process by creating new transformations. (Ali, Pérez-Quiñones et al. 2002). The approach is to transform generic UIML to platform-specific UIML. The platform specific UIML is then rendered using an existing UIML renderer.

5.2.3 Context-sensitive user interface and runtime models

(Clerckx, Luyten et al. 2004) presents a technique that allows adaptive user interfaces, spanning multiple devices, to be rendered from the task specification at runtime taking into account the context of use.

The designer can specify a task model using the ConcurTaskTrees Notation and its context dependent parts, and deploy the user interface immediately from the specification. The context will be resolved by the runtime environment and does not require any manual intervention. This way the same task specification can be deployed for several different contexts of use.

Traditionally, a context-sensitive task specification only took into account a variable single deployment device. They extends this approach as it takes into account task specifications that can be executed by multiple co-operating devices (Clerckx, Luyten et al. 2004). Figure 13 illustrates the design process.

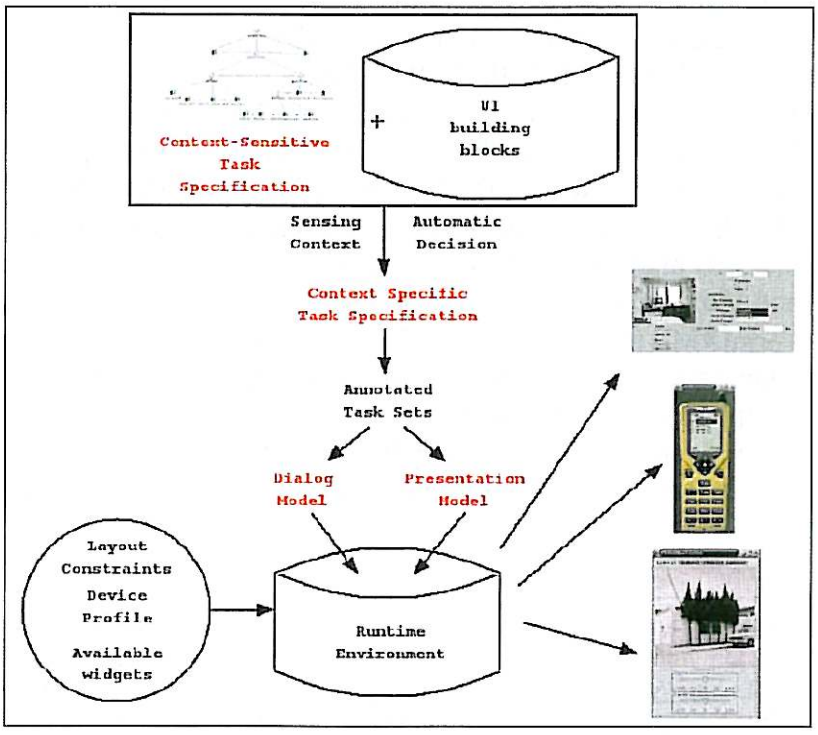


Figure 13: Context-sensitive user interface design process (Clerckx, Luyten et al. 2004).

[26] ReversiXML: http://www.usixml.org/v2/index.php5?module=menu&page=32

(Pribeanu, Limbourg et al. 2001) elaborates how to model context-sensitive tasks with the ConcurTaskTree notion (see section 5.4.9 at page 30). Figure 14 shows the overall approach (steps) with corresponding modeling languages for designing context sensitive user interfaces.

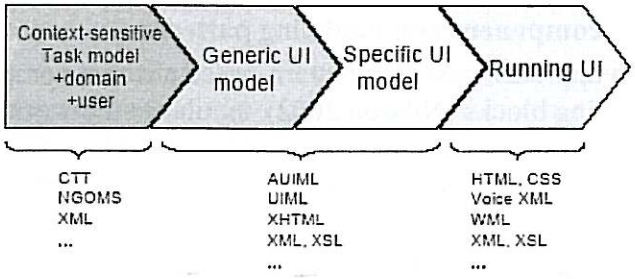


Figure 14: Steps of the method for designing context-sensitive user interfaces (Pribeanu, Limbourg et al. 2001)

In the FAMOUS project they developed support for handling adaptive applications in the context of mobile computing (Nilsson, Floch et al. 2006). They use a more architecture centric approach. That is exploiting architecture models to reason about and control adaptation at run time, and they used generic middleware components that realize the adaptation mechanisms such as reasoning about adaptation. The middleware has three main functions:

1. Detect context changes.
2. Reason about the changes and make decisions about what adaptation to perform.
3. Perform the chosen adaptation.

Figure 15 illustrates the middleware architecture used in FAMOUS (the numbers refer to the three functions just mentioned).

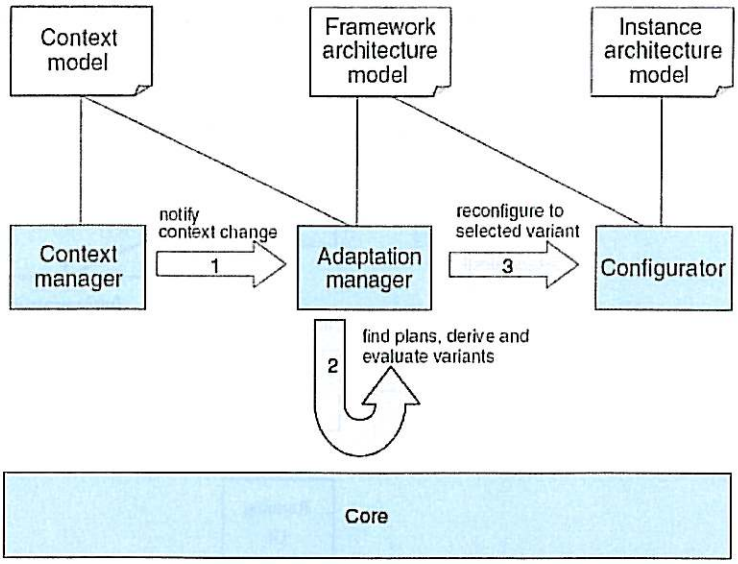


Figure 15: Middleware architecture and the run-time models (Nilsson, Floch et al. 2006)

5.2.4 Tasks, dialogs and concrete interactions

Trætteberg suggest three main user interface design perspectives, the task/domain perspective, the abstract dialogue perspective and the concrete interaction perspective (Trætteberg 2002). The former was described as a problem-oriented perspective, while the latter two where described as solution-oriented. Dialogue modeling is concerned with dataflow, activation and sequencing, while concrete interaction focuses on the dialogue's relation to input and output devices, e.g. windows, widgets and mice. Although he distinguish between these two perspectives, they are partly integrated in one modeling language.

Trætteberg has developed the modeling languages: taskMODL and diaMODL (Trætteberg 2002).

5.2.5 Compound component and modeling patterns

Nilsson presents an approach based on pattern-based abstract compound user interface components as building blocks (Nilsson 2002), similar to the work by (Trætteberg 2002). (Nilsson, Floch et al. 2006) argues that most model-based languages and tools suffer from a combination of two connected characteristics: the languages offer concepts on a too low level of abstraction, and the building blocks are too simple. (Nilsson 2002) uses a combination of compound components and modeling patterns:

- Compound (or composite) user interface components are used to be able to have equal or similar model instances on platforms with significant differences (including traditional GUI, Web user interfaces and user interfaces on mobile equipment).
- Modeling patterns are used partly to obtain the necessary level of abstraction to facilitate common models across different platforms, and partly to render it possible to define generic mapping (or transformation) rules from the patterns-based, abstract compound components to concrete user interfaces on different platforms.
- A mapping rule is a generic and operational description of how a modeling pattern instance should be transformed to a running UI on a given platform, and mapping rules are an important part of the modeling framework.

As a modeling pattern usually involves a number of objects, a user interface supporting a modeling pattern must be a composition of different user interface components (each being simple or composite). The transformation rules describe how the modeling patterns are to be realized on various platforms. This means that the transformation rules must be instantiated with the same concrete classes that the patterns are instantiated with (Nilsson, Floch et al. 2006).

Figure 16 shows how the different main parts of the modeling approach are connected.

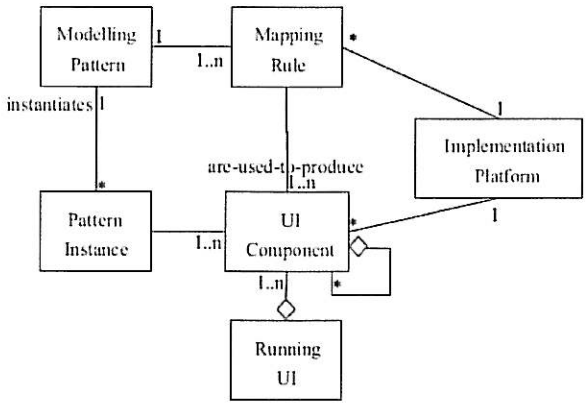


Figure 16: Main concepts in the modelling approach (Nilsson, Floch et al. 2006)

Figure 17 shows the Composite pattern and the instantiation of it.

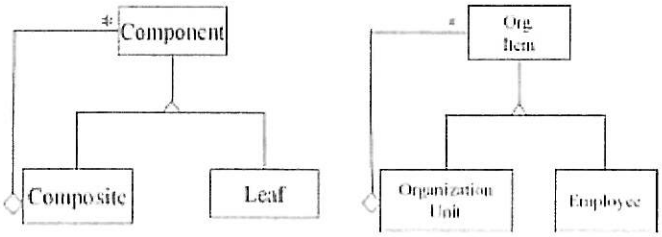


Figure 17: The Composite Pattern (left) and the Department structure/Human resource mgmt. instantiation of it (right) (Nilsson, Floch et al. 2006)

Figure 18 shows two similar mapping rules for PDA platform.

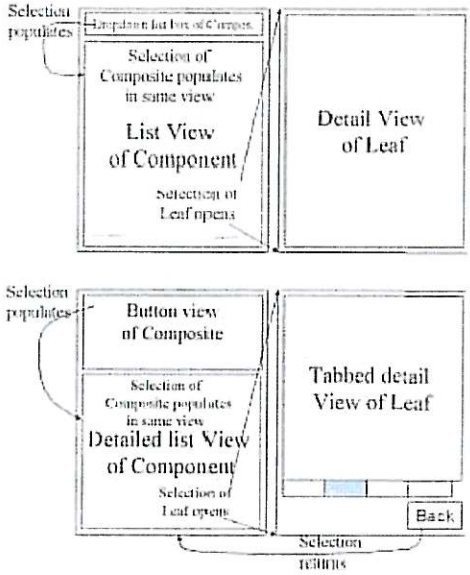


Figure 18: Stylus (top) and finger (bottom) usage mapping schemes for PDA presentation (Nilsson, Floch et al. 2006)

5.2.6 IRTV: Information, Role, Task and View

In the MAPPER project[27] AKM[28] has proposed a modeling approach for model generated workplaces and executable models. The IRTV approach is based on and integrated with the POPS enterprise modeling language approach. POPS stands for the four enterprise dimensions products, organizations, processes and systems. Currently IRTV models are modeled in Metis[29] and workplaces are generated into operational Metis modeling views.

Model-based UI and executable models were also a subject in the ATHENA project[30]. There we describe and partly implemented the language UIM (User Interface Modeling) in Metis executable models environment in order to manage web-based workplaces. The workplaces were used to manage living enterprise models (product models, organization models, process models and system models, i.e. POPS) (Rolfsen, Boell et al. 2007). Figure 19 is an example of a workplace model used in the ATHENA project.

[27] MAPPER project: http://193.71.42.92/websolution/UI/Troux/07/default.asp?webid=260
[28] AKM: http://www.activeknowledgemodeling.com/
[29] Metis: http://www.troux.com/
[30] ATHENA project: http://www.athena-ip.org/

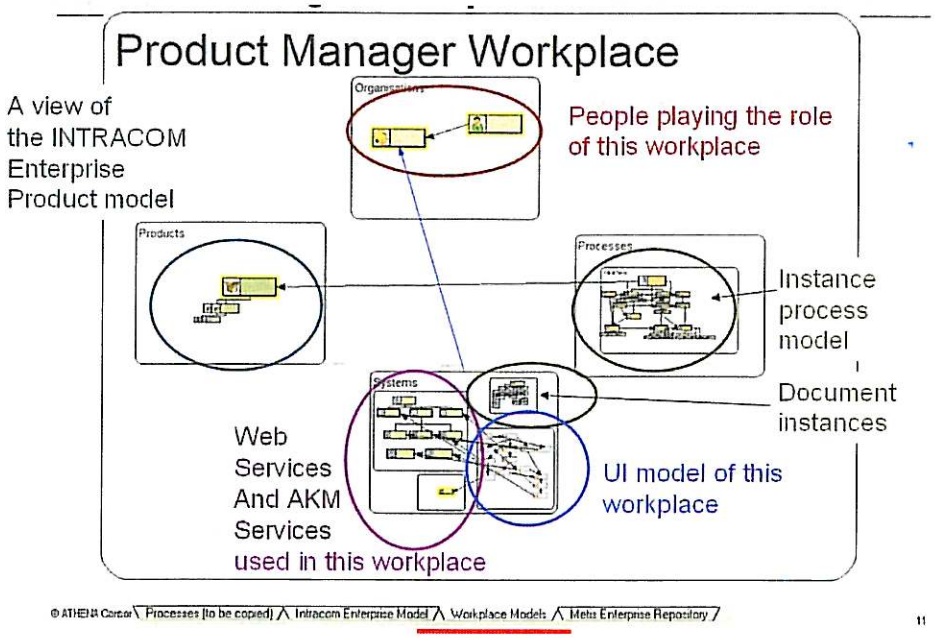


Figure 19: An example of a workplace model

5.2.7 Sketching approaches

"Designing the right User Interface (UI) the first time is very unlikely to occur. Instead, UI design is recognized as a process that is intrinsically open (new considerations may appear at any time), iterative (several cycles are needed to reach an acceptable result), and incomplete (not all required considerations are available at design time). Consequently, means to support early UI design has been extensively researched to identify appropriate techniques such as paper sketching, prototypes, mock-ups, diagrams, etc." (Coyette and Vanderdonckt 2005).

SketchiXML is a multi-platform multi-agent interactive application that enables designers and end users to sketch user interfaces with different levels of details. It provides the designer with on demand design critique and assistance during early design. SketchiXML do not produce code specific to a particular case or environment, but it generates UI specifications written in UsiXML (see section 5.4.7 at 29), a platform-independent User Interface Description Language (UIDL) that will be in turn exploited to produce code for one or several UIs, and for one or many contexts of use simultaneously (Coyette and Vanderdonckt 2005).

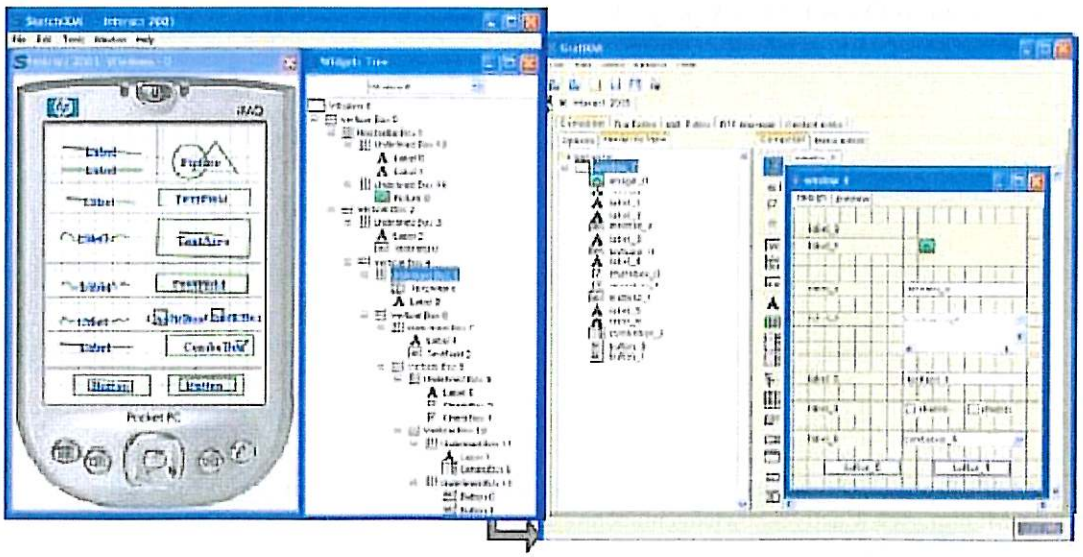


Figure 20: SketchiXML workspace configured for a PDA and its import in GrafiXML (Coyette and Vanderdonckt 2005).

Damask is a sketching and design pattern tool for designing and generating multi-device user interfaces (Lin and Landay 2002; Lin 2005).

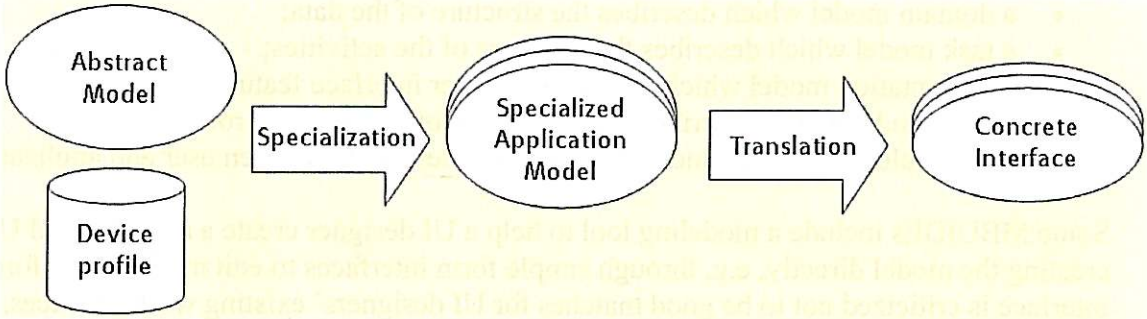


Figure 21: Traditional transformation flows in model-based user interface tools

They claim that designers think in terms of concrete widgets, but with a traditional model-based user interface tools they are forced to start with abstract widgets.

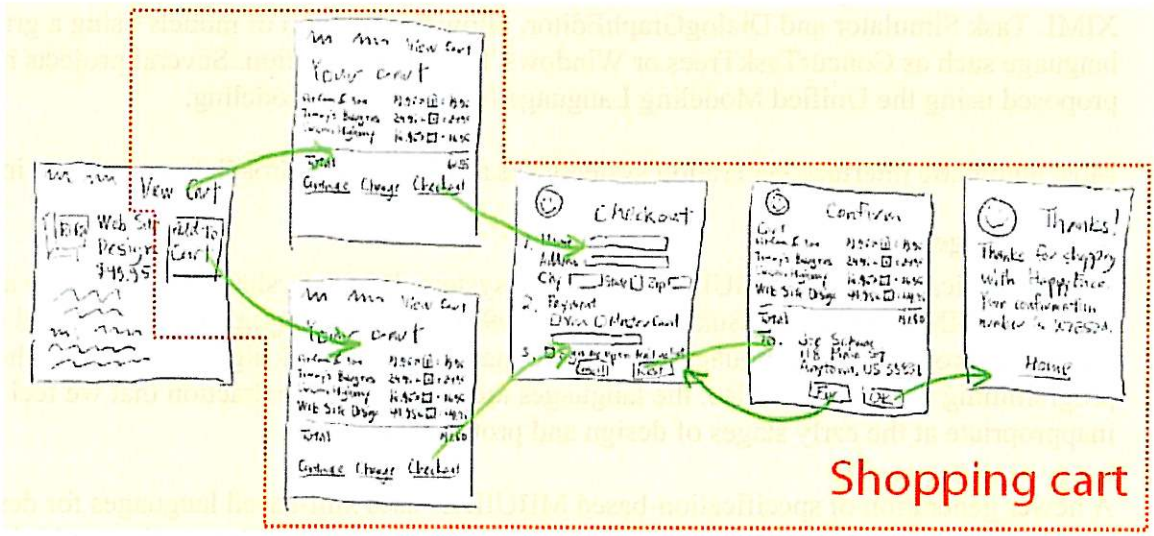


Figure 22: Damask pattern and sketching approach

5.3 Modeling tools: MBUIDEs

Model-based user interface development environments (MBUIDEs) are tools that help designers with building interfaces through automating the generation of interfaces using high-level declarative models. (Gomaa, Salah et al. 2005)

There are two generations of MBUIDEs that appeared as improvements to the previous UIMSs. The first generation was basically aiming at providing a strategy to generate a user interface from the high-level models. The tools of this generation emphasized the automatic generation of an interface instead of a user interface design process. A central limitation of these tools was the lack of user control over the process of UI generation.

In the second generation of MBUIDEs the involvement of users in the development process of interfaces are stressed. The MBUIDEs are user-centered. The interface model was described in better ways, and tools of this generation supported the incremental interface design. (Gomaa, Salah et al. 2005)

In a model-based UI approach, the developer first constructing declarative specifications or models of the various aspects of the user interface, from which the application and user interface are produced. A MBUIDE permits the description of the following aspects (Cooper, McKirdy et al. 2000):

- a domain model which describes the structure of the data;
- a task model which describes the structure of the activities;
- a presentation model which describes the user interface features;
- a user model which describes characteristics of various user roles;
- and a dialogue model which describes the interaction between user and application.

Some MBUIDEs include a modeling tool to help a UI designer create a model-based UI without creating the model directly, e.g. through simple form interfaces to edit models. The form-based interface is criticized not to be good matches for UI designers' existing work practices, which involve freeform sketching of UI designs (Lin 2005).

Other tools use a programming-by-demonstration interface builder as a front-end to creating a model. The model itself is exposed to the designer through a special-purpose modeling language. More recent modeling tools, such as Teresa, Planes, Windows Transition Notation UIDA, and XIML Task Simulator and DialogGraphEditor, allow the creation of models using a graphical language such as ConcurTaskTrees or Windows Transition Notation. Several projects have proposed using the Unified Modeling Language (UML) for UI modeling.

Most automatic interface generation systems use a rule-based approach to create user interfaces.

5.4 Languages

Early specification-based MBUIDEs, like Fruit system, BOSS-System, HUMANOID and MASTERMIND[31] (Szekely, Sukaviriya et al. 1996), use modeling languages that tend to look like traditional programming languages, which are inappropriate for designers, who often have little programming background. Also, the languages are at a level of abstraction that we feel is inappropriate at the early stages of design and prototyping.

A newer generation of specification-based MBUIDEs uses xml-based languages for describing user interface models[32]. Many of these languages were designed with targeting multiple types of devices in mind.

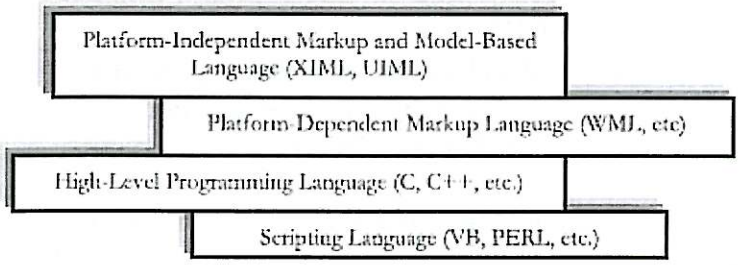


Figure 23: Evolution of UI Development Languages (Seffah, Forbrig et al. 2004).

[31] MASTERMIND: http://www.gvu.gatech.edu/gvu/ui/Mastermind/
[32] XML Markup Languages for User Interface Definition: http://xml.coverpages.org/userInterfaceXML.html

5.4.1 UML[33]

UML (Unified Modeling Language) is a family of languages based on a unified set of concepts described in a meta-model. Several diagram types are defined with a concrete syntax or notation for the abstract (meta) concepts. UML includes a light-weight *extension* mechanism, as an alternative to augmenting the meta-model. Sets of related extensions can be used for defining *profiles* for specific domains, such as interactive systems. UML provides diagrams for static domain modelling (class and object diagrams), functional requirements (use case, sequence and activity diagrams), behaviour (sequence, collaboration, state and activity diagrams) and deployment (component diagram). These could in principle be made to cover the nine subdomains from (Vanderdonckt and Puerta 1999), but it is an open question how easily and well UML cover them (Trætteberg 2002).

5.4.2 AUIML[34]

AUIML is an XML dialect that is a platform- and a technology-neutral representation of panels, wizards, property sheets, etc. AUIML captures relative positioning information of user interface components and delegates their display to a platform-specific renderer. Depending on the platform or device being used, the renderer decides the best way to present the user interface to the user and receive user input.

The AUIML XML is created using the Eclipse-based AUIML VisualBuilder[35], which allows a developer to quickly build and preview user interfaces in the Java Swing and Web renderers. The AUIML VisualBuilder can also automatically create application launchers, data beans, event handlers, and help system skeletons for the user interface. Because it plugs into Eclipse, building the user interface and application code is an integrated process.

5.4.3 UIML[36]

User Interface Markup Language (UIML) was originally a research project at Virginia Tech, now being developed commercially by Harmonia.

UIML is an appliance-independent XML meta-language for describing interfaces. Because it is based on XML, it is easy to write transformations that take the language from one abstract representation to a more concrete representation. It takes the approach of building applications using a generic vocabulary that could then be rendered for multiple platforms. A special renderer for each target device is needed. It could be argued that having multiple renderers is a disadvantage. UIML does not take into account any of the advances of MBUIDE reached in the past decade because it does not allow the abstraction of interaction functionality (Gomaa, Salah et al. 2005) for example, the language provides no notion of task, it mainly aims to define an abstract structure (Mori, Paternò et al. 2003).

---

[33] UML: http://www.uml.org/
[34] AUIML Toolkit from IBM: http://www.alphaworks.ibm.com/tech/auiml/
[35] Abstract User Interface Markup Language Toolkit: http://www.alphaworks.ibm.com/tech/auiml
[36] User Interface Markup Language (UIML): http://www.uiml.org/

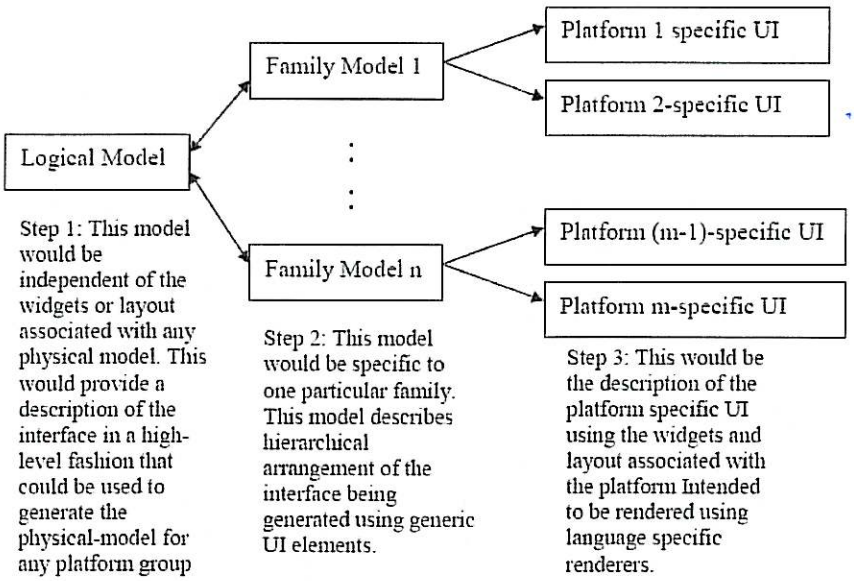


Figure 24: Framework for building multiplatform User Interfaces using UIML

5.4.4 XIML[37]

XIML is a XML-based language, whose initial development took place at the research laboratories of RedWhale Software[38]. It is intended to be a universal user interface specification language, since it provides a way to completely describe a user interface and represent attributes and relations of the important elements of a user interface without worrying about how they will be implemented.

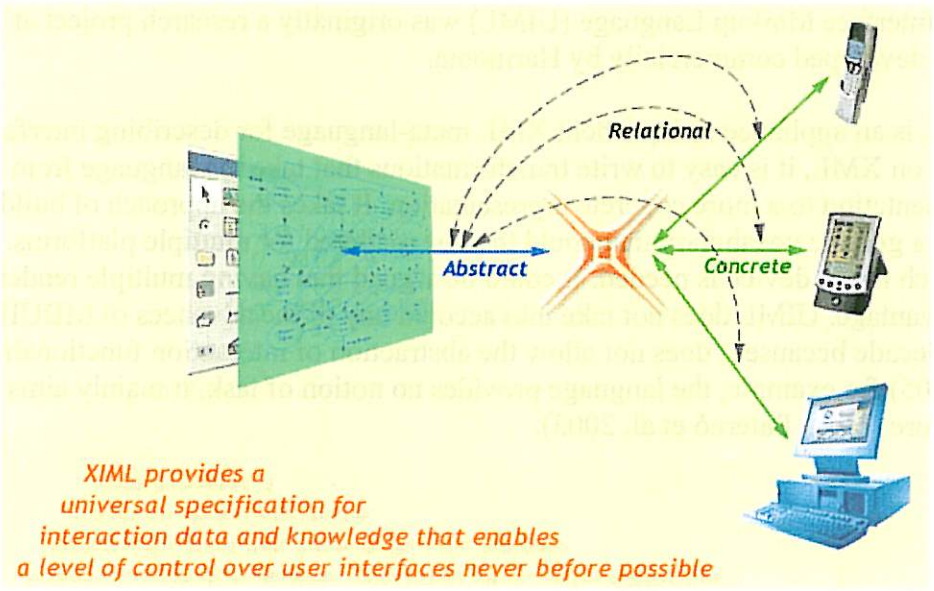


Figure 25: XIML: a universal language for user interfaces (http://www.ximl.org/)

In other words, it enables a framework for the definition and interrelation of interaction data items, thereby providing a standard mechanism for applications and tools to interchange interaction data and interoperate within integrated user-interface engineering processes, from design, to operation, to evaluation. "Today XIML is probably the most advanced UI specification language, as it can serve for contextsensitivity and many other objectives. However, it is worth

[37] eXtensible Interface Markup Language (XIML): http://www.ximl.org
[38] RedWhale Software: http://www.redwhale.com/

noting that XIML mainly focuses on syntactic, rather than semantic aspects. In addition, tool support is not publicly available." (Mori, Paternò et al. 2003)

XIML is based on Mobi-D work (see section 5.5.4 at page 33).

5.4.5 XUL[39]

XML User Interface Language (XUL) is a markup language used for describing user interfaces. It has its focus on window-based UIs by XML. It has a rich notion for creating widgets, and uses box, grid and other layout models (Nichols, Chau et al. 2007). Its disadvantage is the limitation to such graphical user interfaces, which makes it unsuitable to interfaces of small mobile devices. (Gomaa, Salah et al. 2005). It is a part of the Mozilla[40] browser and related applications and is available as part of Mozilla Layout Engine[41]. It is designed to be portable and is available on all versions of Windows, Macintosh as well as Linux and other Unix flavors. With XUL and other Gecko components, you can create sophisticated applications without special tools[42].

5.4.6 XAML[43]

XAML is the declarative markup language Microsoft has made available with Version 2 of the .Net framework, as part of the Windows presentation Foundation (WPD) component of the Vista platform for Windows. XAML rides on the language interoperability of .Net (Nichols, Chau et al. 2007).

Aurora XAML Designer[44] is an XAML authoring and design application that assists web and software developers in the creation .NET 3.0 software products and Web Applications (XBAP) for Windows XP and Windows Vista. The XAML markup generated by Aurora is also suitable for the authoring WPF/E web content that runs cross platform in a variety of browsers.

5.4.7 USIXML[45]

USer Interface eXtensible Markup Language (USIXML) is aimed at describing UIs with various levels of details and abstractions, depending on the context of use. USIXML supports a family of UIs including and not limited to device independent, platform independent, modality independent and context independent. Context sensitive UIs may be specified and produced from the USIML specifications. USIML supports multiple models for UI design such as task, domain, presentation, dialog, and context of use. Context of use in turn is decomposed into user, platform and environment. These models are structured according to 4 layers of CAMELEON framework: task and concepts, abstract UI, concrete UI and final UI (Limbourg, Vanderdonckt et al. 2004)

5.4.8 XForms[46]

The W3C consortium has delivered the XForms standard that presents a description of the architecture, concepts, processing model, and terminology underlying the next generation of web forms based on the separation between the purpose and the presentation of a form. (Gomaa, Salah et al. 2005). Traditional HTML Web forms do not separate the *purpose* from the *presentation* of a form. XForms, in contrast, are comprised of separate sections that describe what the form does, and how the form looks. This allows for flexible presentation options, including classic XHTML forms, to be attached to an XML form definition.

---

[39] XML User Interface Language (XUL): http://www.mozilla.org/projects/xul/
[40] Mozilla: http://www.mozilla.org/
[41] Mozilla Layout Engine: http://www.mozilla.org/newlayout/
[42] XULPlanet: http://www.xulplanet.com/
[43] XAML: http://www.xaml.net/
[44] Aurora XAML Designer: http://www.mobiform.com/products/Aurora/aurora.htm
[45] USIXML: http://www.usixml.org
[46] XForms: http://www.w3.org/MarkUp/Forms/

Some implementations:

- AjaxForms[47]: Transforms XForms documents into HTML+Javascript.
- DataMovil[48]: A platform for the development of PDA (Windows Mobile) applications. It is based on XForms 1.0. It is composed of a client, a server and an edition tool. The client implements around 80% of the standard and a presentation engine completely separated from the XForms part of the documents, providing for the first time of a clean data-logic-presentation separation. The editor allows for graphically developing and previewing applications. The server controls application versions, user authentication and other functions.

5.4.9 Task modelling

We select one model domain language that is frequently mentioned and used by other MBUIDEs, that is the task modeling notation ConcurTaskTrees[49].

ConcurTaskTrees has been developed by Fabio Paternò[50]. Its main purpose is to be an easy-to-use notation that can support the design of real industrial applications, which usually means applications with medium-large dimensions.

The main features of ConcurTaskTrees are:

- Hierarchical structure, a hierarchical structure is something very intuitive, in fact often when people have to solve a problem tend often to decompose it into smaller problems still maintaining the relationships among the smaller parts of the solution; the hierarchical structure of this specification has two advantages: it provides a large range of granularity allowing large and small task structures to be reused, it enables reusable task structures to be defined at both a low and a high semantic level.
- Graphical syntax, a graphical syntax often (not always) is more easy to interpret, in this case it should reflect the logical structure so it should have a tree-like form;
- Concurrent notation, operators for temporal ordering are used to link subtasks at the same abstraction level. This sort of aspect is usually implicit, expressed informally in the outputs of task analysis. Making the analyst use these operators is a substantial change to normal practice. The reason for this innovation is that after an informal task analysis we want designers to express clearly the logical temporal relationships. This is because such ordering should be taken into account in the user interface implementation to allow the user to perform at any time the tasks that should be active from a semantic point of view.
- Focus on activities, thus it allows designers to concentrate on the most relevant aspects when designing interactive applications that that encompass both user and system-related aspects avoiding low levels implementation details that at the design stage would only obscure the decisions to take.

---

[47] AjaxForms: http://ajaxforms.sourceforge.net/index.html
[48] DataMovil: http://www.datamovil.info/
[49] ConcurTaskTrees: http://giove.cnuce.cnr.it/concurtasktrees.html
[50] Fabio Paternò's homepage: http://giove.cnuce.cnr.it/~fabio/

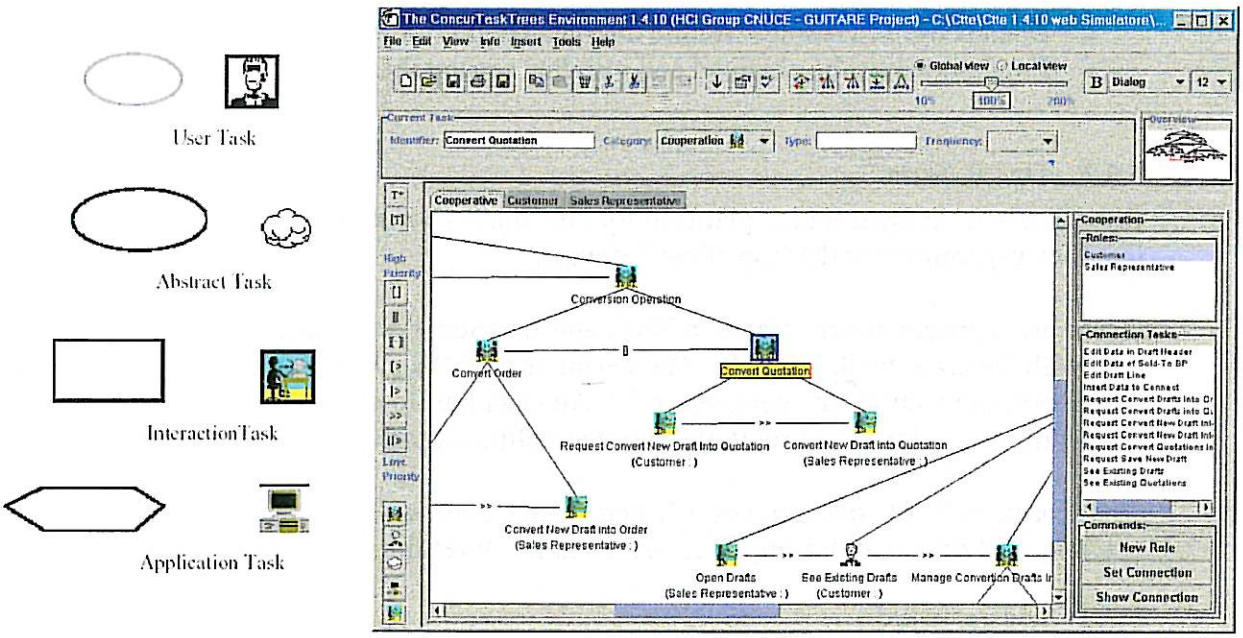


Figure 26: ConcurTaskTrees notation and environment (http://giove.cnuce.cnr.it/ctte.html)

5.5 Some MBUIDEs

Current we find the following commercial tools for designing user interfaces for PDAs and mobile phones:

- IBM WebSphere Studio Device Developer,
- Microsoft Visual Studio with the .net Compact Framework,
- AppForge Crossfire.

These tools allow developers to target many mobile platforms at the same time from one code base. These tools are geared mostly to developers, and assume substantial programming expertise.

5.5.1.1 Crossfire[51]

"Crossfire is one of the easiest ways to deploy intelligent client applications on four very different mobile platforms. Although the Booster virtual machine installation adds more overhead to the requirements, its weight is inconsequential compared to the weight of synchronizing the proprietary SDK's development practices for each respective platform. The solution may be expensive overkill for an easier way to program a rich-client application for the Palm OS, for example, but if that same logic must run unaltered on a Pocket PC and a Nokia Series 60 phone, Crossfire is without question the fastest way to achieve that goal."[52]

Pro:

- Easy to use.
- Deploys applications to multiple mobile platforms from a single code base.
- Impressive set of multimedia widgets and conversion utilities.

Cons:

- Requires AppForge's proprietary Booster Virtual Machine to be installed on any device intended to execute a Crossfire application.
- Expensive.

---

[51] Crossfire: www.appforge.com

[52] Asp:review by Mike Riley:
http://www.aspnetpro.com/productreviews/2004/03/asp200403rm_p/asp200403rm_p.asp

- Doesn't use Microsoft's Mobile Internet Toolkit for ASP.NET
- Emulators must be obtained and installed separately.

5.5.2 OpenLaszlo[53]

OpenLaszlo is an open source platform for creating zero-install web applications with the user interface capabilities of desktop client software.

OpenLaszlo programs are written in XML and JavaScript and transparently compiled to Flash and, with OpenLaszlo 4, DHTML. The OpenLaszlo APIs provide animation, layout, data binding, server communication, and declarative UI. An OpenLaszlo application can be as short as a single source file, or factored into multiple files that define reusable classes and libraries.

OpenLaszlo is "write once, run everywhere." An OpenLaszlo application developed on one machine will run on all leading Web browsers on all leading desktop operating systems.

5.5.3 MasterMind

MasterMind was one of the first systems to integrate multiple models together

MasterMind stands for Models Allowing Shared Tools and Explicit Representations Making Interfaces Natural to Develop. Models in MasterMind are shared via the model server. Whenever a model element in the model server is modified (by request of any tool), all tools that depend on the modified element are informed so that they can update their state. MasterMind's main contribution is that its presentation model is designed to support graphical specification of presentations similar to that of interface builders (Gomaa, Salah et al. 2005).

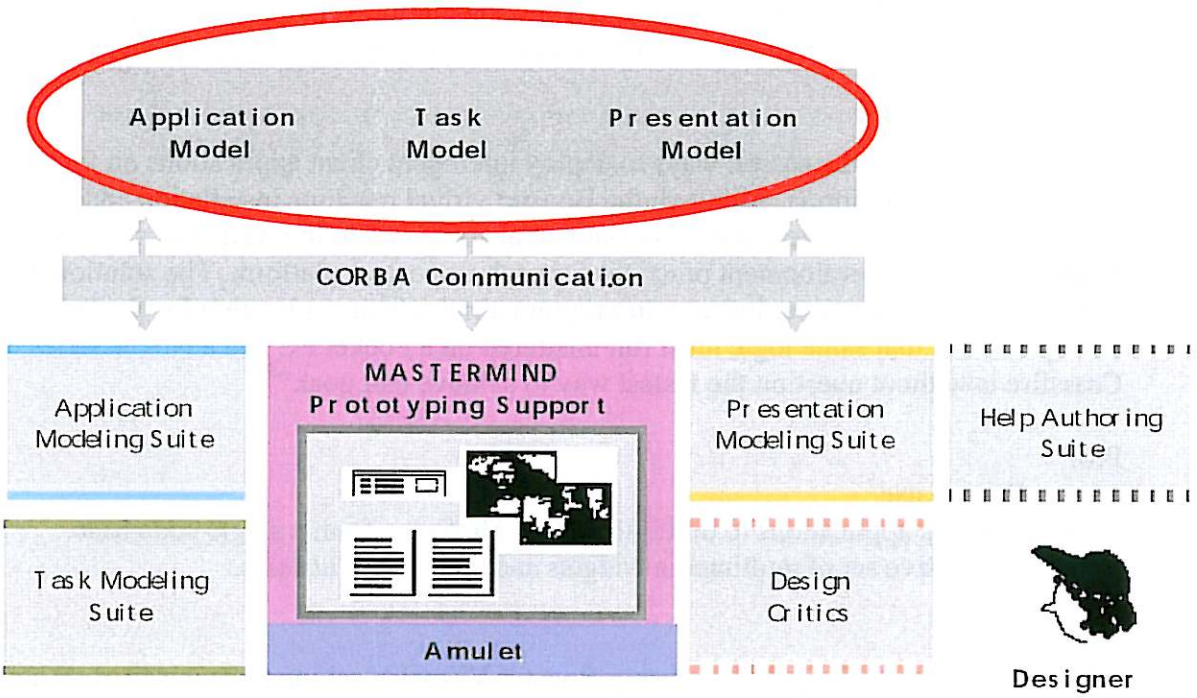


Figure 27: MasterMind

[53] OpenLaszlo: http://www.openlaszlo.org/

5.5.4 Mobi-D

Mobi-D stands for Model-Based Interface Designer. It aims at solving the mapping problem between different abstraction levels for objects in models of interfaces e.g. purely abstract units in such as a user tasks and very concrete units, such as scrollbars and pushbuttons. Developers of Mobi-D believe that the mapping problem defies automation because of the number of variables that can impact each possible mapping. Instead of automation, they propose that model-based systems provide tools that allow developers to interactively set the mappings. The mapping issue is addressed according to three aspects: mapping of domain objects with interactors according to some priorities; style attributes controlling some graphical and textual attributes; and strategy preferences indicating the preferred number of windows, the preferred way to implement sequential constraints, and the preferred interaction and navigation modalities (Gomaa, Salah et al. 2005).

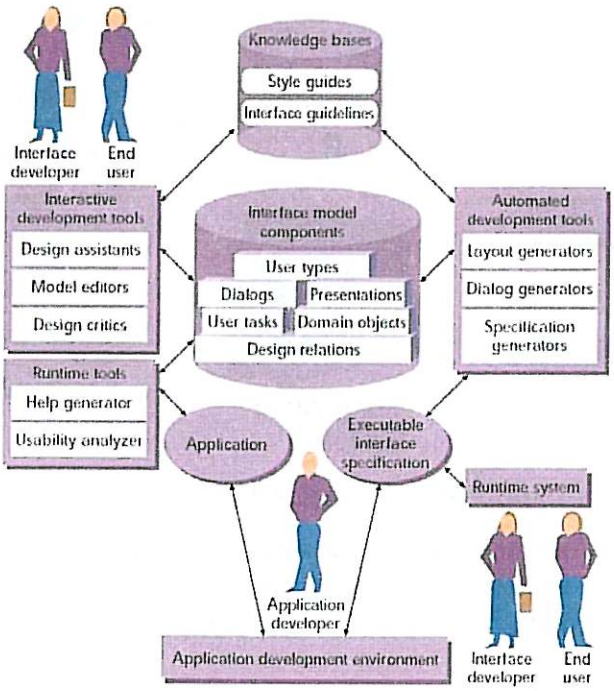


Figure 28: Mobi-D

Mobi-D models:

- *User-task Model:* Describes tasks the user performs and what interaction capabilities are needed
- *Domain Model:* Models of the entities that are manipulated in an interface and their properties
- *Dialog Model:* Describes the human-computer conversation at a low level
- *Presentation Model:* Specifies how the interaction objects appear in all of the different states of the interface.
- *Relations:* Describes how each of the models relate to each other (Tasks/Domain, Dialog/Presentation, Tasks/Dialog, etc.)

Mobi-D provides assistance rather than automating design

5.5.5 MULTIMODAL TERESA[54]

A transformation-based environment, supporting multimodal interfaces designed and developed at the HCI Group of ISTI-C.N.R. It is intended to provide a complete semi-automatic environment supporting a number of transformations useful for designers to build and analyze their design at different abstraction levels and consequently generate the user interface for a specific type of platform. (Mori, Paternò et al. 2003; Mori, Paternò et al. 2004)

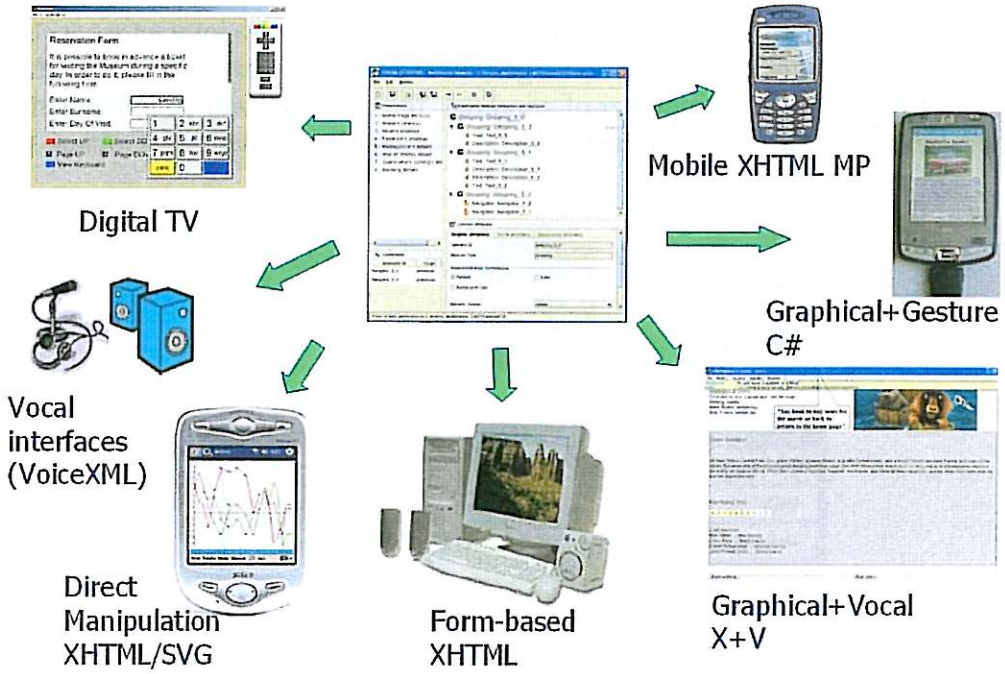


Figure 29: TERESA application domaines: (http://giove.isti.cnr.it/teresa.html)

TERESA is a transformation-based environment that supports the design of an interactive application at different abstraction levels and generates the concrete user interface for various types of platform. Figure 30 shows the main transformations supported in TERESA.

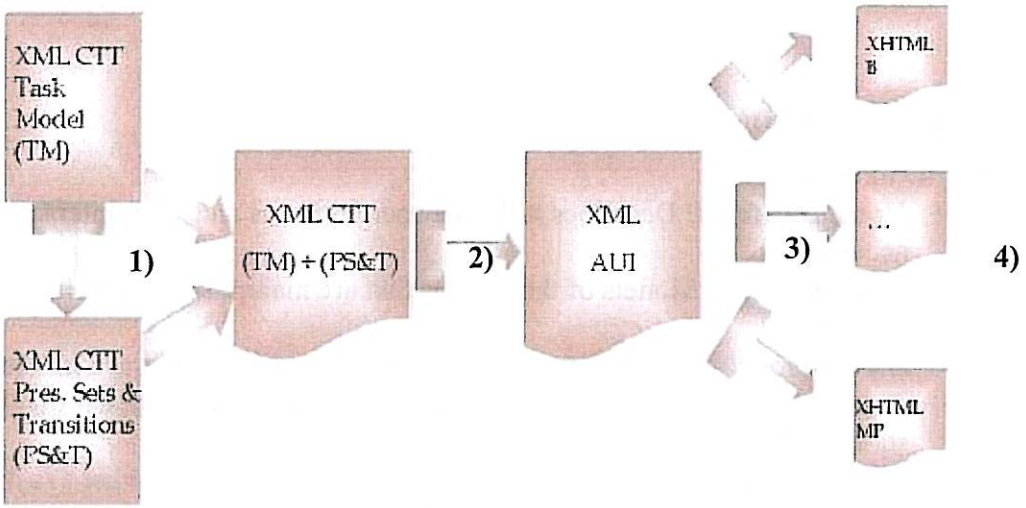


Figure 30: The main transformations in TERESA in terms of XML-based specifications supported.

1) Presentation sets and transitions generation. *From the XML specification of a CTT task model* ("XML CTT Task Model" module it is possible to obtain the set of tasks which are enabled over the same period of time according to the constraints indicated in the model

[54] MULTIMODAL TERASA: http://giove.isti.cnr.it/teresa.html

(enabled task sets). Such sets, depending on the designer's application of a number of heuristics supported by the tool, are grouped into a number of presentation sets and related transitions.

2) *From task model -related information to abstract user interface.* Both the XML task model and Presentation Sets specifications are the input for the transformation generating the associated abstract user interface. The specification of the abstract user interface, in terms of both its static structure (the "presentation" part) and dynamic behavior (the "dialogue" part), is saved for further analyses and transformations.
3) *From abstract user interface to concrete interface for the specific platform.* This transformation starts with the loading of an abstract user interface previously saved for and yields the related concrete user interface for the specific interaction platform selected. A number of parameters related to the customization of the concrete user interface are made available to the designer.
4) *Automatic UI Generation.* Through this option the tool automatically generates the final UI, starting with the currently visualized (single-platform) task model, and using a number of default configuration settings related to the user interface generation.

In short, TERASA is a tool for taking ConcurTaskTrees models, building an abstract interface, and then building a concrete interface on multiple platforms.

5.5.6 XWeb

XWeb enables users to interact with services through automatically generated interfaces in several modalities and client styles, including speech, desktop computers, and pen-based wall displays.

XWeb provides simple approaches for adjusting to different screen sizes

- Shrink portions of the interface
- Add additional columns of widgets

Also capable of generating speech interfaces

- Based on a tree traversal approach like Universal Speech Interfaces

DataView

```
<sprinklers startTime="00:00">
   <days ID="days" sun="0" mon="1" tue="1"
         wed="0" thu="0" fri="0" sat="1"/>
   <zone time="30"/>          <zone time="40"/>
   <zone time="15"/>          <zone time="11"/>
   <zone time="22"/>          <zone time="55"/>
</sprinklers>
```

XView

```
<xview rootID="timer">
  <group ID="timer" loc="" >
    <time loc="" . . .> <name text="Start Time"/>
      . . . . . .
    </time>
    <group loc=""> <name text="Zone durations"/>
      <number loc="0/@time". . .> . . . . </number>
      <number loc="1/@time". . .> . . . . </number>
      . . . . . .
    </group>
    <group loc="days"> <name text="Days"/>
      <enum loc="@mon">
          <name text="Monday"/> . . . .
      </enum>
      <enum loc="@tue">
          <name text="Tuesday"/> . . . .
      </enum>
      . . . . . .
    </group>
  </group>
</xview>
```

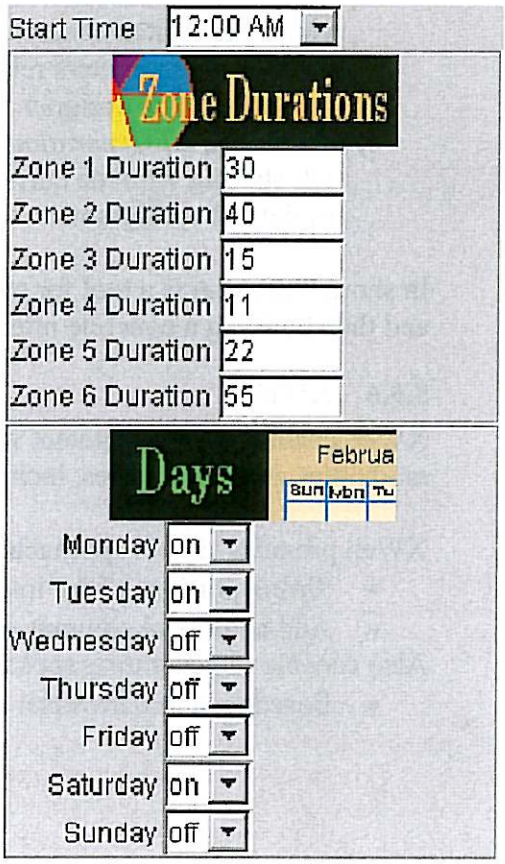


Figure 31: XWeb: DataView and XView

5.5.7 SUPPLE

SUPPLE is a novel toolkit which automatically generates interfaces for ubiquitous applications. Designers need only specify declarative models of the interface and desired hardware device and SUPPLE uses decision-theoretic optimization to automatically generate a concrete rendering for that device (Gajos, Christianson et al. 2005).

SUPPLE takes three inputs: a functional specification of the interface, a device model and a user model. The functional specification defines the types of data that need to be exchanged between the user and the application.

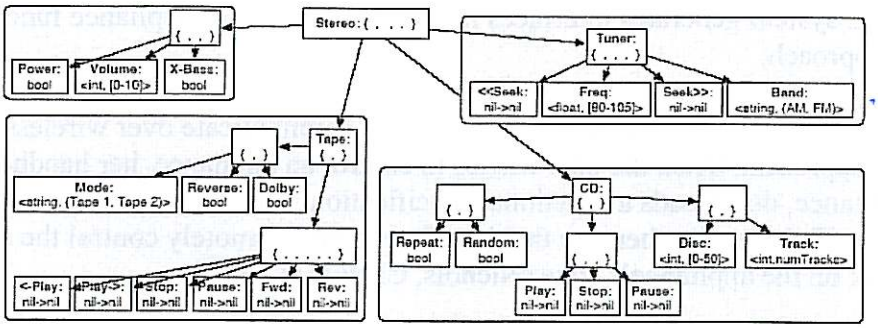


Fig. 1. Graphical representation of the functional specification for a stereo controller. For clarity, different parts of the specification are grouped with gray shading.

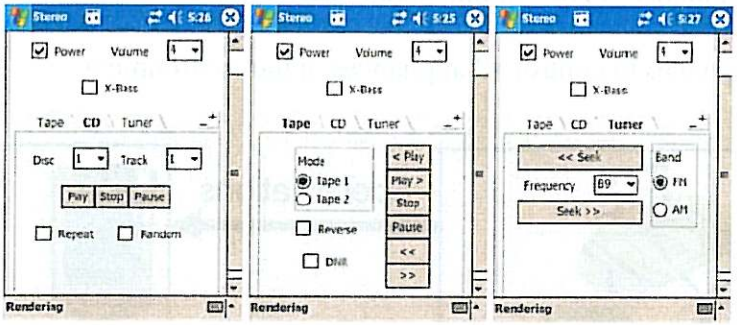


Fig. 2. Three tab views of the stereo specification, rendered for a PDA.

Figure 32: SUPPLE example – functional specification and PDA rendering (Gajos, Christianson et al. 2005).

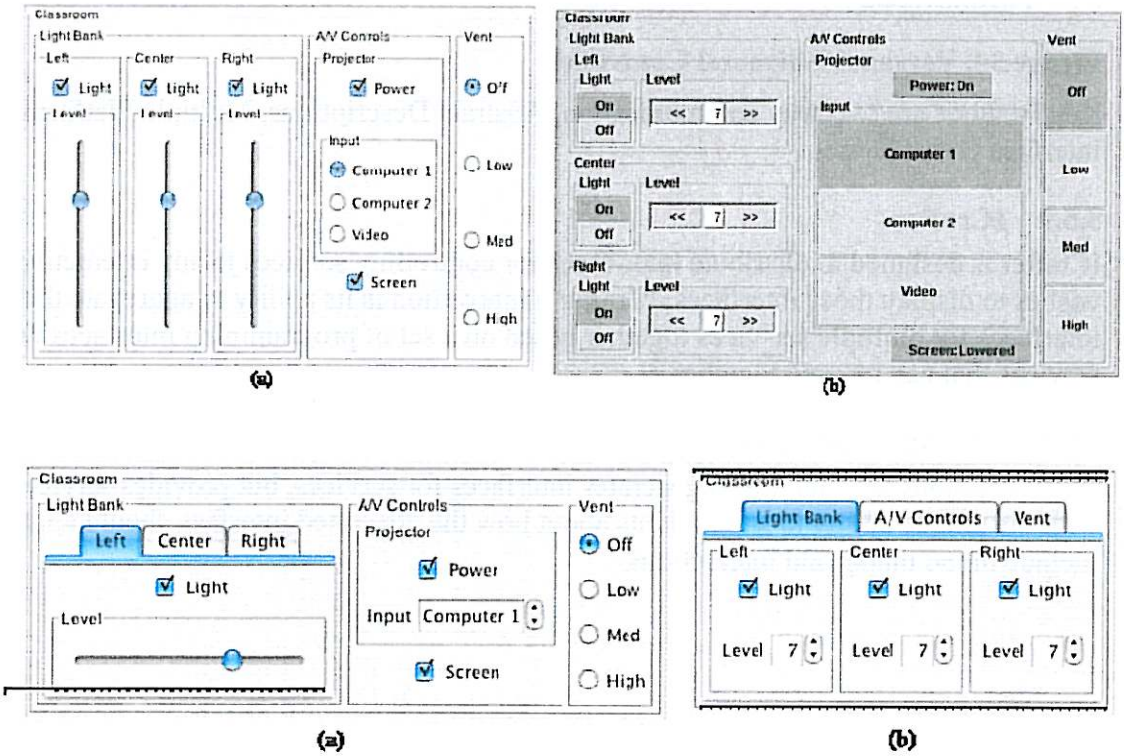


Figure 33: SUPPLE output screenshots examples

5.5.8 Personal Universal Controller (PUC)[55]

The PUC system generates interfaces from specifications of appliance functionality using a rule-based approach.

In the system, handheld devices and appliances communicate over wireless networks using a peer-to-peer approach. When the user wishes to control an appliance, her handheld device connects to the appliance, downloads a functional specification from that appliance, and then generates an interface. The user can then use that interface to both remotely control the appliance and receive feedback on the appliance's state (Nichols, Chau et al. 2007).

Problem: Appliance interfaces are too complex and too idiosyncratic.
Solution: Separate the interface from the appliance and use a device with a richer interface to control the appliance: PDA, mobile phone, etc.

Use mobile devices to control all appliances in the environment

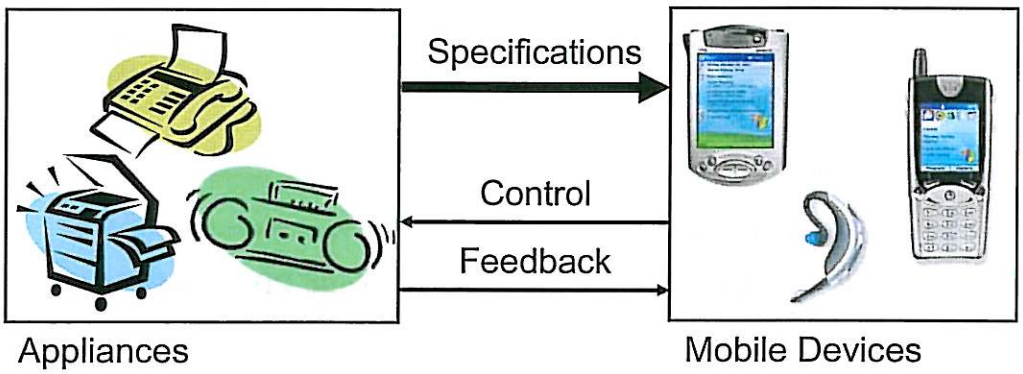


Figure 34: Personal Universal Controller

Key Features are two-way communication, Abstract Descriptions, Multiple Platforms, Automatic Interface Generation

5.5.9 ICrafter

ICrafter is designed to distribute inter-faces for controlling services to any interactive device that wishes to display those interfaces. ICrafter's innovation is its ability to aggregate the user interfaces for multiple ser-vices together based on a set of programming interfaces which identify services that can be used together.

5.5.10 Ubiquitous Interactor

The Ubiquitous Interactor also generates interfaces for services, but provides service provider's with the unique ability to supply hints about how the generated interface should appear and include brand marks and interactions.

[55] PUC: http://www.pebbles.hcii.cmu.edu/puc/index.php

6 Conferences and journals

In this chapter we outline some central conferences regarding model-based form-based mobile user interfaces.

6.1 User Interface Software and Tools (UIST)

UIST (ACM Symposium on User Interface Software and Technology) is the premier forum for innovations in the software and technology of human-computer interfaces. Sponsored by ACM's special interest groups on computer-human interaction (SIGCHI) and computer graphics (SIGGRAPH), UIST brings together researchers and practitioners from diverse areas that include traditional graphical & web user interfaces, tangible & ubiquitous computing, virtual & augmented reality, multimedia, new input & output devices, and CSCW. The intimate size, the single track, and comfortable surroundings make this symposium an ideal opportunity to exchange research results and implementation experiences.
http://www.acm.org/uist

6.2 ACM's Special Interest Group on Computer-Human Interaction

ACM SIGCHI, the ACM's Special Interest Group on Computer-Human Interaction, brings together people working on the design, evaluation, implementation, and study of interactive computing systems for human use. ACM SIGCHI provides an international, interdisciplinary forum for the exchange of ideas about the field of human-computer interaction (HCI).
http://www.sigchi.org/

6.3 Human Computer Interaction with Mobile Devices and Services

MobileHCI provides a forum for academics and practitioners to discuss the challenges, potential solutions and innovations towards effective interaction with mobile systems and services. It covers the analysis, design, evaluation and application of human-computer interaction techniques and approaches for all mobile computing devices, software and services.
http://www.cis.strath.ac.uk/~mdd/mobilehci/

6.4 International Conference on Intelligent User Interfaces (IUI)

IUI is where the community of people interested in Human-Computer Interaction (HCI) meets the Artificial Intelligence (AI) community. We're also very interested in contributions from related fields, such as psychology, cognitive science, computer graphics, the arts, etc. Unlike traditional AI, our focus is not so much to make the computer smart all by itself, but to make the interaction between computers and people smarter. Unlike traditional HCI, we're more willing to consider solutions that involve large amounts of knowledge, heuristics, and emerging technologies such as natural language understanding or gesture recognition.
http://www.iuiconf.org/

7 Reference

1. Ali, M. F., M. A. Pérez-Quiñones, et al. (2002). Building Multi-Platform User Interfaces with UIML. International Workshop of Computer-Aided Design of User Interfaces: CADUI'2002., Valenciennes, France.
2. Bishop, J. (2006). Multi-platform User Interface Construction – a Challenge for Software Engineering-in-the-Small. ICSE'06, Shanghai, China., ACM.
3. Clerckx, T., K. Luyten, et al. (2004). Generating Context-Sensitive Multiple Device Interfaces From Design. the Fifth International Conference on Computer-Aided Design of User Interfaces. CADUI'2004 long paper track, Funchal, Isle of Madeira.
4. Coldewey, J. and I. Krüge (1997). Form-Based User Interface The Architectural Patterns A Pattern Language. Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, Germany,.
5. Cooper, R., J. McKirdy, et al. (2000). Conceptual Modelling for Database User Interfaces. the Fifth Working Conference on Visual Database Systems: Advances in Visual Information Management, Kluwer, B.V. Deventer, The Netherlands, The Netherlands.
6. Coyette, A. and J. Vanderdonckt (2005). Computer Assisted Sketching for the Early Stages of User Interface Design. 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2005, Rome.
7. Draheim, D. and G. Weber (2003). Storyboarding Form-Based Interfaces. Human-Computer Interaction -- INTERACT'03, IOS Press, (c) IFIP.
8. Eisenstein, J., J. Vanderdonckt, et al. (2001). Applying Model-Based Techniques to the Development of UIs for Mobile Computers. the 6th international conference on Intelligent user interfaces, Santa Fe, New Mexico, United States.
9. Eva, Y. h. C. (2002). Resource-Based User Interface Design. Department of Computer Science, The University of York.
10. Florins, M. (2006). Graceful Degradation: a Method for Designing Multiplatform Graphical User Interfaces. Management Sciences. Louvain-la-Neuve, Belgium, The Université catholique de Louvain: 212.
11. Gajos, K., D. Christianson, et al. (2005). Fast and Robust Interface Generation for Ubiquitous Applications. Ubicomp'05, Tokyo, Japan.
12. Gajos, K. and D. S. Weld (2004). SUPPLE: Automatically Generating User Interfaces. the 9th international conference on Intelligent user interfaces, Funchal, Madeira, Portugal.
13. Gomaa, M., A. Salah, et al. (2005). Towards A Better Model Based User Interface Development Environment: A Comprehensive Survey. MICS 2005.
14. Gong, J. and P. Tarasewich (2004). Guidelines for Handheld Mobile Device Interface Design. DSI 2004 Annual Meeting.
15. Graham, I. (2003). A pattern language for web usability, Addison-Wesley.
16. Hekmatpour, S. and D. C. Ince (1986). "Forms as a language facility." SIGPLAN **21**(9): 42-48.
17. Henninger, S. (2001). An Organizational Learning Method for
18. Applying Usability Guidelines and Patterns. the 8th IFIP International Conference on Engineering for Human-Computer Interaction, Toronto, Canada, Springer-Verlag London, UK.
19. Iverson, L. (2004). The DKC Model: An Application Model for Collaborative Work.
20. Limbourg, Q., J. Vanderdonckt, et al. (2004). USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. the 3rd annual conference on Task models and diagrams, Prague, Czech Republic.
21. Lin, J. (2005). Using Design Patterns and Layers to Support the Early-Stage Design and Prototyping of Cross-Device User Interfaces. UNIVERSITY OF CALIFORNIA, BERKELEY.

22. Lin, J. and J. Landay (2002). Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing), San Francisco, CA,.
23. Molina, P. J., S. Meli´a, et al. (2002). User Interface Conceptual Patterns. Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, Springer-Verlag.
24. Mori, G., F. Paternò, et al. (2003). Tool Support for Designing Nomadic Applications. 7 Int. Conf. on Intelligent User Interfaces IUI'03.
25. Mori, G., F. Paternò, et al. (2004). "Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions." IEEE Transactions on Software Engineering **30**: 507-520.
26. Myers, B., S. E. Hudson, et al. (2000). "Past, present, and future of user interface software tools." ACM Transactions on Computer-Human Interaction (TOCHI) **7**(1): 3-28.
27. Nichols, J., D. H. Chau, et al. (2007). Demonstrating the Viability of Automatically Generated User Interfaces. CHI 2007, San Jose, California, USA., ACM.
28. Nilsson, E. G. (2002). Combining Compound Conceptual User Interface Components with Modelling Patterns - A Promising Direction for Model-Based Cross-Platform User Interface Development. the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, Springer-Verlag, London, UK.
29. Nilsson, E. G., J. Floch, et al. (2006). "Model-based user interface adaptation." Computers & Graphics **30**(5): 692–701.
30. Nilsson, E. G., J. Floch, et al. (2006). Using a Patterns-based Modelling Language and a Model-based Adaptation Architecture to Facilitate Adaptive User Interfaces. DSV-IS 2006, The XIII International Workshop on Design, Specification and Verification of Interactive Systems.
31. Pribeanu, C., Q. Limbourg, et al. (2001). Task Modelling for Context-Sensitive User Interfaces. the 8th International Workshop on Interactive Systems: Design, Specification, and Verification.
32. Puerta, A. and J. Eisenstein (2002). XIML: A Common Representation for Interaction Data. International Conference on Intelligent User Interfaces, San Francisco, California, USA, ACM.
33. Rolfsen, R. K., D. Boell, et al. (2007). Model-generated workplaces: An interoperability approach" til konferansen (skrevet av. 3rd International Conference Interoperability for Enterprise Software and Applications (I-ESA 07), Funchal (Madeira Island), Portugal.
34. Schlungbaum, E. and T. Elwert (1996). Automatic User Interface Generation from Declarative Models. CADUI 96.
35. Seffah, A., P. Forbrig, et al. (2004). "Multi-devices ''Multiple'' user interfaces: development models and research opportunities." The Journal of Systems and Software **73**: 287–300.
36. Szekely, P. (1996). Retrospective and Challenges for Model-Based Interface Development. Design, Specification and Verification of Interactive Systems '96, Springer-Verlag, Wien.
37. Szekely, P., P. Sukaviriya, et al. (1996). "Declarative interface models for user interface
38. construction tools: the MASTERMIND approach." Engineering for Human-Computer Interaction.
39. Trætteberg, H. (2002). Model-based User Interface Design. Department of Computer and Information Sciences (Information Systems Group). Trondheim, Norway, Norwegian University of Science and Technology (Faculty of Information Technology, Mathematics and Electrical Engineering): 211.
40. van Welie, M. and H. Trætteberg (2000). Interaction Patterns in User Interfaces. I7th. Pattern Languages of Programs Conference, Allerton Park Monticello, Illinois, USA.

41. Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems. 17th Conf. on Advanced Information Systems Engineering CAiSE'05, Porto, Springer-Verlag, Berlin.
42. Vanderdonckt, J. M. and A. R. Puerta (1999). Introduction to Computer-Aided
43. Design of User Interfaces. the third international conference on Computer-aided design of user interfaces, Louvain-la-Neuve, Belgium, Kluwer Academic Publishers.

User Interface Modelling

Erik G. Nilsson, SINTEF ICT

1. User interface development

Basically, there are four approaches for implementing a user interface (UI)[1]:

1. Programming the UI. This involves using a programming library (or application programming interface – API) that facilitates implementation of a user interface using components, classes, methods, functions etc.
2. Drawing the UI. This involves using a screen painter to make a "painting" of the intended user interface, combined with a programming facility (textual or graphical) to specify the behaviour of the UI.
3. Model the UI. This involves using a modelling language (graphically and/or textual) for specifying the UI on some level of abstraction, and applying a mapping process (automatic, semiautomatic, or manual) transforming the UI model to a running user interface (generated code or by interpreting the model by a run-time system which may be part an application or a separate tool).
4. Mark up the UI. This involves using a mark-up language (HTML, WML, XML, etc.) to specify the contents and to a varying degree the appearance of a UI. This specification is interpreted by a generic client (browser).

Here, we look at combinations of approach 1, 3 and 4, and investigate how models may be utilized in different ways when developing UIs. In approach 1, this means e.g. having some sort of model(s) describing various aspects of a UI that the program reads and use as input to the design of the UI turns out. In approach 4, the mark-up code as such may be viewed as models, but more important are cases where the mark-up languages are target code, i.e. the result from a mapping process.

1.1 Different ways of using UI models

In this section we look at different "levels" of using UI models, where the levels denote how general the solution is. The three levels we discuss are using UI models as a part of a specific application, having domain specific solutions and having generic modelling languages.

1.1.1 Application specific UI models

This "level" covers cases where it for some reason is convenient to specify some of the information that is needed for showing the UI of an application outside the code that runs the UI. A traditionally implemented UI (almost) all of the details about the UI design of the UI is specified at design time (usually using approach 1 or 2 listed above). Using models in this context is usually a means for postponing some design choices to run time. In practice, this means that the information needed for these postponed choices are stored on file(s) or in a database, that is read when the program runs, and used as a basis for generating or adapting (parts of) the UI.

---

[1] Mark that a given tool may use a combination of these four approaches, and that the approaches overlap

A typical solution following this scheme may be used if a part of an application uses a set of attributes that changes over time (i.e. which attributes that are used, not the attribute values). Instead of having to change the application each time the set of attributes change, the application may have a configuration part that lets a super user specify the current attributes to use (and maybe how the attributes are grouped into different tab folders). This may typically be stored in a configuration file that is read by the application when the relevant part(s) of the UI is initiated. If the number of attributes / the way the attributes are presented vary, there is a need for a layout algorithm that decides how the components representing the configured attributes are placed inside its container(s). Such layout algorithms are needed for all solutions (and indeed on the "levels" presented below) where some portion of the UI is generated based on dynamic information.

1.1.2 Domain specific UI models

This "level" covers cases where a specific modelling language is constructed for a specific domain [Nilsson99]. Thus, these models are more generic than the application specific ones, but more specialized than the generic ones described below. Domain specific UI models may e.g. be used in a highly configurable standard system for a specific trade. Such systems usually have a kernel of common functionality, but large parts may be adapted to each organization using the system.

To ease the configuration of such systems, smaller or larger part of the systems may be adapted using models instead of implementing the specific part using traditional means. The models will typically both cover the functional part of the system (domain models) and the UI (UI models). Similar to [Balzert95, Märtin96, and Roberts98], the user interface model in such solutions may, but need not, be an integral part of the conceptual model of the system. The information in the models involved may be divided into three parts:

1. Model information that is only interesting for the other parts of the system than the UI. This includes parts of the model that are not showed in the UI, computational parts, etc.
2. Model information that is interesting both for the UI and for other parts of the system. This includes e.g. information about which object types to exclude in a specific variant, additional attributes, names, data types, etc. for attributes, address formats, enumeration values for enumerator type attributes, etc.
3. Model information that is only interesting for the UI. This includes e.g. which UI element type to use in forms based presentations, maximum number of characters, whether an attribute should be suppressed in the UI, the sequence in which the attributes should be shown in the presentations, which attribute to use as "label" in the visual presentations, etc.

Such solutions may include tailored UI components for compound data types such as enumeration, address and co-ordinates, duration, and time. "Label" attributes may be very convenient in various presentations, in choosing which information to show in tool tips, in window headings, etc.

It should be marked that the division of the models in the three parts just presented is fairly general. This means that it may also be used in advanced solutions applying application specific UI models, and it may also relevant for solutions based on generic

UI models, although these solutions usually make a clearer distinction between different types of models.

Possible configuration enhancements on this "level" may also include possibilities for defining dialog dynamics like user interface actions (typically navigation between windows/dialogs), and functionality like defining pop-up menus with actions, to tailor icons, and to define which icons to use depending on attribute values.

1.1.3 Generic UI models

Generic UI models are used in languages and tools that are not application or domain specific. As the term indicates, the models are meant to be used to model an arbitrary application. Thus, these types of models are usually utilized by general, model-based systems development aids. Being generic, the modelling languages involved should be able to describe any possible UI. Despite this, the modelling languages and tools in this category are usually limited to a certain style of user interaction. In many cases, they are restricted to forms based application.

Types of UI models

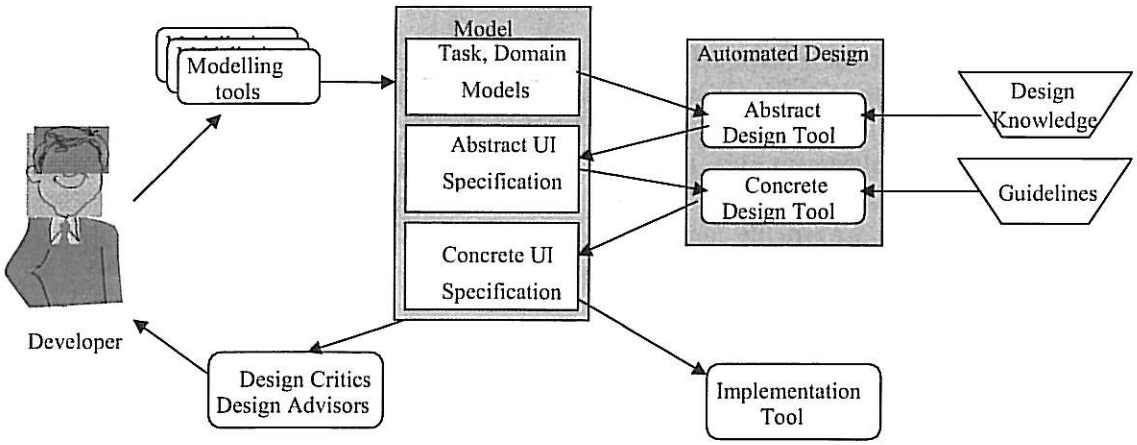
Within the UI modelling research tradition, a number of model types are exploited. Some of the models are used as input to the UI modelling activities; others are the results of the UI modelling. These are the main categories [?Schlungbaum96 (or 99)?, Puerta96, Puerta97, Wilson96, Vanderdonckt99b]:

- task models - descriptions of the tasks performed or to be performed by the users
- domain models - descriptions of the application area the UI is to support. These models are usually used as input, and are expressed as object or data models (in some cases expressed in UML).
- dialog models - descriptions of the dynamics of a UI on a fairly high level. Sometimes denoted as conversation models.
- presentation models - descriptions of the composition of UIs, and to some extent the appearance.
- behaviour models - descriptions of the behaviour of a UI on a fairly low level.
- interaction models - a common term for dialog, presentation and behaviour models. These models are most important for describing a user interface.
- design knowledge models - formalizations of how a UI is to be design to be usable. This is knowledge that is needed to be able to design a UI automatically. Often, these models are only represented implicitly as functionality in automated design tools.
- application models - descriptions of aspects of an application not covered by the models above.
- platform models - descriptions of the equipment the UI is to run on (implementation platform)
- user models - descriptions of users, e.g. knowledge and experience possessed by the users.
- work place models - descriptions of the work and cultural contexts the UI is to function within.
- composite models - models that cover a number of the areas above in one model.

Most work on model based UI development utilizes at least one - preferably a number of - the interaction models to describe the appearance and behaviour of the UI to be

developed. <<Which of the other model types that are used vary (see the description of *Modelling approaches* and *Modelling languages* below).>>

In spite of these differences, the architecture exploited by most languages and tools similar [Szekely96, Puerta97]. The figure below (a simplified version of a figure from [Szekely96]) shows how most tools work.



In the two "levels" presented above (application and domain specific), the "normal" way of using the models is to have mechanisms, either generic or part of an application, that are able to read the models and interpret the information in the model, i.e. transform the models to a running UI, at run time. For generic UI models, transforming the models to running UIs are traditionally done using code generation facilities. I.e. the process of transforming the model to code is done by a designated tool at design time. Although there are good reasons for this, it should be noted that also for generic UI models, interpreting the models (i.e. doing the model transformation) at run time is a possible solution. Furthermore, utilizing such models as a means for offering adaptive solutions requires that the models are interpreted at some level (see <ref. to section below>).

In the list of model types above it is convenient to make a distinction between the models that are made for other purposes than describing a UI, and the models that primarily describe different issues regarding the UI. In the list above, the domain and task models are usually in the first group, while the other are in the second. This distinction corresponds roughly to the two topmost white boxes inside the grey box labelled "Models" in the architecture figure (from [Szekely96]) above.

As can be seen from the figure, the models in the first group are primarily used as input for generating the models in the second group. In MDA terms, this may be denoted transformation between different types of PIM models on similar level of abstraction. This type of transformation is sometimes called derivation or translation. As can also be seen from the figure, the second group of models are used as input for model transformation to models on a lower level of abstraction. In MDA terms, this is PIM to PSM transformation, sometimes called refinement or reification.

Within the model-based UI development research community there are two "schools" – the ones using domain models and the ones using task models as the starting point. Of course, most approaches use both type of models to some extent, but almost all approaches have a main foundation in one of these "schools".

Generally, the task model "school" requires more manual work to map both from the models used as starting point to the UI models, and specially from the UI models to a running UI. On the other hand, UIs developed using an approach from this school tends to be more *task-oriented* than domain model based UIs.

UIs developed using an approach from the domain model "school" tend to be more data oriented, often with only more or less generic operations available (Create, Store, Delete, First, Last, Next, etc.) – i.e. *data-oriented*. On the other hand, approaches in this school usually do mappings from domain models to UI models and from UI models to running UI in a more automated manner. The reason for this is essentially that it is easier to generate a data-oriented user interface than a task-oriented one.

2. Reasons for using UI models

An important rationale for using MDA in general is enhanced productivity in the development phase. This is also partly a reason for using UI models, but seldom alone. I.e. using UI models to develop a fairly straightforward UI on one platform will not cause a big productivity gain (if any at all) unless there is an available development tool that is very well suited to the given UI.

Productivity gains are (as often is the case with MDA in general) usually connected to cross-platform development. I.e. when a UI is to run on a number of platforms, using common models across these platforms may give substantial savings.

The considerations above are mainly connected to generic UI models. Looking at domain and especially at application specific models, the main reason for utilizing UI models is to facilitate some level of flexibility in the UIs. Such flexibility is usually connected to changes in the UI based on either information obtain by the application or specifications in the configuration of the application. Changes based on information, e.g. that some fields in a UI are added based on choices made by the UI, may be solved using models, but are usually handled using different means. Thus, in this context changes based on configurations are more relevant. Such changes are often performed at start-up, but may also be done during execution of the application, e.g. based on stimuli external to the application. The latter case is usually called adaptive behaviour rather than configurable, but the borders are not distinct.

Along a different axis, adaptive/configurable UIs are examples of UIs whose final design is determined at run-time. Traditionally, the design of a UI that is specified using screen-painters (usually forms-based ones) is determined at design time by the drawing in the screen painter. In addition to adaptive and configurable UIs whose portion of the UI that is determined at run time often is fairly small, there are classes of UIs that only to a small extent are possible to determine at design time. Examples of such UIs are visual UIs consisting of symbols/icons and utilizing drag and drop as interaction mechanism, like diagram editors, screen painters and file managers. These UIs share the characteristics that the major parts of the UIs reflect underlying data.

Traditionally, such UIs are not model-based, but using models e.g. to describe layout rules or visual details may make such UIs more flexible.

3. Types of information in UI models

Most of the terms and principles presented in this section are general terms and principles used when developing UIs, regardless of whether UI models are used or not[2].

One of the most central concepts when specifying UIs (regardless of which approach that is used) is the concept of an *instance hierarchy of UI components.* I.e. any UI is composed of a set of UI components (each one being an instance of a component type) in a hierarchical structure. The instance hierarchy represents how the UI is built, where the root in the hierarchy represents the outmost component (often a window, but it may also be some other type of container), and the children representing components inside each other. For a UI designed using a screen painter utilizing concrete platform-specific UI components (often called UI controls or widgets), the instance hierarchy represents the exact instance hierarchy of the UI controls at run time. For UIs specified using some type of modelling language utilizing abstractions of the concrete UI controls, the instance hierarchy in the models may vary from the instance hierarchy of the concrete running UI on a given platform, both because the modelling language hide implementation details and because the modelling language offers compound components as building blocks. E.g. the modelling language may offer a choice element as building block. This may be mapped to different concrete UI controls (a set of radio buttons, a combo box, etc.) on different platforms/on the same platform depending on different properties of the choice element.

When building instance hierarchies (independent of abstraction level of the building blocks), it is common to make a distinction between *containers* and *simple components* (the control concept is often used only for simple components, while UI component or widget usually cover both). Containers may act as parent to other components, while simple components always are leaf nodes in an instance hierarchy. The containers usually have a visual appearance, but may also be invisible, i.e. acting only as grouping mechanism. Some UI libraries offer inheritance mechanisms on the instance level, i.e. some properties (like background colour) set on a container is automatically inherited by all its children unless the property is overridden by the child component. This type of instance inheritance must not be confused with inheritance in sub-/super-type hierarchies.

As indicated above, when introducing abstraction in a UI modelling language, it is natural to also introduce compound components. Simple examples of this are radio group (a container embedding a number of radio buttons) or a labelled entry field (a container embedding a label control and some data entry component, usually a text entry field). More complex compound component are a file selection dialog or a browser component consisting of a set of connected list boxes and a details pane.

---

[2] Or one may say that all four approaches for developing UIs presented above include models represented in some way, which is correct in the sense that when a UI is drawn in a screen painter, the drawing is indeed a model, even though it looks very much like the final UI. To accept that a program that produces a running UI also may be viewed as a model of the UI requires that the model concept is stretched a bit.

4. Levels of abstraction

As mentioned in the prior section, UI models use concepts on different levels of abstraction. At a low level, a language may offer a *radio group* concept that is an abstraction of a set of radio buttons in a frame component on different implementation platforms. At a higher level, a language may offer a *choice element* concept that is an abstraction of radio group, drop-down list box, list box, etc. On different implementation platforms only a number of the concrete components may be available, but as long as at least one of them is available, the abstract specification may be mapped to the platform.

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

One way of handling this is to divide the specification of a given user interface in two parts, one describing the commonalities across the platforms and one describing the specialities on each platform. Of course, there will one common model and a certain number of platform-specific ones. In many languages and tools, e.g. in UIML [UIML] – one of the languages with broadest platform support – this division between the common and the platform-specific models must be done at a quite early stage in a user interface specification [Nilsson01]. Furthermore, the amount of specification code constituting the platform-specific parts tends to be much more voluminous than the common part. In such a situation, it is relevant to question whether a model-based approach gives any benefit over the most relevant alternative, which is to develop the user interface on each platform from scratch [Nilsson02a].

An approach for meeting this challenge is to base the modelling language on *compound* user interface components. By using composite user interface components that act as abstraction of typically *patterns* that occur in most user interfaces, the developer will be offered much more powerful building blocks than what is commonly used nowadays (usually only atomic elements like buttons, text fields, menus, etc.). For the user interface patterns to be applicable, it must be possible to couple the user interface patterns to corresponding model patterns in the domain models [Nilsson02b].

Using patterns-based composite user interface components also gives major benefits when the specifications are to be mapped to specific platforms having different combinations of screen size, interaction style and interaction mechanisms. As the number of compound components is limited, it is possible to optimize how the compound components are mapped to the various platforms. In this way, the desired combination of abstraction level and expressive power is obtained. The chosen approach also to some extent will solve the usability problem for generated user interfaces, as the mapping process will be more predictable when the building blocks are larger.

5. Configuring applications – what and who

Traditionally, it has been common to make a distinction between end-user applications and development tools. When having configurable applications and/or applications that are partly configured at run time using UI models, the distinction becomes less clear. The same is the case if one considers applications like spreadsheets where the distinction between development and use is at the best fuzzy. This may be viewed as a continuum with tailored, specialised, non-configurable applications in one end and general systems development tools (including most model-based UI development tools) in the other [Nilsson99]. This is shown in *Figure 1*.

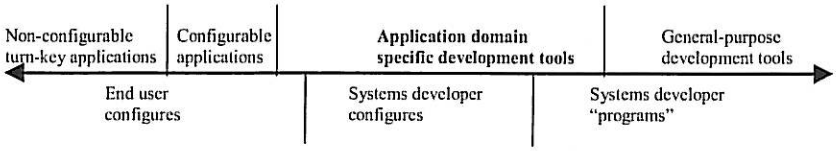


Figure 1. Application domain specific development tools compared to turn-key applications and development tools

In general, tools situated to the left are less configurable, but also easier to configure than tools on the right side (having fewer and more usable configuration mechanisms). As indicated in the figure, this also influences whom the configuration mechanisms are targeted at, and what kind of "actions" the person doing the configuration uses. "End user" may in this context also be a super user or system administrator.

The generality and flexibility obtained on the right hand side, often has a price: the developer must do very much work to develop an application.

6. References

[Balzert95] H. Baltzert: *From OOA to GUI - the JANUS System.* In "Proceedings of Interact '95"

[Märtin96] C. Märtin: *Software Life Cycle Automation for Interactive Applications: The AME Design Environment.* In "Computer-Aided Design of User Interfaces – Proceedings of CADUI '96"

[Nilsson99] E. G. Nilsson: *Using application domain specific run-time systems and lightweight user interface models - a novel approach for CADUI.* In Proceedings of CADUI '99

[Nilsson01] E. G. Nilsson: *Modelling user interfaces – challenges, requirements and solutions.* Proceedings of Yggdrasil 2001 – Norwegian Computer Society's annual conference on user interface design and user documentation.

[Nilsson02a] E. G. Nilsson: *User Interface Modelling and Mobile Applications – Are We Solving Real World Problems?* Proceedings of Tamodia'2002.

[Nilsson02b] E. G. Nilsson: Combining compound conceptual user interface components with modelling patterns – a promising direction for model-based cross-platform user interface development. In Proceedings of DSV-IS 2002

[Puerta96]A. Puerta: *Work Group Report: Issues in Automatic Generation of User Interfaces in Model-Based Systems,* In "Computer-Aided Design of User Interfaces – Proceedings of CADUI '96"

[Puerta97] A. Puerta: *A Model-Based Interface Development Environment.* In "IEEE Software, July/August 1997".

[Roberts98] D. Roberts, D. Berry, S. Isensee, J. Mullaly: *Designing for the User with OVID: Bridging User Interface Design and Software Engineering.* Macmillan Technical Publishing, 1998

[Schlungbaum96] E. Schlungbaum and T. Elwert: *Automatic User Interface Generation from Declarative Models.* In "Computer-Aided Design of User Interfaces – Proceedings of CADUI '96"

[Szekely96] Pedro Szekely: *Retrospective and Challenges for Model-Based Interface Development.* In "Computer-Aided Design of User Interfaces – Proceedings of CADUI '96"

[Vanderdonckt99b] J. Vanderdonckt and A. Puerta: *Introduction to Computer-Aided Design of User Interfaces.* In "Computer-Aided Design of User Interfaces II – Proceedings of CADUI '99"

[Wilson96] S. Wilson: *Work Group Report: Reflections on Model-Based Design: Definitions and Challenges,* In "Computer-Aided Design of User Interfaces – Proceedings of CADUI '96"

[UIML] Universal Interface Technology: *White paper: The UIML Vision.* Available at http://www.universalit.com/uiml/UIMLVisionWhitePaperV4b.pdf

SINTEF

SINTEF ICT

Address: P.O.Box 124, Blindern
0314 Oslo NORWAY
Location: Forskningsveien 1
0373 Oslo
Telephone: +47 22 06 73 00
Fax: +47 22 06 73 50

Enterprise No.: NO 948 007 029 MVA

MEMO

MEMO CONCERNS

FLAMINCO
Generation of Form-based Mobil User Interface

Literature survey

	FOR YOUR ATTENTION	COMMENTS ARE INVITED	FOR YOUR INFORMATION	AS AGREED
DISTRIBUTION Erik Nilsson FLAMINCO project members		X	X	

FILE CODE	CLASSIFICATION
	Open

ELECTRONIC FILE CODE

Appendix B - Literature survey.doc

PROJECT NO.	DATE	PERSON RESPONSIBLE / AUTHOR	NUMBER OF PAGES
	2007-04-17	Rolf Kenneth Rolfsen	28

1 Selected papers

1.1 Generating context-sensitive multiple device interfaces from Design (+) (Clerckx, Luyten et al. 2004)

This paper shows a technique that allows adaptive user interfaces, spanning multiple devices, to be rendered from the task specification at runtime taking into account the context of use. The designer can specify a task model using the ConcurTaskTrees Notation and its context dependent parts, and deploy the user interface immediately from the specification. By defining a set of context-rules in the design stage, the appropriate context-dependent parts of the task specification will be selected before the concrete interfaces will be rendered. The context will be resolved by the runtime environment and does not require any manual intervention. This way the same task specification can be deployed for several different contexts of use. Traditionally, a context-sensitive task specification only took into account a variable single deployment device. This paper extends this approach as it takes into account task specifications that can be executed by multiple co-operating devices.

FLAMINCO Classification: model-based mobile UI
Contribution: technique: ConcurTaskTrees Notation
Application: adaptive user interfaces, multiple devices
Approach: task model, context model, run-time rendering

1.2 Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions (+) (Mori, Paternò et al. 2004)

The increasing availability of new types of interaction platforms raises a number of issues for designers and developers. There is a need for new methods and tools to support development of nomadic applications, which can be accessed through a variety of devices. This paper presents a solution, based on the use of three levels of abstractions, that allows designers to focus on the relevant logical aspects and avoid dealing with a plethora of low-level details. We have defined a number of transformations able to obtain user interfaces from such abstractions, taking into account the available platforms and their interaction modalities while preserving usability. The transformations are supported by an authoring tool, TERESA, which provides designers and developers with various levels of automatic support and several possibilities for tailoring such transformations to their needs.

FLAMINCO Classification: model-based mobile UI
Contribution: tool: TERESA (authoring tool)
Application: multi-devices
Approach: « abstract modelling », three model levels, model transformation (MDA like approach)

1.3 ISML: An Interface Specification Meta-Language ( ) (Crowle and Hole 2003)

In this paper we present an abstract metaphor model situated within a model-based user interface framework. The inclusion of metaphors in graphical user interfaces is a well established, but mostly craft-based strategy to design. A substantial body of notations and tools can be found within the model-based user interface design literature, however an explicit treatment of metaphor and its mappings to other design views has yet to be addressed. We introduce the Interface Specification Meta-Language (ISML) framework and demonstrate its use in comparing the

semantic and syntactic features of an interactive system. Challenges facing this research are outlined and further work proposed.

FLAMINCO Classification: model-based UI
Contribution: language: ISML, framework
Application: model-based UI
Approach: metaphor modelling, design view mappings
Study:

1.4 Fast and Robust Interface Generation for Ubiquitous Applications (+) (Gajos, Christianson et al. 2005)

We present Supple, a novel toolkit which automatically generates interfaces for ubiquitous applications. Designers need only specify declarative models of the interface and desired hardware device and Supple uses decision-theoretic optimization to automatically generate a concrete rendering for that device. This paper provides an overview of our system and describes key extensions that barred the previous version (reported in [3]) from practical application. Specifically, we describe a functional modeling language capable of representing complex applications. We propose a new adaptation strategy, split interfaces, which speeds access to common interface features without disorienting the user. We present a customization facility that allows designers and end users to override Supple's automatic rendering decisions. We describe a distributed architecture which enables computationally-impoverished devices to benefit from Supple interfaces. Finally, we present experiments and a preliminary user-study that demonstrate the practicality of our approach.

FLAMINCO Classification: Model-based mobile UI
Contribution: tool: SUPPLE (toolkit)
Application: multi-devices, ubiquitous applications
Approach: declarative modelling, functional modelling, automatic generate user interface
Study: experiments, preliminary user-study

1.5 Model Based Approach For Context Aware And Adaptive User Interface Generation (+) (Hanumansetty 2004)

User interface design and development for ubiquitous software applications is challenged by the presence of varying contexts. Context comprises of user's computing platform, the environment in which the user is interacting with the application and user characteristics which comprise of user's behavior during interaction and user preferences for interface display and interaction. We present a framework for adaptive user interface generation where adaptation occurs when context changes. This framework introduces three new concepts. First, formalization for representing context is introduced. Our design of context specification is unique since it reflects the association of context with level and nature of user interface adaptation. Secondly, user interface generation life cycle is studied and we define a context model on top of task model to introduce the contextual conditions into user interface generation process. Using the context model, user interface designer can specify contextual requirements and its effect on the user interface. Third, context aware adaptation of user interfaces is achieved by mapping context specifications to various levels of user interface generation life cycle. We designed a specification language called rule specification using which the user interface designer can specify the mapping. With the new design of context representation, context model, and rule specification, we demonstrate how changes in contexts adapts task model which in turn adapts the user interface.

FLAMINCO Classification: model-based mobile UI
Contribution: framework
Application: multi-devices, context aware UI, adaptive UI
Approach: context modelling, context and UI-mapping through rule specifications, task modelling
Study: demonstration

1.6 Just-UI: A User Interface Specification Model ( ) (Molina, Meliá et al. 2002)

A Model for the Specification of Abstract User Interfaces based on Conceptual Patterns is proposed to enhance the semantic captured by an object-oriented analysis method. The model gathers both Presentation and Navigation issues. A graphical notation is also provided to make easier the specification tasks. This simple graphical notation allows that a non analyst can understand and participate in the specification process.

FLAMINCO Classification: model-based UI
Contribution: language, approach
Application: model-generated UI
Approach: abstract UI, conceptual patterns, OOA
Study:

1.7 Applying Model-Based Techniques to the Development of UIs for Mobile Computers (+) (Eisenstein, Vanderdonckt et al. 2001)

Mobile computing poses a series of unique challenges for user interface design and development: user interfaces must now accommodate the capabilities of various access devices and be suitable for different contexts of use, while preserving consistency and usability. We propose a set of techniques that will aid UI designers who are working in the domain of mobile computing. These techniques will allow designers to build UIs across several platforms, while respecting the unique constraints posed by each platform. In addition, these techniques will help designers to recognize and accommodate the unique contexts in which mobile computing occurs. Central to our approach is the development of a user-interface model that serves to isolate those features that are common to the various contexts of use, and to specify how the user interface should adjust when the context changes. We claim that without some abstract description of the UI, it is likely that the design and the development of user-interfaces for mobile computing will be very time consuming, error-prone or even doomed to failure.

FLAMINCO Classification: model-based mobile UI
Contribution: techniques
Application: mobile computing, multi-devices,
Approach: abstract UI modelling, context specification and mapping
Study: claim

1.8 SUPPLE: Automatically Generating User Interfaces (+) (Gajos and Weld 2004)

In order to give people ubiquitous access to software applications, device controllers, and Internet services, it will be necessary to automatically adapt user interfaces to the computational devices at hand (e.g., cell phones, PDAs, touch panels, etc.). While previous researchers have proposed

solutions to this problem, each has limitations. This paper proposes a novel solution based on treating interface adaptation as an optimization problem. When asked to render an interface on a specific device, our Supple system searches for the rendition that meets the device's constraints and minimizes the estimated effort for the user's expected interface actions. We make several contributions: 1) precisely defining the interface rendition problem, 2) demonstrating how user traces can be used to customize interface rendering to particular user's usage pattern, 3) presenting an efficient interface rendering algorithm, 4) performing experiments that demonstrate the utility of our approach.

FLAMINCO Classification: model-based mobile UI
Contribution: tool: SUPPLE (toolkit)
Idea: "treating interface adaptation as an optimization problem"
Application: multi-devices, interface adaptation
Approach: domain modelling, interface rendering algorithm
Study: experiments, demonstrations

1.9 Model-Based User Interface Design By Example And By Interview (-) (Frank and Foley 1993)

Model-based user interface design is centered around a description of application objects and operations at a level of abstraction higher than that of code, A good model can be used to support multiple interfaces, help separate interface and application, describe input sequencing in a simple way, check consistency and completeness of the interface, evaluate the interface's speed-of-use, generate context-specific help and assist in designing the interface. However, designers rarely use computer-supported application modelling today and prefer less formal approaches such as story boards of user interface prototypes. One reason is that available tools often use cryptic languages for the model specification. Another reason is that these tools force the designers to specify the application model before they can start working on the visual interface, which is their main area of expertise. We present the Interactive User Interface Design Environment (Interactive UIDE), a novel framework for concurrent development of the application model and the user interface which combines story-boarding and model-based interface design. We also present Albert. an intelligent component within this framework, which is able to infer an application model from a user interface and from an interview process with the designer.

FLAMINCO Classification: Model-based UI Design
Contribution: framework: interactive user interface design environment (interactive UIDE)
Application: UI design
Approach: story-boarding, model-based interface design
Study:

1.10 GUI Generation from Annotated Source Code (+) ?? (Jelinek and Slavik 2004)

Creating user interfaces is a common task in application development. It can become time and money consuming if the same application is to be run on more platforms with different restrictions and requirements. To reduce the development cost and time the user interface can be defined on an abstract level in the form of a task model. Explicit defining and maintaining the task model can complicate the development especially in its early stages when application prototypes are built. We present a way to define a user interface on an abstract level without explicit definition of the user interface module while keeping it transparent, thus reducing the work required to make functional application prototypes. Our approach is based on derivation of a user

interface module directly from an application source code enriched by abstract commands of user interaction to control the generation of the user interface.

FLAMINCO Classification: UI generation
Contribution: technique
Application: prototyping
Approach: derivate UI from code, abstract UI commands
Study:

1.11 Towards a User Interface Generation Approach Based on Object Oriented Design and Task Model (+) (Mahfoudhi, Abed et al. 2005)

This paper presents an approach of generating the user interface from the task model. Our work is situated in the course of approaches based on models. This approach called TOOD (Task Object Oriented Design) is based on a formal notation, which gives quantitative results that may be checked by designers and provide the possibility of performing mathematical verifications on the models. The modelling formalism is based on the joint use of the object approach and high level Petri nets. The TOOD method integrates different models (task model, user model, local model of the interface, abstract model of the interface, and model of the implementation) and their relations. An example, extracted from the air traffic control, is presented to illustrate TOOD methodology.

FLAMINCO Classification: Model-based UI
Contribution: approach: task object oriented design (TOOD)
Application: generating UI
Approach: object oriented design, task modelling, user modelling, UI modelling, realization modelling
Study: example

1.12 Tool Support for Designing Nomadic Applications (+) (Mori, Paternò et al. 2003)

Model-based approaches can be useful when designing nomadic applications, which can be accessed through multiple interaction platforms. Various models and levels of abstraction can be considered in such approaches. The lack of automatic tool support has been the main limitation to their use. We present a tool, TERESA, supporting top-down transformations from task models to abstract user interfaces and then to user interfaces for different types of interaction platforms (such as mobile phones or desktop systems). It allows designers to keep a unitary view of the design of a given nomadic application. Moreover, the tool provides support for obtaining effective interfaces for each type of platform available, taking into account the consequent differences in terms of tasks and their performance.

FLAMINCO Classification: model-based mobile UI
Contribution: tool: TERESA (authoring tool)
Application: multi-devices
Approach: top-down transformation: task models, abstract UI, platform UI
Study:

1.13 Generating User Interface Code in a Model Based User Interface Development Environment (da Silva, Griffiths et al. 2000)

Declarative models play an important role in most software design activities, by allowing designs to be constructed that selectively abstract over complex implementation details. In the user interface setting, Model-Based User Interface Development Environments (MB-U1DEs) provide a context within which declarative models can be constructed and related, as part of the interface design process. However, such declarative models are not usually directly executable, and may be difficult to relate to existing software components. It is therefore important that MB-UIDEs both fit in well with existing software architectures and standards, and provide an effective route from declarative interface specification to running user interfaces. This paper describes how user interface software is generated from declarative descriptions in the Teallach MB-UIDE. Distinctive features of Teallach include its open architecture, which connects directly to existing applications and widget sets, and the generation of executable interface applications in Java. This paper focuses on how Java programs, organized using the model-view-controller pattern (MVC), are generated from the task, domain and presentation models of Teallach.

FLAMINCO Classification: Model-based UI
Contribution: tool: Teallach MB-UIDE (model-based user interface development environment)
Application: UI generation, open architecture,
Approach: declarative interface specifications, model-view-controller pattern (MVC), task modelling, domain modelling, presentation modelling
Study:

1.14 Task Modelling for Context-Sensitive User Interfaces (Pribeanu, Limbourg et al. 2001)

With the explosion of devices, computing platforms, contextual conditions, user interfaces become more confronted to a need to be adapted to multiple configurations of the context of use. In the past, many techniques were developed to be performed to task analysis for obtaining a single user interface that is adapted for a single context of use. As this user interface may become unusable for other contexts of use, there emerges a need for modeling tasks which can be supported in multiple contexts of use, considering multiple combinations of the contextual conditions. For this purpose, the concept of unit task is exploited to identify a point where traditional task models can break into two parts: a context-insensitive part and a context-sensitive part. A widespread task model notation is then used to examine, discuss, and criticize possible configurations for modeling a context-sensitive task as a whole. One particular form is selected that attempts to achieve a clear separation of concern between the context-insensitive part, the context-sensitive part, and a new decision tree which branches to context-sensitive tasks, depending on contextual conditions. The questions of factoring out possible elements that are common across multiple contexts of use and representation of the resulting task model are discussed.

FLAMINCO Classification: model-based mobile UI
Contribution: technique
Idea: "context-insensitive and context sensitive parts of task models"
Application: multi-devices,
Approach: task modelling, context modelling,
Study:

1.15 Resource-Based User Interface Design (Eva 2002)

This thesis aims to bridge the gap between user interface programming and user-centred design for interactive systems.

Good user interface design has proved to be crucial to the success of an interactive system. The increasing difficulty for developing a good interactive system comes from different types of end users, different purposes and requirements, and more complicated interaction techniques. The design of a user interface occupies a significant amount of time. We need a systematic method of improving the design cycle for an effective and efficient user interface.

Model-based user interface approaches concern the use of models in user interface design. Generally speaking, model-based user interface development environments take the development of a user interface as a transformation from one model to another. Different perspectives, such as task modelling, dialogue design, and usability evaluation, emphasise different models. However, most model-based systems lack a systematic method to bridge the gap from model design to interface implementation. As a result, a lot of work still needs to be done by the interface designer.

We claim that user interface design should take account of how people interact with the interface in practice, instead of presuming that the user follows the sequence of actions the programmer planned. The resource model has provided an analysis of how and what resources – including goals, plans, state, action-effects, possibilities and history – are actually impacting the user in action-taking decisions. In this thesis, we investigate a systematic method of using the resources model to aid the design. The method (and then the tools) is aimed at taking a user oriented task description and transform it systematically into a specification of an interface.

The result of this investigation is a carefully designed prototype of a user interface development environment, implemented in Java and ready for use or further development.

FLAMINCO Classification: model-based UI
Contribution: tool, method
Idea: systematic method of using the resource model
Application: UI
Approach: Transform user oriented task description into a specification of an interface
Study: prototyping

1.16 Generating Form-Based User Interfaces for XML Vocabularies (-) (Kuo, Shih et al. 2005)

So far, many user interfaces for XML data (documents) have been constructed from scratch for specific XML vocabularies and applications. The tool support for user interfaces for XML data is inadequate. Forms-XML is an interactive component invoked by applications for generating user interfaces for prescribed XML vocabularies automatically. Based on a given XML schema, the component generates a hierarchy of HTML forms for users to interact with and update XML data compliant with the given schema. The user interface Forms-XML generates is very simple with an abundance of guidance and hints to the user, and can be customized by user interface designers as well as developers.

FLAMINCO Classification: XML-based UI
Contribution: Forms-XML
Idea:

Application: data-oriented UI
Approach: automatically generate user interfaces for prescribed XML vocabularies
Study:

1.17 Automatic Dialog Mask Generation for Device-Independent Web Applications (Book, Gruhn et al. 2006)

When building web applications for use on different devices, developers need to deal with a wide range of input/output capabilities that affect how users interact with the application: A dialog that can be completed in one step on a desktop client may have to be broken up into a number of steps on a small-screen mobile device. Since it is time-consuming to define all the possible dialog masks and dialog flow variants for different channels manually, it would be desirable to automate the adaptation of dialog masks and flows. To address this need, we introduce the DiaDef language for the abstract, device-independent definition of the widgets in a dialog, and the DiaGen framework that automatically breaks this abstract dialog definition down into sufficiently small dialog masks for the users' mobile devices and incorporates them into suitable micro dialog flows that are generated at run-time in order to be handled by our Dialog Control Framework.

FLAMINCO Classification: model-based mobile UI
Contribution: language: DiaDef, framework: DiaGen
Idea: automate the adoption of dialog masks and flows
Application: multi-devices, adoption
Approach: abstract language (DiaDef), device-independent widgets definitions, generate dialog masks and dialog flows (DiaGen)
Study:

1.18 Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances (Nichols, Rothrock et al. 2006)

Systems of connected appliances, such as home theaters and presentation rooms, are becoming commonplace in our homes and workplaces. These systems are often difficult to use, in part because users must determine how to split the tasks they wish to perform into sub-tasks for each appliance and then find the particular functions of each appliance to complete their sub-tasks. This paper describes Huddle, a new system that automatically generates task-based interfaces for a system of multiple appliances based on models of the content flow within the multi-appliance system.

FLAMINCO Classification: mode-based UI
Contribution: tool: Huddle
Idea:
Application: distributed systems,
Approach: automatically generate task-based interfaces
Study:

1.19 User Interface Conceptual Patterns (Molina, Meli´a et al. 2002)

User Interface Patterns are not sufficiently explored at the Conceptual phase. Work in area of User Interface patterns is predominantly done at design phase but not enough work is dedicated to analysis patterns. This paper shows different examples of abstract user interface patterns and explores the impact of such patterns in the software lifecycle. Conceptual User Interface Patterns

can be used for direct specification of device independent interfaces that can be refined using UI design patterns, or moreover, used to automatically obtain prototypes of the user interface specified in several devices.

FLAMINCO Classification: pattern-based UI
Contribution: Technique : Conceptuel User Interface Patterns
Idea:
Application: multi-devices, prototype UI
Approach: from conceptual UI patterns to specific UI (prototype, automatic)
Study:

1.20 Robust Content Creation with Form-Oriented User Interfaces (Draheim, Lutteroth et al. 2005)

In this paper we describe how content can be created in a way that ensures its integrity at all times, and how the user interface for such a content editing program can be modelled using the methodology of form-oriented analysis. The paper discusses aspects concerning the data that is being created, as well as aspects of the content editor itself. We show that technological features like typing, opaque identities and user transactions can facilitate the process of content creation as experienced by the user significantly, and that these features can be effectively incorporated when using the form-oriented analysis model.

FLAMINCO Classification: model-based form-based UI
Contribution: Methodology: form-oriented analysis
Idea:
Application: Content Editor UI
Approach: from models to content editor UI through form-oriented analysis
Study:

1.21 Application requirements for middleware for mobile and pervasive systems(Raatikainen, Christensen et al. 2002)

In this paper, we examine the requirements for future middleware to support mobile and pervasive applications and identify key research areas. We illustrate the research areas with requirements identified in two specific research projects concerning pervasive healthcare and home entertainment.

FLAMINCO Classification: middleware requirements
Contribution: requirements and key research areas
Idea:
Application: mobile and pervasive computing, middleware
Approach:
Study:

1.22 USIXML: A User Interface Description Language for Context-Sensitive User Interfaces (Limbourg, Vanderdonckt et al. 2004)

This paper presents USIXML (USer Interface eXtensible Markup Language), a User Interface Description Language aimed at de-scribing user interfaces with various levels of details and abstractions, depending on the context of use. USIXML supports a family of user interfaces such

as, but not limited to: device-independent, platform-independent, modality independent, and ultimately context-independent. This paper consequently details how context-sensitive user interfaces may be specified and produced from the USIXML specifications. USIXML allows specifying multiple models involved in user interface design such as: task, domain, presentation, dialog, and context of use, which is in turn decomposed into user, platform, and environment. These models are structured according to the four layers of the Cameleon framework: task & concepts, abstract user interface, concrete user interface, and final user interface. To support relationships between these models, a model for inter-model mapping is also introduced that cover forward and reverse engineering as well as translation from one context of use to another.

FLAMINCO Classification: model-based mobile UI
Contribution: language: USIMXML, framework: Cameleon
Idea:
Application: multi-devices, -platform, - modality, -context
Approach: modelling: task, domain, presentation, dialog context (user, platform, environment), inter-model mapping
Study:

1.23 A MDA-Compliant Environment for Developing User Interfaces of Information Systems (Vanderdonckt 2005)

To cope with the ever increasing diversity of markup languages, programming languages, tool kits and interface development environments, conceptual modeling of user interfaces could bring a framework for specifying, designing, and developing user interfaces at a level of abstraction that is higher than the level where code is merely manipulated. For this purpose, a complete environment is presented based on conceptual modeling of user interfaces of information systems structured around three axes: the models that characterize a user interface from the end user's viewpoint and the specification language that allows designers to specify such interfaces, the method for developing interfaces in forward, reverse, and lateral engineering based on these models, and a suite of tools that support designers in applying the method based on the models. This environment is compatible with the Model-Driven Architecture recommendations in the sense that all models adhere to the principle of separation of concerns and are based on model transformation between the MDA levels. The models and the transformations of these models are all expressed in UsiXML (User Interface eXtensible Markup Language) and maintained in a model repository that can be accessed by the suite of tools. Thanks to this environment, it is possible to quickly develop and deploy a wide array of user interfaces for different computing platforms, for different interaction modalities, for different markup and programming languages, and for various contexts of use.

FLAMINCO Classification: model-based mobile UI
Contribution: Environment: MDA-compliant
Idea:
Application: multi-devices
Approach: UsiXML, MDA
Study:

1.24 Towards an Interoperability Framework for Model-Driven Development of Software Systems (Elvesæter, Hahn et al. 2005)

This paper presents an interoperability framework for enterprise applications and software systems. The framework provides a foundation for model-driven development of software systems supporting the business interoperability needs of an enterprise. This is achieved through a set of reference models that addresses interoperability issues for conceptual integration, technical integration and applicative integration of software systems.

FLAMINCO Classification: model-based interoperability
Contribution: Framework
Idea:
Application: interoperable systems
Approach: reference models: conceptual, technical, and applicative integration
Study:

1.25 Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing (Chung, Hong et al. 2004)

Design patterns are a format for capturing and sharing design knowledge. In this paper, we look at a new domain for design patterns, namely ubiquitous computing. The overall goal of this work is to aid practice by speeding up the diffusion of new interaction techniques and evaluation results from researchers, presenting the information in a form more usable to practicing designers. Towards this end, we have developed an initial and emerging pattern language for ubiquitous computing, consisting of 45 pre-patterns describing application genres, physical-virtual spaces, interaction and systems techniques for managing privacy, and techniques for fluid interactions. We evaluated the effectiveness of our pre-patterns with 16 pairs of designers in helping them design location-enhanced applications. We observed that our pre-patterns helped new and experienced designers unfamiliar with ubiquitous computing in generating and communicating ideas, and in avoiding design problems early in the design process.

FLAMINCO Classification: design-patterns for ubiquitous computing
Contribution: language: initial and emerging pattern-language for ubiquitous computing, 45-pre-patterns
Idea:
Application: interactive design, ubiquitous systems
Approach: pattern-based design and evaluation
Study: effectiveness evaluation

1.26 Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces (Lin and Landay 2002)

People often use a variety of computing devices, such as PCs, PDAs, and cell phones, to access the same information. The user interface to this information needs to be different for each device, due to different input and output constraints. Currently, designers designing such multi-device user interfaces either have to design a UI separately for each device, which is time consuming, or use a program to automatically generate interfaces, which often results in interfaces that are awkward. We are creating a system called Damask to better support multi-device UI design. With Damask, the designer will design a UI for one device, by sketching the design and by specifying which design patterns the interface uses. The patterns will help Damask generate user interfaces optimized for the other target devices. The generated interfaces will be of sufficient quality so that

it will be more convenient to use Damask than to design each of the other interfaces separately, and the ease with which designers will be able to create designs will encourage them to engage in iterative design.

Damask is a pattern-based design tool for early-stage design and prototyping of multi-device user interfaces. UI designers can use it to create storyboard prototypes. It contains a catalog of patterns from which a designer selects patterns as they build their design. The designer then customizes the patterns to match the problem domain. Each design pattern includes many different examples illustrating how to apply the pattern. As well, each 35 pattern provides a general solution for each device (PC, cell phone, PDA) supported. When a storyboard sketch for one interface has been completed the system can create an equivalent UI design for other devices. Damask automatically updates the examples section of a pattern when new designs are created. Designers can also create new patterns to add to the repository but the solution must be illustrated using Denim's sketching

FLAMINCO Classification: pattern-based mobile UI
Contribution: tool: Damask (pattern-based design tool)
Idea:
Application: multi-devices, prototyping
Approach: sketching, specifying, design-patterns, prototyping
Study:

1.27 Form-Based User Interface - The Architectural Patterns (A Pattern Language) (Coldewey and Krüge 1997)

They claim that despite all the benefits of object-oriented user interfaces, there are still domains that call for a form-based user interface. Business information systems that support fast processing of few, well-defined use cases are typical examples. They present a pattern language to support the development of software architecture for business information systems with form-based UI.

The paper is part of a larger effort to collect patterns for business information systems, currently pursued by the ARCUS team.

FLAMINCO Classification: form-based UI
Contribution: language: pattern-language
Idea:
Application: form-based UI
Approach: pattern-language
Study: examples

1.28 Storyboarding Form-Based Interfaces (Draheim and Weber 2003)

This paper identifies two-staged interaction as the abstract concept behind form-based interfaces. From this we derive form storyboarding, a method for eliciting, specifying and communicating functional requirements for applications with form-based interfaces. The method encompasses a visual language for the documents which have to be created, and a set of proposals for activities. The visual language is based on bipartite state transition diagrams.

FLAMINCO Classification:
Contribution:
Idea:

Application:
Approach:
Study:

1.29 Managing UI Pattern Collections (Deng, Kemp et al. 2005)

A large number of user interface (UI) pattern collections have been developed by different researchers. This paper discusses the main requirements for a tool to be used by researchers and user interface designers that can manage a repository of possibly disparate pattern collections. An analysis of the main requirements and specifications for such a tool has been carried out. Pattern tools have been surveyed to identify the functionality they provide. A framework for a UI pattern management tool has been developed in the light of the analysis and the survey.

FLAMINCO Classification: form-based UI
Contribution: method: form storyboarding
Idea:
Application:
Approach:
Study: survey of pattern-tools

1.30 Computer Assisted Sketching for the Early Stages of User Interface Design (Coyette and Vanderdonckt 2005)

Sketching activities are widely adopted during early design phases of user interface development to convey informal specifications of the interface presentation and dialog. Designers or even end users can sketch some or all of the future interface they want. With the ever increasing availability of different computing platforms, a need arises to continuously support sketching across these platforms with their various programming languages, interface development environments and operating systems. To address needs along these dimensions, which pose new challenges to user interface sketching tools, SketchiXML is a multi-platform multi-agent interactive application that enable designers and end users to sketch user interfaces with different levels of details and support for different contexts of use. The results of the sketching are then analyzed to produce interface specifications independently of any context, including user and platform. These specifications are exploited to progressively produce one or several interfaces, for one or many users, platforms, and environments.

FLAMINCO Classification: sketch-based UI
Contribution: tool: SketchiXML
Idea:
Application: multi-devices
Approach: sketching
Study:

1.31 Interaction Patterns in User Interfaces (van Welie and Trætteberg 2000)

This paper discusses and presents interaction patterns in user interfaces. These patterns are focused on solutions to problems end-users have when interacting with systems. The patterns take an end-user perspective which leads to a format where usability is the essential design quality. The format is discussed and presented along with twenty patterns that have been written in that format.

FLAMINCO Classification: pattern-based UI
Contribution: UI patterns
Idea:
Application:
Approach:
Study:

1.32 Using Design Patterns and Layers to Support the Early-Stage Design and Prototyping of Cross-Device User Interfaces (Lin 2005)

People often use a variety of computing devices, such as pcs, pdas, and cell phones, to access the same information. The user interface to this information needs to be different for each device, due to the different input and output constraints of each device. Currently designers designing such cross-device user interfaces either have to design a ui separately for each device, which is time consuming, or use a program to automatically generate interfaces, which often result in interfaces that are awkward. Each method also discourages iterative design, considered critical for creating good user interfaces.

I have created a system called Damask to support the early-stage design of user interfaces targeted at multiple devices. Within Damask, designers use layers to specify which parts of a user interface is common across all devices and which are specific to one device. They use design patterns to specify higher-level concepts within a user interface. Design patterns in Damask include pre-built ui fragments that are already optimized for each device. Designers can instantiate patterns in their designs and then customize the instances to fit their particular requirements.

Through a study performed with twelve professional web designers, we have found that, in the early stages of design, designers using patterns and layers in Damask create cross-device user interfaces that are as good as or better than those created without patterns and layers, in less time.

FLAMINCO Classification: pattern-based mobile UI
Contribution: tool: Damask (pattern-based design tool)
Idea:
Application: multi-devices, prototyping
Approach: sketching, specifying, design-patterns, prototyping
Study: case study

1.33 Automatic User Interface Generation from Declarative Models (Schlungbaum and Elwert 1996)

Automatic user interface generation is a widely discussed topic in the research community. In recent years several approaches have been developed to support this kind of generation. There is a need to summarise this. This article should provide a basis for a founded discussion in this direction. The article gives an overview about model-based user interface software tools. The special attention is paid to the declarative models. The process of user interface generation is highlighted on a basis of a categorisation. The main section contains ideas of TADEUS about automatic user interface generation explained by an example.

FLAMINCO Classification: model-based UI

Contribution: survey: overview of model-based user interface software tools
Idea:
Application:
Approach: user interface generation through declarative models
Study: examples, survey/review

1.34 User Interface Declarative Models and Development Environments: A Survey (da Silva 2000)

Model-Based User Interface Development Environments (MB-UIDEs) provide a context within which user interface declarative models can be constructed and related, as part of the user interface design process. This paper provides a review of MB-UIDE technologies. A framework for describing the elements of a MB-UIDE is presented. A representative collection of 14 MB-UIDEs are selected, described in terms of the framework, compared and analysed from the information available in the literature. The framework can be used as an introduction to the MB-UIDE technology since it relates and provides a description for the terms used in MB-UIDE papers.

FLAMINCO Classification: Model-based UI
Contribution: survey: 14 MB-UIDE technologies, framework: MB-UIDE
Idea:
Application:
Approach: MB-UIDE provide a context within which user interface declarative models can be constructed and related, as part of the user interface design process
Study: Survey/review

1.35 Tool-Supported Interpreter-Based User Interface Architecture for Ubiquitous Computing (Braubach, Pokahr et al. 2002)

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

FLAMINCO Classification: Model-based mobile UI
Contribution: requirements and needs, survey MB-UIDE, Architecture: Arch model
Idea: Arch model supports the special UbiComp requirements
Application: multi-devices, ubiquitous computing
Approach: layered architecture support different implementation techniques and UI modalities
Study: implementation demo (Vesuf), usability validation

1.36 User Interface Modeling in UML*i* (da Silva and Paton 2003)

Although user interfaces represent an essential part of software systems, the Unified Modeling Language seems to have been developed with little specific attention given to user interface issues. You can use standard UML to model important aspects of user interfaces, but this often results in unwieldy and unnatural representations.

FLAMINCO Classification: Model-based UI
Contribution: Language: UMLi
Idea:
Application: UI
Approach: UML-based modeling
Study:

1.37 Multi-devices "Multiple" user interfaces: development models and research opportunities (Seffah, Forbrig et al. 2004)

Today, Internet-based appliances can allow a user to interact with the server-side services and information using different kinds of computing platforms including traditional office desktops, palmtops, as well as a large variety of wireless devices including mobile telephones, Personal Digital Assistants, and Pocket Computers. This technological context imposes new challenges in user interface software engineering, as it must run on different computing platforms accommodating the capabilities of various devices and the different contexts of use. Challenges are triggered also because of the universal access requirements for a diversity of users. The existing approaches of designing a single user interface using one computing platform do not adequately address the challenges of diversity, cross-platform consistency, universal accessibility and integration. Therefore, there is an urgent need for a new integrative framework for modeling, designing and evaluating multi-device user interfaces for the emerging generation of interactive systems.

This paper begins by describing a set of constraints and characteristics intrinsic to multi-device user interfaces, and then by examining the impacts of these constraints on the specification, design and validation processes. Then, it discusses the research opportunities in important topics relevant to multi-device user interface development, including task and model-based, pattern-driven and device-independent development. We will highlight how research in these topics can contribute to the emergence of an integrative framework for Multiple-User Interface design and validation.

FLAMINCO Classification: mobile (multi-device) UI design and validation
Contribution: research opportunities: task and model-based, pattern-driven and device-independent development
Idea: Vision: an integrative framework for Multiple-User Interface design and validation
Application: multi-devices
Approach:
Study:

1.38 Guidelines for Handheld Mobile Device Interface Design (Gong and Tarasewich 2004)

While there has been much successful work in developing rules to guide the design and implementation of interfaces for desktop machines and their applications, the design of mobile device interfaces is still relatively unexplored and unproven. This paper discusses the characteristics and limitations of current mobile device interfaces, especially compared to the

desktop environment. Using existing interface guidelines as a starting point, a set of practical design guidelines for mobile device interfaces is proposed.

FLAMINCO Classification: pattern-based UI design
Contribution: guidelines for mobile UI
Idea:
Application: mobile, multi-device UIs
Approach:
Study:

This paper discusses the characteristics and limitations of current mobile device interface. Using existing interface guidelines as a starting point, a set of practical design guidelines for mobile device interfaces is proposed:

- Enable frequent users to use shortcuts
- Offer informative feedback
- Design dialogs to yield closure
- Support internal locus of control
- Consistency
- Reversal of actions
- Error prevention and simple error handling
- Reduce short-term memory load
- Design for multiple and dynamic contexts
- Design for small devices
- Design for limited and split attention
- Design for speed and recovery
- Design for "top-down" interaction
- Allow for personalization
- Design for enjoyment

1.39 Towards A Better Model Based User Interface Development Environment: A Comprehensive Survey (Gomaa, Salah et al. 2005)

The introduction of user devices with built-in computer programs has introduced a number of challenges to the design of user interfaces. Automating the management and generation of interfaces greatly improves their quality and maintainability and significantly reduces the cost of development. Model-based user interface development environments (MBUIDEs) are tools that help designers with building interfaces through automating the generation of interfaces using high-level declarative models.

In this paper, surveyed different interface generation techniques and built a framework to compare and analyze their suitability to handle the changes imposed by universal usability. The paper points out limitation with current techniques and proposes the use of a multi-model conceptual layer that will be used as a management system to control the specification, creation, and manipulation of the interfaces. We claim this framework will be able to overcome many of the limitations of today's techniques in facing the above mentioned challenges.

FLAMINCO Classification: Model-based mobile UI
Contribution: survey: UI generation techniques, framework: MB-UIDE
Idea: a multi-model conceptual layer to control the specification, creation, and manipulation of UI

Application: multi-devices
Approach: multi-model conceptual layer
Study:

1.40 A Framework for Generating Spatial Configurations in User Interfaces (Fischer 1998)

This paper describes an approach to the problem of designing and implementing visual presentations in direct-manipulative user interfaces. Such presentations are often complex and their construction requires in-depth design knowledge. A framework is proposed that includes declarative models and inference mechanisms, aimed to significantly reduce the demands on the interface developer. Models of application characteristics form the input for a generation system which is parameterised by the interface developer. The inferred layout is produced both as a declarative model and executable code, which, integrated with the rest of the application, produces the presentation at application run-time.

FLAMINCO Classification: Model-based UI
Contribution: framework: generate spatial configuration in UI
Idea:
Application: visual presentation in direct-manipulative UI
Approach: declarative models and inference mechanisms,
Study:

1.41 Combining Compound Conceptual User Interface Components with Modelling Patterns - a Promising Direction for Model-based Cross-platform User Interface Development (Nilsson 2002)

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

FLAMINCO Classification: Model-based mobile UI
Contribution: Technique: compound conceptual UI component. Language: requirements, tools: requirements
Idea:
Application: cross-platform UI
Approach: pattern-based abstract compound UI components, platform dependent mapping rules
Study: example

1.42 Model-based user interface adaptation (Nilsson, Floch et al. 2006)

Exploiting contextual information in user interfaces (UIs) on mobile equipment is an important means for enhancing the user experience on such devices. Implementing flexible and adaptive UIs that are multitasking and possibly exploiting multiple modalities requires expensive application-

specific solutions, and reuse is difficult or impossible. To make it viable to implement adaptive UIs for a broader range of applications, there is both a need for new architecture and middleware, and ways of constructing applications. In this paper, we show how a combination of a patterns-based modelling language using compound UI components and mapping rules as building blocks, and a generic adaptive architecture based on components with ports and utility functions for finding the optimal configuration in a given situation facilitates implementation of applications with adaptive UIs. After an introduction discussing special needs for mobile UIs and related work, we briefly present our modelling approach, and the adaptive middleware architecture. With this as a background, we investigate how the combination of these facilitates model-based UI adaptation. Finally, we present some more general principles, illustrated by two examples, for how model-based approaches may be used for visual adaptation of UIs.

FLAMINCO Classification: Model-based mobile UI
Contribution: approach: modeling, architecture: adaptive middleware
Idea: new architecture, new middleware and new ways of constructing applications
Application: flexible and adaptive UI
Approach: patterns-based modeling language, compound UI components and mapping rules, generic adaptive architecture
Study: examples

1.43 Using a Patterns-based Modelling Language and a Model-based Adaptation Architecture to Facilitate Adaptive User Interfaces (Nilsson, Floch et al. 2006)

To design usable mobile applications, exploiting context changes is of vital importance. The rapid context changes in a mobile setting cause the need for flexible and adaptive user interfaces that are multitasking and possibly exploiting multiple modalities. Implementing adaptive user interfaces requires expensive application-specific solutions. Reuse of this type of solutions is difficult or impossible. To make it viable to implement adaptive user interfaces for a broader range of applications, there is both a need for new architecture and middleware, and ways of constructing applications. In this paper, we show how a combination of a patterns-based modelling language using compound user interface components and mapping rules as building blocks, and a generic adaptive architecture based on components with ports and utility functions for finding the optimal configuration in a given situation, facilitates implementation of applications with adaptive user interfaces. First we briefly present our modelling approach, and the adaptive architecture including the generic middleware exploiting architecture models at runtime. With this as a background we show how the presented modelling approach may be combined with the adaptive architecture to facilitate model-based user interface adaptation. Finally, we compare our approach with other approaches for realizing adaptive user interfaces, and we give some conclusions and directions for future research.

FLAMINCO Classification: Model-based, pattern-based mobile UI
Contribution:
Idea:
Application:
Approach:
Study:

1.44 The DKC Model: An Application Model for Collaborative Work (Iverson 2004)

For many years, researchers and software developers have been seeking to develop systems and applications to enable efficient and effective group work and organizational memory. The systems proposed and developed have in many respects had little impact on the effectiveness of group activities outside the laboratory. Other researchers have identified many of the challenges that groupware systems face, but these insights have done little to structure subsequent research.

We suggest that these difficulties are primarily due to an operating system model and a model of application development that has significantly restricted the ability of users to properly manage their own data and work products much less share them with others. Moreover, it is nearly impossible to effectively integrate collaborative activities with natural practices of work and communication. Instead, we propose an alternative system architecture, the DKC model, that places HCI, knowledge representation and management, and distributed, hypertext operating systems in a coordinated structure. We clearly delineate the roles of each of these aspects within the whole, collaborative environment. By adopting this model, we suggest that researchers and developers of both single-user and collaborative systems will be able to design effective, multi-platform, multi-application, and multi-workplace collaborative environments that may finally have some impact beyond the laboratory.

FLAMINCO Classification: Model-based UI
Contribution: system architecture: DKC Model
Idea:
Application: collaborative systems, multi-devices
Approach:
Study:

1.45 Design Patterns for Ubiquitous Computing (Landay and Borriello 2003)

FLAMINCO Classification: Pattern-based UI
Contribution: collection of design-patterns for ubiquitous computing
Idea:
Application:
Approach:
Study:

1.46 Adaptation in automated user-interface design (Eisenstein and Puerta 2000)

Design problems involve issues of stylistic preference and flexible standards of success; human designers often proceed by intuition and are unaware of following any strict rule-based procedures. These features make design tasks especially difficult to automate. Adaptation is proposed as a means to overcome these challenges. We describe a system that applies an adaptive algorithm to automated user interface design within the framework of the MOBI-D (Model-Based Interface Designer) interface development environment. Preliminary experiments indicate that adaptation improves the performance of the automated user interface design system.

FLAMINCO Classification: model-based UI
Contribution: Technique: adaptive algorithm to automated UI design
Idea:
Application: model-based UI design

Approach: MOBI-D (Model-Based Interface designer) interface development environment.
Study: preliminary experiments

"Several systems have attempted to automatically generate user interfaces in a model-based environment. UIDE [l] is one of the first model-based user interface design systems, and it automatically selects interactors on the basis of data type-one of the discriminants we consider here.

MECANO [5] generates form-based interfaces automatically, and performs interactor selection by considering several discriminants, including type, cardinality, and the number of allowed values.

TRIDENT [12] introduces a comprehensive decision tree that takes into account a broad set of discriminants. These systems represent progress towards automated user interface design, but they do not incorporate adaptation.

Quinlan has developed ID3 [lo], an algorithm for decision tree induction. ID3 is based on information theory, and it is intended for batch learning problems that start from scratch with a large number of examples. Our approach favors theory-refinement and sensitivity, so ID3 was not an acceptable choice as an induction algorithm. Maclin and Shavlik present a theory-refinement algorithm that utilizes user advice to minimize the amount of training time necessary [4], but their algorithm is for connectionist learning systems, which we have ruled out because of their poor human-comprehensibility. We were unable to locate a sensitive theory-refinement algorithm for decision tree induction in the literature; ours may be one of the first.

Inference Bear [2] is a programming-by-demonstration application that infers design specifics from user-generated snapshots. Inference Bear uses adaptation to generate a custom user interface by observing the behavior of the interface designer. But whereas Inference Bear is designed to infer application-speczjic design knowledge, our approach seeks to induce and refine general principles of user interface design. Every time a designer begins to design an interface with Inference Bear, it starts afresh. In contrast, the adaptive version of MOBI-D is smarter every time it runs."

1.47 Generating mobile device user interfaces for diagram-based modelling tools (Zhao, Grundy et al. 2006)

Mobile display devices such as phones and PDAs have become very widely available and used. However, most content on these devices is limited to text, static images and motion video. Displaying and interacting with dynamic diagrammatic content on such devices is difficult, as is engineering applications to implement such functionality. We describe a set of plug-in components for a meta-diagramming tool that enable a diagram type to be visualized and interacted with on mobile devices. Key features of our approach include generating diagram content from an existing meta-tool, run-time user configuration of diagram appearance and navigation, and multi-level, zoomable diagrams and diagram content. We describe our experiences prototyping, using and evaluating this new mobile device diagramming technology

FLAMINCO Classification:
Contribution:
Idea:
Application:
Approach:
Study:

1.48 Conceptual Modelling for Database User Interfaces (Cooper, McKirdy et al. 2000)

Model-based user interface development environments show promise for improving the speed of production and quality of user interfaces. Such systems usually have separate description of domain, task and presentation structure. The Teallach system applies model based techniques to the important area of database interfaces, which increases the importance of domain information. This exists in the form of a schema and can be captured in a high-level format, so that the developer need not build a domain description arbitrarily. This paper describes such a Domain Model, how it is captured and how it contributes to the systematic development of a user interface.

FLAMINCO Classification: model-based UI
Contribution: Tool: Teallach, Language: Domain Model
Idea: "Capture Domain Model from database schema
Application: UI
Approach: capture domain model from database schema, use the domain model in UI development
Study:

1.49 Model-based User Interface Design (Trætteberg 2002)

This work is about supporting user interface design by means of explicit design representations, in particular models.

We take as a starting point two different development traditions: the formal, analytic, topdown engineering approach and the informal, synthetic, bottom-up designer approach. Both are based on specific design representations tailored to the respective approaches, and are found to have strengths and weaknesses. We conclude that different representations should be used during user interface design, based on their specific qualities and the needs of the design process.

To better understand the use of design representations a framework for classifying them is developed. A design representation may be classified along three dimensions: the perspective (problem- or solution-oriented) of the representation, the granularity of the objects described and the degree of formality of the representation and its language. Any design approach must provide representation languages that cover the whole classification space to be considered complete. In addition, the transitions between different representations within the representation space must be supported, like moving between task-based and interaction-oriented representations or up and down a hierarchic model. Movements between representations with different degrees of formality are particularly important when combining user-centered design with a model-based approach.

The design representation classification framework has guided the development of diagram-based modelling languages for the three main perspectives of user interface design, tasks, abstract dialogue and concrete interaction. The framework has also been used for evaluating the languages. A set-based conceptual modelling language is used for domain modeling within all these perspectives. The task modelling language is designed as a hybrid of floworiented process languages and traditional hierarchical sequence-oriented task languages.

Key features are tight integration with the domain modelling language, expressive and flexible notation and support for classification of task structures. The language for modeling abstract dialogue is based on the interactor abstraction for expressing composition and information flow, and the Statecharts language for activation and sequencing. Parameterized interactors are supported, to provide means of expressing generic and reusable dialogue structures. Modelling of concrete interaction is supported by a combination of the dialogue and domain modelling

languages, where the former captures the functionality and behavior and the latter covers concepts that are specific for the chosen interaction style.

The use of the languages in design is demonstrated in a case study, where models for tasks, dialogue and concrete interaction are developed. The case study shows that the languages support movements along the perspective, granularity and formality dimensions.

FLAMINCO Classification: model-based UI
Contribution: Framework: classifying design representation, Language: diagram-based modeling (user interface design, tasks, abstract and concrete interaction)
Idea:
Application: UI design
Approach: combining user-centered design and model-based approach
Study: case study

1.50 Declarative interface models for user interface construction tools: the MASTERMIND approach (Szekely, Sukaviriya et al. 1996)

Currently, building a user interface involves creating a large procedural program. Model- based programming provides an alternative new paradigm. In the model-based paradigm, developers create a declarative model that describes the tasks that users are expected to accomplish with a system, the functional capabilities of a system, the style and requirements of the interface, the characteristics and preferences of the users, and the I/O techniques supported by the delivery platform. Based on the model, a much smaller procedural program then determines the behavior of the system.

There are several advantages to this approach. The declarative model is a common representation that tools can reason about, enabling the construction of tools that automate various aspects of interface design, that assist system builders in the creation of the model, that automatically provide context sensitive help and other run-time assistance to users. The common model also allows the tools that operate on it to cooperate. Because all components of the system share the knowledge in the model, this promotes interface consistency within and across systems and reusability in the construction of new interfaces. The declarative nature of the model allows system builders to more easily understand and extend systems.

This paper describes the modeling language of MASTERMIND, a model-based user interface development environment.

1.51 Retrospective and Challenges for Model-Based Interface Development (Szekely 1996)

Research on model-based user interface development tools is about 10 years old. Many approaches and prototype systems have been investigated in universities and research laboratories around the world. This paper proposes a generic architecture for these tools, reviews the different approaches in light of this architecture, and discusses their progress towards the goals of increasing the quality and reducing the cost of developing interfaces. The paper closes with a discussion of challenges for future model-based development tools.

1.52 Adapting to Mobile Contexts with User-Interface Modeling (Eisenstein, Vanderdonckt et al. 2000)

Mobile computing offers the possibility of dramatically expanding the versatility of computers, by bringing them off the desktop and into new and unique contexts. However, this newfound

versatility poses difficult challenges for user-interface designers. We propose three model-based techniques that will aid UI designers who are working in the domain of mobile computing. These techniques will allow designers to build UIs across several platforms, while respecting the unique constraints posed by each platform. In addition, these techniques will help designers to recognize and accommodate the unique contexts in which mobile computing occurs. All three techniques depend on the development of a user-interface model which serves to isolate those features that are common to the various contexts of use, and to specify how the user-interface should adjust when the context changes. User-interface models allow automatic and automated tool support that will enable UI designers to overcome the challenges posed by mobile computing.

1.53 Dygimes - Dynamically Generating Intergaces for Mobile Computing Devices and Embedded Systems (Coninx, Luyten et al. 2003)

Constructing multi-device interfaces still presents major challenges, despite all efforts of the industry and several academic initiatives to develop usable solutions. One approach which is finding its way into general use is XML-based User Interface descriptions to generate suitable User Interfaces for embedded systems and mobile computing devices. Another important solution is Model-based User Interface design, which evolved into a very suitable but academic approach for designing interfaces. They introduce a framework, Dygimes, which uses XML-based User Interface descriptions in combination with selected models, to generate User Interfaces for different kinds of devices at run-time. With this framework task specifications are combined with XML-based User Interface building blocks to generate User Interfaces that can adapt to the context of use. The design of the User Interfaces and the implementation of the application code can be separated, with smooth integration of the functionality and the User interface is supported. The resulting interface is location independent: it can migrate over devices while invoking functionality using standard protocols.

1.54 Past, present, and future of user interface software tools (Myers, Hudson et al. 2000)

A user interface software tool helps developers to design and implement the user interface. Research on past tools has had enormous impact on today's developers – virtually all applications today are built using some form of user interface tool. In their article, they consider cases of both success and failure in past user interface tools. From these cases they extract a set of themes which can serve as lessons for future work. Using these themes, past tools can be characterized by what aspects of the user interface they addressed, their threshold and ceiling, what path of least resistance they offer, how predictable they are to use, and whether they addressed a target that became irrelevant. They believe the lessons of these past themes are particularly important now, because increasingly rapid technological changes are likely to significantly change user interfaces. We are at the dawn of an era where user interfaces are about to break out of the "desktop" box where they have been stuck for the past 15 years. They think that the next millenium will open with an increasing diversity of user interfaces on an increasing diversity of computerized devices. These devices include hand-held personal digital assistants (PDAs), cell phones, pagers, computerized pens, computerized notepads, and various kinds of desk and wall-size computers, as well as devices in everyday objects (such as mounted on refrigerators, or even embedded in truck tires).

1.55 Model-Generated Workplaces: An Interoperability Approach (Rolfsen, Boell et al. 2007)

This paper presents an approach to using model-generated workplaces (MGWP) in tandem with service-oriented architectures in order to meet interoperability needs in a corporate environment. More specifically, the interoperability study and research undertaken is based on a use case scenario related to the process of product portfolio management (PPM) in a large enterprise of the Telecommunication sector, INTRACOM Telecom. Starting from enterprise modeling constructs,

Web-based workplaces are created that offer navigation and work views to the end user, supporting all his operational tasks as depicted in the enterprise model. Interaction with enterprise information repositories is facilitated via an underlying service-oriented architecture.

References

1. Book, M., V. Gruhn, et al. (2006). Automatic dialog mask generation for device-independent web applications. the 6th international conference on Web engineering, Palo Alto, California, USA.
2. Braubach, L., A. Pokahr, et al. (2002). Tool-Supported Interpreter-Based User Interface Architecture for Ubiquitous Computing. Interactive Systems - Design, Specification, and Verification. Q. L. P. Forbrig, B. Urban, J. Vanderdonckt, Springer Heidelberg: 89 -103.
3. Chung, E. S., J. I. Hong, et al. (2004). Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing. DIS2004, Cambridge, Massachusetts, USA, ACM.
4. Clerckx, T., K. Luyten, et al. (2004). Generating Context-Sensitive Multiple Device Interfaces From Design. the Fifth International Conference on Computer-Aided Design of User Interfaces. CADUI'2004 long paper track, Funchal, Isle of Madeira.
5. Coldewey, J. and I. Krüge (1997). Form-Based User Interface The Architectural Patterns A Pattern Language. Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, Germany,.
6. Coninx, K., K. Luyten, et al. (2003). Dygimes: Dynamically Generating Interfaces for Mobile Computing Devices and Embedded Systems. Human-Computer Interaction with Mobile Devices and Services (Mobile HCI).
7. Cooper, R., J. McKirdy, et al. (2000). Conceptual Modelling for Database User Interfaces. the Fifth Working Conference on Visual Database Systems: Advances in Visual Information Management, Kluwer, B.V. Deventer, The Netherlands, The Netherlands.
8. Coyette, A. and J. Vanderdonckt (2005). Computer Assisted Sketching for the Early Stages of User Interface Design. 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2005, Rome.
9. Crowle, S. and L. Hole (2003). ISML: An Interface Specification Meta-Language. tenth International Workshop on the Design, Specification and Verification of Interactive Systems. DSV-IS'2003, Funchal, Madeira.
10. da Silva, P. P. (2000). User Interface Declarative Models and Development Environments: A Survey. Interactive Systems: Design, Specification, and Verification (7th International Workshop DSV-IS), Limerick, Ireland, Springer.
11. da Silva, P. P., T. Griffiths, et al. (2000). Generating user interface code in a model based user interface development environment. the working conference on Advanced visual interfaces, Palermo, Italy.
12. da Silva, P. P. and N. W. Paton (2003). "User Interface Modeling in UMLi." EEE Software **20**(4): 62-69.
13. Deng, J., E. Kemp, et al. (2005). Managing UI Pattern Collections. the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural, Auckland, New Zealand.
14. Draheim, D., C. Lutteroth, et al. (2005). Robust Content Creation with Form-Oriented User Interfaces. the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural, Auckland, New Zealand.
15. Draheim, D. and G. Weber (2003). Storyboarding Form-Based Interfaces. Human-Computer Interaction -- INTERACT'03, IOS Press, (c) IFIP.
16. Eisenstein, J. and A. Puerta (2000). Adaptation in automated user-interface design. the 5th international conference on Intelligent user interfaces, New Orleans, Louisiana, United States, ACM Press New York, NY, USA.

17. Eisenstein, J., J. Vanderdonckt, et al. (2000). Adapting to Mobile Contexts with User-Interface Modeling. Workshop on Mobile Computing Systems and Applications, Monterey, CA, IEEE Press.
18. Eisenstein, J., J. Vanderdonckt, et al. (2001). Applying Model-Based Techniques to the Development of UIs for Mobile Computers. the 6th international conference on Intelligent user interfaces, Santa Fe, New Mexico, United States.
19. Elvesæter, B., A. Hahn, et al. (2005). Towards an Interoperability Framework for Model-Driven Development of Software Systems. the 1st International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA 2005), Geneva, Switzerland.
20. Eva, Y. h. C. (2002). Resource-Based User Interface Design. Department of Computer Science, The University of York.
21. Fischer, M. (1998). A Framework for Generating Spatial Configurations in User Interfaces. Design, Specification and Verification of Interactive Systems.
22. Frank, M. R. and J. D. Foley (1993). Model-Based User Interface Design By Example And By Interview. ACM Symposium on User Interface Software and Technology: 129-137.
23. Gajos, K., D. Christianson, et al. (2005). Fast and Robust Interface Generation for Ubiquitous Applications. Ubicomp'05, Tokyo, Japan.
24. Gajos, K. and D. S. Weld (2004). SUPPLE: Automatically Generating User Interfaces. the 9th international conference on Intelligent user interfaces, Funchal, Madeira, Portugal.
25. Gomaa, M., A. Salah, et al. (2005). Towards A Better Model Based User Interface Development Environment: A Comprehensive Survey. MICS 2005.
26. Gong, J. and P. Tarasewich (2004). Guidelines for Handheld Mobile Device Interface Design. DSI 2004 Annual Meeting.
27. Hanumansetty, R. G. (2004). Model Based Approach For Context Aware And Adaptive User Interface Generation.
28. Iverson, L. (2004). The DKC Model: An Application Model for Collaborative Work.
29. Jelinek, J. and P. Slavik (2004). GUI Generation from Annotated Source Code. the 3rd annual conference on Task models and diagrams, Prague, Czech Republic.
30. Kuo, Y. S., N. C. Shih, et al. (2005). Generating Form-Based User Interfaces for XML Vocabularies. the 2005 ACM symposium on Document engineering, Bristol, United Kingdom.
31. Landay, J. A. and G. Borriello (2003). "Design Patterns for Ubiquitous Computing." Computer, EEE Computer Society Press Los Alamitos, CA, USA **36**(8): 93 - 95.
32. Limbourg, Q., J. Vanderdonckt, et al. (2004). USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. the 3rd annual conference on Task models and diagrams, Prague, Czech Republic.
33. Lin, J. (2005). Using Design Patterns and Layers to Support the Early-Stage Design and Prototyping of Cross-Device User Interfaces. UNIVERSITY OF CALIFORNIA, BERKELEY.
34. Lin, J. and J. Landay (2002). Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing), San Francisco, CA,.
35. Mahfoudhi, A., M. Abed, et al. (2005). Towards a User Interface Generation Approach Based on Object Oriented Design and Task Model. the 4th international workshop on Task models and diagrams, Gdansk, Poland.
36. Molina, P. J., S. Meli´a, et al. (2002). User Interface Conceptual Patterns. Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, Springer-Verlag.
37. Molina, P. J., S. Meliá, et al. (2002). Just-UI: A User Interface Specification Model. Computer Aided Design of User Interfaces, CADUI'2002,, Les Valenciens, France.

38. Mori, G., F. Paternò, et al. (2003). Tool Support for Designing Nomadic Applications. 7 Int. Conf. on Intelligent User Interfaces IUI'03.
39. Mori, G., F. Paternò, et al. (2004). "Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions." IEEE Transactions on Software Engineering **30**: 507-520.
40. Myers, B., S. E. Hudson, et al. (2000). "Past, present, and future of user interface software tools." ACM Transactions on Computer-Human Interaction (TOCHI) **7**(1): 3-28.
41. Nichols, J., B. Rothrock, et al. (2006). Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances. the 19th annual ACM symposium on User interface software and technology, Montreux, Switzerland.
42. Nilsson, E. G. (2002). Combining Compound Conceptual User Interface Components with Modelling Patterns - A Promising Direction for Model-Based Cross-Platform User Interface Development. the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, Springer-Verlag, London, UK.
43. Nilsson, E. G., J. Floch, et al. (2006). "Model-based user interface adaptation." Computers & Graphics **30**(5): 692–701.
44. Nilsson, E. G., J. Floch, et al. (2006). Using a Patterns-based Modelling Language and a Model-based Adaptation Architecture to Facilitate Adaptive User Interfaces. DSV-IS 2006, The XIII International Workshop on Design, Specification and Verification of Interactive Systems.
45. Pribeanu, C., Q. Limbourg, et al. (2001). Task Modelling for Context-Sensitive User Interfaces. the 8th International Workshop on Interactive Systems: Design, Specification, and Verification.
46. Rolfsen, R. K., D. Boell, et al. (2007). Model-generated workplaces: An interoperability approach" til konferansen (skrevet av. 3rd International Conference Interoperability for Enterprise Software and Applications (I-ESA 07), Funchal (Madeira Island), Portugal.
47. Raatikainen, K., H. Christensen, et al. (2002). "Application requirements for middleware for mobile and pervasive systems." ACM SIGMOBILE Mobile Computing and Communications Review **6**(4): 16 - 24.
48. Schlungbaum, E. and T. Elwert (1996). Automatic User Interface Generation from Declarative Models. CADUI 96.
49. Seffah, A., P. Forbrig, et al. (2004). "Multi-devices ''Multiple'' user interfaces: development models and research opportunities." The Journal of Systems and Software **73**: 287–300.
50. Szekely, P. (1996). Retrospective and Challenges for Model-Based Interface Development. Design, Specification and Verification of Interactive Systems '96, Springer-Verlag, Wien.
51. Szekely, P., P. Sukaviriya, et al. (1996). "Declarative interface models for user interface construction tools: the MASTERMIND approach." Engineering for Human-Computer Interaction.
52. Trætteberg, H. (2002). Model-based User Interface Design. Department of Computer and Information Sciences (Information Systems Group). Trondheim, Norway, Norwegian University of Science and Technology (Faculty of Information Technology, Mathematics and Electrical Engineering): 211.
53. van Welie, M. and H. Trætteberg (2000). Interaction Patterns in User Interfaces. 17th. Pattern Languages of Programs Conference, Allerton Park Monticello, Illinois, USA.
54. Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems. 17th Conf. on Advanced Information Systems Engineering CAiSE'05, Porto, Springer-Verlag, Berlin.
55. Zhao, D., J. Grundy, et al. (2006). Generating mobile device user interfaces for diagram-based modelling tools. the 7th Australasian User interface conference, Hobart, Australia, Australian Computer Society, Inc. Darlinghurst, Australia.

Paper	Contribution	Application	Approach	Study
(Clerckx, Luyten et al. 2004)	***Technique:*** *ConcurTaskTrees Notation*	*adaptive user interfaces, multi-devices*	*task model, context model, run-time rendering*	
(Mori, Paternò et al. 2004)	***Tool:*** *TERESA (authoring tool)*	*multi-devices*	*"abstract modelling", three model levels, model transformation (MDA like approach)*	
(Gajos, Christianson et al. 2005)	***Tool:*** *SUPPLE (toolkit)*	*multi-devices, ubiquitous applications*	*declarative modelling, functional modelling, automatic generate user interface*	*experiments, user-study*
(Hanumansetty 2004)	***Framework***	*multi-devices, context aware UI, adaptive UI*	*context modelling, context and UI-mapping through rule specifications, task modelling*	*Demonstration*
(Eisenstein, Vanderdonckt et al. 2001)	***Techniques***	*mobile computing, multi-devices,*	*abstract UI modelling, context specification and mapping*	*Claim*
(Gajos and Weld 2004)	***Tool:*** *SUPPLE (toolkit)*	*multi-devices, interface adaptation*	*domain modelling, interface rendering algorithm*	*experiments, demonstrations*
(Mori, Paternò et al. 2003)	***Tool:*** *TERESA (authoring tool)*	*multi-devices*	*top-down transformation: task models, abstract UI, platform UI*	
(Pribeanu, Limbourg et al. 2001)	***Technique***	*multi-devices*	*task modelling, context modelling*	
(Book, Gruhn et al. 2006)	***Language:*** *DiaDef, framework: DiaGen*	*multi-devices, adoption*	*abstract language (DiaDef), device-independent widgets definitions, generate dialog masks and dialog flows (DiaGen)*	
(Limbourg, Vanderdonckt et al. 2004)	***Language:*** *USIMXML,* ***Framework:*** *Cameleon*	*multi-devices, -platform, -modality, -context*	*modelling: task, domain, presentation, dialog context (user, platform, environment), inter-model mapping*	
(Vanderdonckt 2005)	***Environment:*** *MDA-compliant*	*multi-devices*	*UsiXML, MDA*	
(Braubach, Pokahr et al. 2002)	***Survey:*** *MB-UIDE* ***Tools:*** *requirements and needs (MB-UIDE)* ***Architecture:*** *Arch model*	*multi-devices, ubiquitous computing*	*layered architecture support different implementation techniques and UI modalities*	*implementation demo (Vesuf), usability validation*
(Gomaa, Salah et al. 2005)	***Survey:*** *UI generation techniques* ***Framework:*** *MB-UIDE*	*multi-devices*	*multi-model conceptual layer*	
(Nilsson 2002)	***Technique:*** *compound conceptual UI component.* ***Language:*** *requirements,* ***Tools:*** *requirements*	*cross-platform UI*	*pattern-based abstract compound UI components, platform dependent mapping rules*	*Example*
(Nilsson, Floch et al. 2006)	***Approach:*** *modelling* ***Architecture:*** *adaptive middleware*	*flexible and adaptive UI*	*patterns-based modelling language, compound UI components and mapping rules, generic adaptive architecture*	*Examples*

Table 1: model-based mobile UI

Type	Contribution	Application	Approach	Paper
mobile (multi-device) UI design and validation	**Survey:** research opportunities	multi-devices	*Research opportunities within the fields of task and model-based, pattern-driven and device-independent development*	(Seffah, Forbrig et al. 2004)
UI generation	**Technique**	prototyping	*derivate UI from code, abstract UI commands*	(Jelinek and Slavik 2004)
XML-based UI	**Language:** Forms-XML	data-oriented UI	*automatically generate user interfaces for prescribed XML vocabularies*	(Kuo, Shih et al. 2005)
middleware requirements	**Survey:** requirements and key research areas	mobile and pervasive computing, middleware		(Raatikainen, Christensen et al. 2002)
model-based interoperability	**Framework**	interoperable systems	*reference models: conceptual, technical, and applicative integration*	(Elvesæter, Hahn et al. 2005)
sketch-based UI	**Tool:** SketchiXML	multi-devices	*sketching*	(Coyette and Vanderdonckt 2005)

Table 2: Div.

Paper	Contribution	Application	Approach	Study
(Molina, Meli´a et al. 2002)	**Technique**: Conceptuel User Interface Patterns	multi-devices, prototype UI	*from conceptual UI patterns to specific UI (prototype, automatic)*	
(Chung, Hong et al. 2004)	**Language:** initial and emerging pattern-language for ubiquitous computing **Collection:** 45- pre-patterns	interactive design, ubiquitous systems, mobile UI	*pattern-based design and evaluation*	effectiveness evaluation
(Lin and Landay 2002)	**Tool:** Damask (pattern-based design tool)	multi-devices, prototyping, mobile UI	*sketching, specifying, design-patterns, prototyping*	
(van Welie and Trætteberg 2000)	**Method:** UI-interaction patterns	UI	*UI patterns*	
(Lin 2005)	**Tool:** Damask (pattern-based design tool)	multi-devices, prototyping, mobile UI	*sketching, specifying, design-patterns, prototyping*	case study
(Gong and Tarasewich 2004)	**Guidelines** for mobile UI	mobile, multi-device UIs		
(Landay and Borriello 2003)	**Collection** of design-patterns for ubiquitous computing	mobile UI		
(Nilsson 2002)	**Technique:** compound conceptual UI component. **Language:** requirements, **Tools:** requirements	cross-platform UI	*pattern-based abstract compound UI components, platform dependent mapping rules*	example
(Nilsson, Floch et al. 2006)	**Approach:** modelling **Architecture**: adaptive middleware	flexible and adaptive UI	*patterns-based modelling language, compound UI components and mapping rules, generic adaptive architecture*	examples
(Coldewey and Krüge 1997)	**Language:** pattern-language	form-based UI	*pattern-language*	examples

Table 3: Pattern-based UI

Paper	Contribution	Application	Approach	Study
(Coldewey and Krüge 1997)	**Language:** pattern-language	form-based UI	*pattern-language*	examples
(Draheim and Weber 2003)	**Method:** two-staged interaction as the abstract concept behind form-based interfaces	form-based UI	*visual language, state transition diagrams*	
(Deng, Kemp et al. 2005)	**Method:** form storyboarding	form-based UI		survey of pattern-tools
(Draheim, Lutteroth et al. 2005)	**Methodology:** model-based form-oriented analysis	Content Editor UI	*from models to content editor UI through form-oriented analysis*	

Table 4: Form-based UI

Paper	Contribution	Application	Approach	Study
(Crowle and Hole 2003)	**Language:** ISML **Framework**	model-based UI	*metaphor modelling, design view mappings*	
(Molina, Meliá et al. 2002)	**Language** and **approach**	model-generated UI	*abstract UI, conceptual patterns, OOA*	
(Frank and Foley 1993)	**Framework:** interactive user interface design environment (interactive UIDE)	UI design	*story-boarding, model-based interface design*	
(Mahfoudhi, Abed et al. 2005)	**Approach:** task object oriented design (TOOD)	generating UI	*object oriented design, task modelling, user modelling, UI modelling, realization modelling*	example
(da Silva, Griffiths et al. 2000)	**Tool:** Teallach MB-UIDE (model-based user interface development environment)	UI generation, open architecture	*declarative interface specifications, model-view-controller pattern (MVC), task modelling, domain modelling, presentation modelling*	
(Eva 2002)	**Tool** **Method**	model-based UI	*transform user oriented task description into a specification of an interface*	prototyping
(Nichols, Rothrock et al. 2006)	**Tool:** Huddle	distributed systems	*automatically generate task-based interfaces*	
(Schlungbaum and Elwert 1996)	**Survey:** overview of model-based user interface software tools	Model-based UI	*user interface generation through declarative models*	examples, survey/review
(da Silva 2000)	**Survey:** 14 MB-UIDE technologies, **Framework:** MB-UIDE	UI design	*MB-UIDE provide a context within which user interface declarative models can be constructed and related, as part of the user interface design process*	Survey/review
(da Silva and Paton 2003)	**Language:** UMLi	UI	*UML-based modeling*	
(Fischer 1998)	**Framework:** generate spatial configuration in UI	visual presentation in direct-manipulative UI	*declarative models and inference mechanisms*	
(Iverson 2004)	**System architecture:** DKC Model	collaborative systems, multi-devices		
(Eisenstein and Puerta 2000)	**Technique:** adaptive algorithm to automated UI design	model-based UI design	*MOBI-D (Model-Based Interface designer) interface development environment*	preliminary experiments
(Cooper, McKirdy et al. 2000)	**Tool:** Teallach, **Language:** Domain Model	UI design	*capture domain model from database schema, use the domain model in UI*	
(Trætteberg 2002)	**Framework:** classifying design representation, **Language:** diagram-based modeling (user interface design, tasks, abstract and concrete interaction)	UI design	*combining user-centered design and model-based approach*	case study

Table 5: Model-based UI

References

1. Book, M., V. Gruhn, et al. (2006). Automatic dialog mask generation for device-independent web applications. the 6th international conference on Web engineering, Palo Alto, California, USA.
2. Braubach, L., A. Pokahr, et al. (2002). Tool-Supported Interpreter-Based User Interface Architecture for Ubiquitous Computing. Interactive Systems - Design, Specification, and Verification. Q. L. P. Forbrig, B. Urban, J. Vanderdonckt, Springer Heidelberg: 89 -103.
3. Chung, E. S., J. I. Hong, et al. (2004). Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing. DIS2004, Cambridge, Massachusetts, USA, ACM.
4. Clerckx, T., K. Luyten, et al. (2004). Generating Context-Sensitive Multiple Device Interfaces From Design. the Fifth International Conference on Computer-Aided Design of User Interfaces. CADUI'2004 long paper track, Funchal, Isle of Madeira.
5. Coldewey, J. and I. Krüge (1997). Form-Based User Interface The Architectural Patterns A Pattern Language. Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee, Germany,.
6. Cooper, R., J. McKirdy, et al. (2000). Conceptual Modelling for Database User Interfaces. the Fifth Working Conference on Visual Database Systems: Advances in Visual Information Management, Kluwer, B.V. Deventer, The Netherlands, The Netherlands.
7. Coyette, A. and J. Vanderdonckt (2005). Computer Assisted Sketching for the Early Stages of User Interface Design. 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact'2005, Rome.
8. Crowle, S. and L. Hole (2003). ISML: An Interface Specification Meta-Language. tenth International Workshop on the Design, Specification and Verification of Interactive Systems. DSV-IS'2003, Funchal, Madeira.
9. da Silva, P. P. (2000). User Interface Declarative Models and Development Environments: A Survey. Interactive Systems: Design, Specification, and Verification (7th International Workshop DSV-IS), Limerick, Ireland, Springer.
10. da Silva, P. P., T. Griffiths, et al. (2000). Generating user interface code in a model based user interface development environment. the working conference on Advanced visual interfaces, Palermo, Italy.
11. da Silva, P. P. and N. W. Paton (2003). "User Interface Modeling in UMLi." EEE Software **20**(4): 62-69.
12. Deng, J., E. Kemp, et al. (2005). Managing UI Pattern Collections. the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural, Auckland, New Zealand.
13. Draheim, D., C. Lutteroth, et al. (2005). Robust Content Creation with Form-Oriented User Interfaces. the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural, Auckland, New Zealand.
14. Draheim, D. and G. Weber (2003). Storyboarding Form-Based Interfaces. Human-Computer Interaction -- INTERACT'03, IOS Press, (c) IFIP.
15. Eisenstein, J. and A. Puerta (2000). Adaptation in automated user-interface design. the 5th international conference on Intelligent user interfaces, New Orleans, Louisiana, United States, ACM Press New York, NY, USA.
16. Eisenstein, J., J. Vanderdonckt, et al. (2001). Applying Model-Based Techniques to the Development of UIs for Mobile Computers. the 6th international conference on Intelligent user interfaces, Santa Fe, New Mexico, United States.

17. Elvesæter, B., A. Hahn, et al. (2005). Towards an Interoperability Framework for Model-Driven Development of Software Systems. the 1st International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA 2005), Geneva, Switzerland.
18. Eva, Y. h. C. (2002). Resource-Based User Interface Design. Department of Computer Science, The University of York.
19. Fischer, M. (1998). A Framework for Generating Spatial Configurations in User Interfaces. Design, Specification and Verification of Interactive Systems.
20. Frank, M. R. and J. D. Foley (1993). Model-Based User Interface Design By Example And By Interview. ACM Symposium on User Interface Software and Technology: 129-137.
21. Gajos, K., D. Christianson, et al. (2005). Fast and Robust Interface Generation for Ubiquitous Applications. Ubicomp'05, Tokyo, Japan.
22. Gajos, K. and D. S. Weld (2004). SUPPLE: Automatically Generating User Interfaces. the 9th international conference on Intelligent user interfaces, Funchal, Madeira, Portugal.
23. Gomaa, M., A. Salah, et al. (2005). Towards A Better Model Based User Interface Development Environment: A Comprehensive Survey. MICS 2005.
24. Gong, J. and P. Tarasewich (2004). Guidelines for Handheld Mobile Device Interface Design. DSI 2004 Annual Meeting.
25. Hanumansetty, R. G. (2004). Model Based Approach For Context Aware And Adaptive User Interface Generation.
26. Iverson, L. (2004). The DKC Model: An Application Model for Collaborative Work.
27. Jelinek, J. and P. Slavik (2004). GUI Generation from Annotated Source Code. the 3rd annual conference on Task models and diagrams, Prague, Czech Republic.
28. Kuo, Y. S., N. C. Shih, et al. (2005). Generating Form-Based User Interfaces for XML Vocabularies. the 2005 ACM symposium on Document engineering, Bristol, United Kingdom.
29. Landay, J. A. and G. Borriello (2003). "Design Patterns for Ubiquitous Computing." Computer, EEE Computer Society Press Los Alamitos, CA, USA **36**(8): 93 - 95.
30. Limbourg, Q., J. Vanderdonckt, et al. (2004). USIXML: A User Interface Description Language for Context-Sensitive User Interfaces. the 3rd annual conference on Task models and diagrams, Prague, Czech Republic.
31. Lin, J. (2005). Using Design Patterns and Layers to Support the Early-Stage Design and Prototyping of Cross-Device User Interfaces. UNIVERSITY OF CALIFORNIA, BERKELEY.
32. Lin, J. and J. Landay (2002). Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. The 8th International Conference on Distributed Multimedia Systems (2002 International Workshop on Visual Computing), San Francisco, CA,.
33. Mahfoudhi, A., M. Abed, et al. (2005). Towards a User Interface Generation Approach Based on Object Oriented Design and Task Model. the 4th international workshop on Task models and diagrams, Gdansk, Poland.
34. Molina, P. J., S. Meli´a, et al. (2002). User Interface Conceptual Patterns. Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, Springer-Verlag.
35. Molina, P. J., S. Meliá, et al. (2002). Just-UI: A User Interface Specification Model. Computer Aided Design of User Interfaces, CADUI'2002,, Les Valenciens, France.
36. Mori, G., F. Paternò, et al. (2003). Tool Support for Designing Nomadic Applications. 7 Int. Conf. on Intelligent User Interfaces IUI'03.

37. Mori, G., F. Paternò, et al. (2004). "Design and Development of Multi-device User Interfaces through Multiple Logical Descriptions." IEEE Transactions on Software Engineering **30**: 507-520.
38. Nichols, J., B. Rothrock, et al. (2006). Huddle: Automatically Generating Interfaces for Systems of Multiple Connected Appliances. the 19th annual ACM symposium on User interface software and technology, Montreux, Switzerland.
39. Nilsson, E. G. (2002). Combining Compound Conceptual User Interface Components with Modelling Patterns - A Promising Direction for Model-Based Cross-Platform User Interface Development. the 9th International Workshop on Interactive Systems. Design, Specification, and Verification, Springer-Verlag, London, UK.
40. Nilsson, E. G., J. Floch, et al. (2006). "Model-based user interface adaptation." Computers & Graphics **30**(5): 692–701.
41. Pribeanu, C., Q. Limbourg, et al. (2001). Task Modelling for Context-Sensitive User Interfaces. the 8th International Workshop on Interactive Systems: Design, Specification, and Verification.
42. Raatikainen, K., H. Christensen, et al. (2002). "Application requirements for middleware for mobile and pervasive systems." ACM SIGMOBILE Mobile Computing and Communications Review **6**(4): 16 - 24.
43. Schlungbaum, E. and T. Elwert (1996). Automatic User Interface Generation from Declarative Models. CADUI 96.
44. Seffah, A., P. Forbrig, et al. (2004). "Multi-devices ''Multiple'' user interfaces: development models and research opportunities." The Journal of Systems and Software **73**: 287–300.
45. Trætteberg, H. (2002). Model-based User Interface Design. Department of Computer and Information Sciences (Information Systems Group). Trondheim, Norway, Norwegian University of Science and Technology (Faculty of Information Technology, Mathematics and Electrical Engineering): 211.
46. van Welie, M. and H. Trætteberg (2000). Interaction Patterns in User Interfaces. I7th. Pattern Languages of Programs Conference, Allerton Park Monticello, Illinois, USA.
47. Vanderdonckt, J. (2005). A MDA-Compliant Environment for Developing User Interfaces of Information Systems. 17th Conf. on Advanced Information Systems Engineering CAiSE'05, Porto, Springer-Verlag, Berlin.