

# **Model-based User Interface Software Tools**

## **Current state of declarative models**

by

**Egbert Schlungbaum**

**GIT-GVU-96-30**

**November 1996**

**Graphics, Visualization & Usability**

**Center**

**Georgia Institute of Technology**

**Atlanta, GA 30332-0280**

# Model-based User Interface Software Tools

## Current state of declarative models

**Egbert Schlungbaum**

Department of Computer Science

University of Rostock

D-18051 Rostock

Egbert.Schlungbaum@informatik.uni-rostock.de

GVU Center

Georgia Institute of Technology

Atlanta, GA 30332-0280

eggi@cc.gatech.edu

### Abstract

The Interface Model is central to all model-based user interface software tools. This report investigates the different use of declarative models as a part of the Interface Model in model-based interface development environments. Furthermore, we introduce definitions for the different declarative models. The report concludes with a description of an ontology of declarative models for future model-based interface development environments.

### Keywords

Model-based User Interface Software Tools, Model-based Interface Development Environments, Interface Model, Declarative models

### Acknowledgments

This research is accomplished at Georgia Tech's GVU Center and is supported by funding of the German Academic Exchange Service (Post-Doc fellowship). The author would like to thank Jim Foley for his support preparing this research visit, Spencer Rugaber and Kurt Stirewalt for the very interesting and fruitful cooperation during the past 8 month. This report reflects issues discussed with Pedro Szekely and Angel Puerta during the CADUI'96 workshop in Namur (Belgium).

# 1 Introduction

End users expect advanced interactive applications to be easy-to-use and easy-to-learn. Today end users expect to be able to sit down and use software without spending their time reading manuals. But such user interfaces are hard to design and implement. Different studies have shown that an average of 48% of the code of an application is devoted to the user interface, and that about 50% of the implementation time is devoted to implementing the user interface portion [Myers92]. As user interfaces become easier to use, they become harder to create. User interface developers need tools which provide a rich support for the development of advanced user interfaces.

Over the last years several tools were created to support user interface developers, e.g. Toolkits, User Interface Management Systems, Interface Builders, User Interface Development Environments. In his state of the art report B. Myers has introduced a classification of these user interface software tools [Myers95]. It is based on the way user interface developers can specify the layout and the dynamic behavior of a user interface. There are *language-based tools* (they require the developer to program in a special-purpose language), *interactive graphical specification tools* (they allow an interactive design of the user interface), and *model-based generation tools* (they use a high level model or specification to generate the user interface automatically).

The user interface development still remains difficult and time consuming when using language-based or interactive graphical specification tools because they support the specification of either the dynamic behavior or the layout in an easy way but mostly not both parts at one time. Current user interface tools support only the development phase of the user interface life cycle, and the abstractions they provide have only a distant connection to the results of a user-task analysis. The model-based user interface development approach and its supporting tools is an emerging technology to remedy these shortcomings of current technology through a comprehensive support of the whole life-cycle, and a user-centered design methodology with corresponding environments. Furthermore, Olsen et.al. [Olsen93] suggest the automatic user interface generation is an essential part of future user interface

development environments (e.g., Model-based User Interface Software Tools - MbUIST or Model-Based Interface Development Environments - MB-IDEs).

Central to the model-based approach is that all aspects of a user interface design are represented using declarative models. This paradigm offers a number of key benefits: user-centered development cycle, centralized user interface specification, comprehensive design time tools for interactive and automated development, re-use of user interface designs, and use of explicitly represented design knowledge.

There are at least two necessary criteria for a user interface tool to be a model-based interface development environments:

- (1) MB-IDEs must include a high-level, abstract, and explicitly represented (declarative) model about the interactive system to be developed (either a task model or a domain model or both).
- (2) MB-IDEs must exploit a clear and computer-supported relation from (1) to the desired and running user interface. That means, that there is some kind of automatic transformation(s) like knowledge-based generation or simple compilation to implement the running user interface.

These criteria exclude Interface Builders (e.g., DevGuide, UIMX, NeXT) or interface design advisors (e.g., IDA [Reiterer94], EXPOSE [Gorny95]).

Several model-based user interface software tools have been built. Some of these are UIDE [Foley88, Foley89, Foley95], ADEPT [Johnson95, Wilson93], HUMANOID [Szekely93], ITS [Wiecha90], MECANO [Puerta94b], Mobi-D [Puerta96a], MASTERMIND [Neches93, Szekely96a], TRIDENT [Bodart95, Bodart96], AME [Märting96], FUSE [Lonczewski96], GENIUS [Janssen93], JANUS [Balzert96], TADEUS [Elwert95].

## 2 Model-based Interface Development Environments

### 2.1 Generic Architecture

The typical components and the principal development procedure of a model-based interface development environment are shown in Fig. 1. The central component of each MB-IDE is the *Interface Model* which includes different declarative models. These declarative models are discussed in the following paragraph. MB-IDEs include tools for *interactive development* (Modeling Tools, Design Critics, Design Advisors) and *automated development*. Automatic Generation Tools deal with transformations between different declarative models and implement an executable representation of the desired user interface. Design Critics, Design Advisors, and Automatic Generation Tools require additional knowledge represented in the *Knowledge Bases*.

User Interface Developers use Modeling Tools to create and manipulate the declarative models. Usually, MB-IDEs contain special-purpose (graphical) editors for each declarative model, e.g., a task modeling editor for handling the task model, etc.

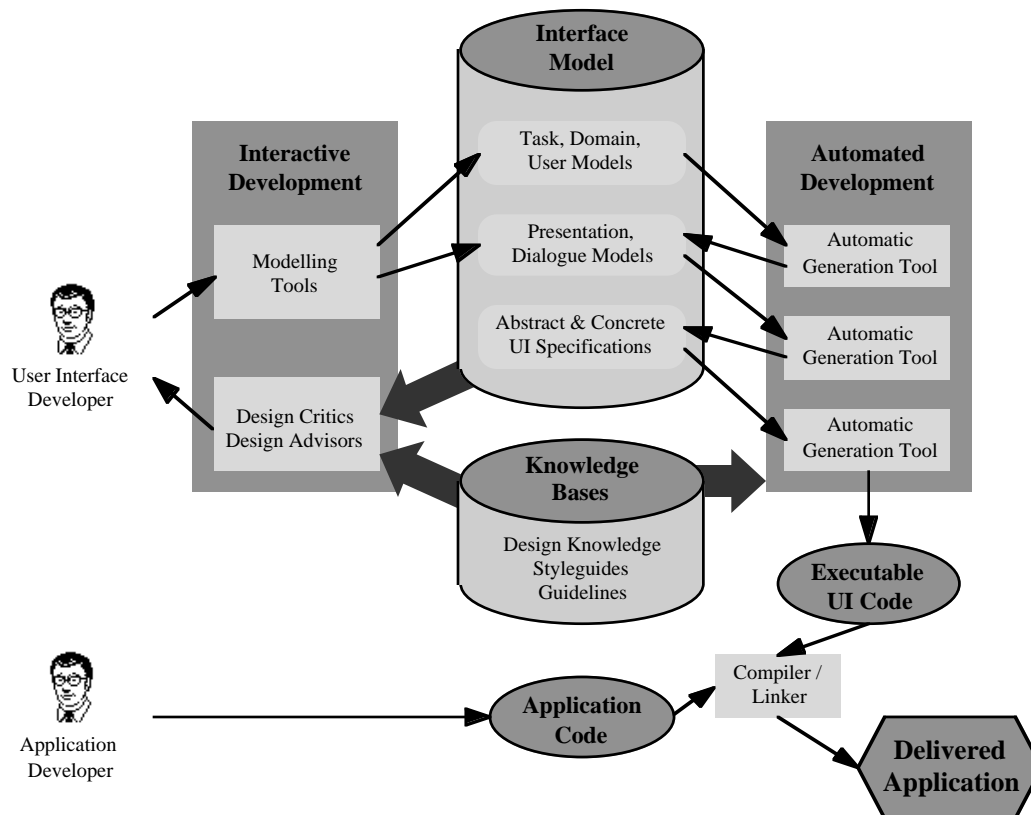


Fig. 1 Generic architecture of a MB-IDE and model-based user interface development process

## 2.2 Declarative Models

Different model-based approaches use different declarative models (see also [Puerta94a], [Wilson94], [Schlungbaum96a], [Wilson96b]). Generally, all MB-IDEs use either a task or a domain model. Some MB-IDEs use both. Many MB-IDEs use dialogue and/or presentation models. Furthermore, researchers mention user, implementation platform, and workplace models, but these are rarely used. Henceforth, we discuss only the most frequently used models: task, domain, user, dialogue and presentation models because these are the declarative models which are used most frequently.

Before we discuss the use of the declarative models in certain MB-IDEs we introduce their definitions. Until now, there are not any clear definitions of the different declarative models used in the MB-IDEs [Wilson96b].

### Task model

The book [Diaper89] presents 5 different approaches to task analysis, but there is no consensus on the precise definition of a task. As Paul Walsh put it:

"In different methods, it is possible to find a description of tasks expressed in terms of one, some or all of the following: objects, actions, roles, goals, procedures, functions, processes, forms, attributes, relations, predicates, rules, inputs/outputs, and transitions between states." [Walsh89, p.191]

The task analysis community distinguishes between task processing and goals, but these are closely related. People act to achieve their purposes or goals. They do so by performing some actions which usually change the state of a certain artifact in a certain context. This is what we usually call task processing. Each task processing serves a certain goal and is prerequisite to the satisfaction of that goal. Task processing describes an activity carried out by a certain person to change the state of a domain.

G. Storrs has tried to unify the different definitions of a task. For our purposes we can reuse his definition:

A *task* is a goal together with the ordered set of tasks and actions that would satisfy it in the appropriate context. [Storrs95, p.358]

This definition explicates the intertwining nature of tasks and goals. Actions are required to satisfy goals. Furthermore, the definition allows the decomposition of tasks into sub-tasks and there exist some ordering among the sub-tasks and actions. In order to complete this definition we need to add the definition of goal and action:

A *goal* is an intention to change or maintain the state of an artifact (based on [Storrs95, , p. 359]).

An *action* is any act that has the effect of changing or maintaining the state of an artifact (based on [Storrs95, p. 359]).

These two definitions imply that the artifact is essential for task performance. Without an artifact tasks lose their existence. Artifacts are real things existing in the context of task performance - in the domain. Artifacts are modeled as objects and represented in the domain model. This implies a close relationship between the task model and the domain model.

With these definitions we can derive the information necessary to represent in a task model. According to [Storrs95], one task description includes

- one goal,
- a non-empty set of actions or other tasks which are necessary to achieve the goal,
- a plan of how to select actions or tasks, and
- a model of an artifact which is influenced by the task.

The above task description can be used as a basis for a task specification language. In fact, the task models of MB-IDEs mentioned in this report are based on this description (see next section). Two remarks are necessary. First, for software developers it is difficult to distinguish between goals and actions. Usually both have the same designation, and this may cause that a task specification language contains only a goal or an action. Second, a task description usually includes a reference to an object in the domain model rather than a model of the domain itself.

Which tasks should be modeled in a task model? The simple and clear answer is all tasks the end user plans to accomplish using the interactive system. To achieve this, care must be taken. Task analysis methods tend to model ideal situations and they do not recognize real world obstacles [Hsi95]. Trouble arises if the accomplishment of goals depends on various factors inside the domain or involves different users like in CSCW systems. Therefore, a task model must include not only the tasks end users need to accomplish, but also tasks which deal with different obstacle-situations. The ordering operators of task modeling

languages (e.g., Sequence, Choice, Order independence) are not expressive enough to describe plans with obstacle-situations. One possible solution is the additional use of preconditions and effects in the task model in order to describe such complicated plans as they are for obstacle-situations in an effective way. The user interface developer should use ordering operators and preconditions with effects carefully in order to avoid inconsistencies. Such task modeling must be supported by an appropriate tool.

The use of preconditions and effects reveals another important fact of model-based user interface development. End users perform tasks to achieve their goals which represent a state of an artifact of the domain. This artifact is modeled using the domain model (see next subsection). In order to formulate preconditions and effects in the task model the user interface developer needs some understanding of the domain model. Consequently, the development of the task model and of the domain model is interrelated.

MB-IDEs usually include one task model which describes the tasks end users plan to carry out using the developed interactive system. In the community of task-based design approaches [Wilson96a], this task model is called *envisioned task model*. An envisioned task model is a result of a design step based on the *existing task model*. The existing task model describes the current work situation and can be a result of a task analysis. The task-based design recommends the exploration of multiple envisioned task models based on the one existing task model. If recent MB-IDEs include a task model then it should be an envisioned task model, but these tools do not worry about the way to get it. One of the goals of MB-IDEs is to support user-centered interface design. Therefore, they must enable the user interface designer to create the different task models.

## **Domain Model**

The first MB-IDEs were using a domain model to drive the user interface at runtime. These domain models describe the application in general and include some specific information for the user interface. For example, the UIDE domain model [Foley88, Foley89, Foley91] includes:

- a class hierarchy of objects which exist in the application,
- properties of the objects,
- actions which can performed on the objects,
- units of information (parameters) required by the actions, and

- pre- and postconditions for the actions.

Such a domain model is very similar to the models known from software engineering methods and corresponding tools, especially the recent object-oriented methods and their supporting tools. However, the software engineering methods do not have any real focus on the user interface. On the other hand, the domain models developed by the user interface community additionally includes some user interface information. Nevertheless, the independent developments in these apparently different communities contain a danger. Jim Foley wrote:

"... some information has to be specified twice, once by the software engineer and once by the user interface designer. These tools should clearly be merged to speed up the design process, avoid duplication of efforts, and avoid potential inconsistencies between the dual specification." [Foley91, p. 11].

In their basic form domain models should represent the important entities together with their attributes, methods, and relationships. This kind of a domain model corresponds to the object model of recent object-oriented software development methods. MB-IDEs which feature an automatic generation of the desired user interface usually extend the basic domain model with user interface specific information (see next section).

After more than a decade, software engineering methods, irrespective of whether they are structured or object-oriented, are criticized for their over-simplified or non-existing support of user interface development. This trend remains even though most of today's software developed is interactive. Even the most recent development by Grady Booch, James Rumbaugh, and Ivar Jacobson, the Unified Modeling Language, a third generation object-oriented modeling language [UML96], does not change this long-time observed situation. Consequently, the only real way to integrate user interface and system development is the simultaneous use of the data model. That is why, recent MB-IDEs include a domain model known from the software engineering methods.

## **User Model**

A user model describes the characteristics of the desired end users or groups of end users (such a group is called a role) of the interactive system to be developed. The main purpose of a user model is to support the creation of individual user interfaces [Schlungbaum96c]. The individualization of a user interface can be constituted during design time or runtime. Design

time individualization (e.g., development of adapted user interfaces) is usually supported by a user model which represents the different roles, whereas the runtime individualization (e.g., support of adaptive user interfaces) requires an individual user model for each user.

User characteristics can be classified as application independent and application dependent. Application independent characteristics include preferences, capabilities, psychomotor skills, etc. Application dependent characteristics include goals, knowledge of system and application, etc. Furthermore, a user model include rules to reason from a certain user characteristic to a user interface design decision (e.g., selection of interaction objects)

### **Dialogue Model**

A dialogue model is used to describe the human-computer conversation. It describes when the end-user can invoke commands, select or specify inputs and when the computer can query the end-user and presents information. In other words, the dialogue model describes the sequencing of input tokens, output tokens and their interleaving. It describes the syntactic structure of human-computer interaction. The input and output tokens are lexical elements.

The earlier MB-IDEs did not use an explicit dialogue model. The dialogue structure information was included into the task or domain models.

### **Presentation Model**

A presentation model describes the constructs that can appear on an end user's display, their layout characteristics, and the visual dependencies among them. The displays of most applications consist of a static part and of a dynamic part. The static part includes the presentation of the standard widgets like buttons, menus, list boxes. Typically, the static part remains fixed during run-time of the interactive system except for state changes like enable/disable, visible/invisible. The dynamic part displays application-dependent data what typically changes during run-time (e.g., the application generates output information, the end user constructs application specific data). The presentation of application-dependent data is one of the biggest pitfall if user interface developers use recent Interface Builders. Interface Builders support only a general drawing area (e.g., Canvas widget) and all output inside a canvas must be programmed using a general purpose programming language and a low-level graphical library.

### 2.3 MB-IDEs and their declarative models

This section summarizes the declarative models used in different Model-based Interface Development Environments. This discussion includes the kind of declarative models, what information these models represent and in which way the information is used to create the running user interface. But it does not focus on the development of the functional core of the application and the way the user interface is connected with the application core. All recent MB-IDEs support an separated development of the user interface and application parts. The presented information is summarized in Tables at the end of this section.

#### UIDE

In UIDE [Sukaviriya93b] the designer has to specify an *application model* (domain model) that consists of *application actions*, *interface actions*, and *interaction techniques*. Parameters, pre-conditions, and post-conditions are assigned to each action. The pre- and post-conditions are used to control the user interface during run time by means of the UIDE runtime system. An extension to UIDE [Sukaviriya95] provides an *application model* and an *interface model*. The application model consists of tasks which will be performed by end-users, their operational constraints, and objects on which these tasks operate. Interface components, application-independent interface tasks, and operational constraints on these tasks are specified in the interface model. The application model drives a special purpose runtime system (e.g., the Simple User Interface Management System [Foley89]) to create the running user interface. In this way, the application semantic information which is stored in the application model is preserved from design time to run time. So it can be used for some sophisticated tools to support the end-user, e.g. automatic generation of context-sensitive, animated help [Sukaviriya90] or adaptive user interfaces [Sukaviriya93a].

#### AME and JANUS

The AME [Märting96] and JANUS [Balzert96] systems are very similar from the point of view of declarative models. Both systems emphasize the automatic generation of the desired user interface from a much extended object-oriented *domain model*. During this automatic generation process both systems make use of different comprehensive knowledge bases. They do not include any other declarative models. Both systems generate user interface code

for different target systems like C++ source code in order to link it with UI toolkits or input files for UIMS like ISA Dialogue Manager or Open Interface.

## **TRIDENT**

In TRIDENT [Bodart95a, Bodart95b, Bodart96] the designer has to specify a *task model* which is represented by an Activity Chaining Graph (ACG) and an *application model* in form of an entity-relationship diagram. The task model includes the interactive tasks the end-user has to perform, and the sequencing information for tasks in order to achieve the related goal. During the further development of the user interface the Activity Chaining Graph is extended by presentation units. These presentation units represent an additional input for the automatic generation procedure in the TRIDENT system. They could be understood as a very simple presentation model. But the presentation units do not include any more information regarding the user interface layout and that is why we will consider them as an extension to the task model. Finally, a textual description of the user interface is generated by using of different comprehensive knowledge bases.

## **GENIUS**

In GENIUS [Janssen93] the designer uses the *existing data model* of the application to design the user interface. This model of the application (domain model) is represented as an extended entity relationship model. In the data model the developer must define *views* that are used for explicit *dialogue modelling* by means of Dialogue nets and for the layout generation. A textual description of the user interface is generated which is an input for the UIMS ISA Dialogue Manager.

## **TADEUS**

In TADEUS the user interface developer creates the task, domain, user, and dialogue models [Elwert94, Elwert95] which are the basis of subsequent development of the interactive and noninteractive parts of interactive applications. The *task model* represents a hierarchical structure of the tasks the end user plans to carry out using the interactive system. Like TKS [Johnson92] each task representation includes a goal, a procedure to achieve this goal, preconditions and effects, and subtasks with an ordering. The *domain model* represents the

important entities (objects with attributes and methods) of the application domain using an object model known from object-oriented analysis methods like OMT [Rumbaugh91]. The *user model* describes prospective end users in terms of roles and their relations to the tasks they perform.

On the basis of these three declarative models the dialogue developer creates the *dialogue model*. The dialogue design includes manual, computer-aided, and automatic steps. Two levels of dialogue are distinguished in the TADEUS dialogue model: the navigation and the processing dialogue. The navigation dialogue describes the sequencing between different task-oriented presentation units called dialogue views. It can be specified by means of Dialogue graphs [Schlungbaum96b] which are based on Coloured Petri Nets. The processing dialogue deals with the description of the dialogue within a dialogue view including the realisation of state changes on the level of user interface objects. All parts of processing dialogue could be generated automatically, but in order to improve the generation result the user interface developer can provide additional presentation information using interaction tables.

In TADEUS the final user interface is generated automatically from the specified declarative models, a software ergonomics knowledge base, and auxiliary interaction with the user interface developer in order to request non-specified information. The result of the generation procedure is a dialogue script file for an existing UIMS, e.g. the ISA Dialogue Manager.

### **MECANO --> Mobi-D**

The development of the Mobi-D modeling language (MIMIC) and the corresponding tool environment [Puerta96a] is based on the experience developing the MECANO system [Puerta94b]. While the MECANO system like the AME or JANUS systems only used a domain model to generate the final user interface the Mobi-D system supports the user interface developer to design the user interface through specifying *task*, *domain*, *user*, *dialogue*, and *presentation models*. Furthermore, the MIMIC modeling language supports a *design model*. This design model describes dynamic relationships between entities of the other declarative models. It allows a considerable flexibility for the designer whereas for example, the relationships between the declarative models in TADEUS are predefined.

Although the Mobi-D environment emphasizes support of the user interface design instead of automatic generation like in MECANO, it will contain automatic tools to do automatic transformations between the different declarative models and to implement the final user interface without additional programming.

## **FUSE**

The use of declarative models in FUSE [Lonczewski96] is similar to TADEUS. The developer creates task, domain, user, and dialogue models. The *task model* represents a hierarchical structure of the end user's tasks. The *domain model* uses an algebraic specification to describe the functions and data structures of the UI-relevant part of the functional core of the application. The *user model* is a description of static and dynamic properties of user groups and individual users which influence both the UI generation process and the kind and depth of the help offered by the user guidance component. The *dialogue model* is generated by using the task, domain, and user models and additional dialogue design guidelines and can be modified by the UI developer. The dialogue model is represented by Hierarchic Interaction graph Templates (HIT) [Schreiber95] which are based on attribute grammars and dataflow diagrams.

The final user interface is automatically generated from all four declarative models. Furthermore, the FUSE system allows the automatic generation of an intelligent user guidance component from the task and dialogue models.

## **MASTERMIND**

In MASTERMIND [Szekely96a] the user interface developer has to create task, application (domain), and presentation models. The *application model* is specified using the CORBA interface definition language (IDL). The *task model* describes the end user's tasks in a hierarchical structure and contains the necessary ordering information to control the user interface at runtime. The *presentation model* describes the layout of the user interface including static and dynamic displays. It allows the specification of automatic presentation updates when the application data or the presentation context changes. Furthermore, it incorporates principles of graphic design in order to give a comprehensive support to the dialogue designer.

Table 1 Declarative Models and their application in MB-IDEs










































	<b>Task</b>	<b>Domain</b>	<b>User</b>	<b>Dialogue</b>	<b>Presentation</b>
<b>UIDE</b> [Foley95]					
<b>ADEPT</b> [Johnson95]					
<b>MASTERMIND</b> [Szekely96a]					
<b>MECANO</b> [Puerta94b]					
<b>Mobi-D</b> [Puerta96a]					
<b>TRIDENT</b> [Bodart95,96]					
<b>AME</b> [Märting96]					
<b>GENIUS</b> [Janssen93]					
<b>FUSE</b> [Lonczewski96]					
<b>JANUS</b> [Balzert96]					
<b>TADEUS</b> [Elwert94,95]					
	model X in MB-IDE Y fits our definition				
	model X in MB-IDE Y fits our definition but the model is much extended				
	model X is included in the Interface Model of MB-IDE Y, but there are not any examples which make use of it				

Table 2 Domain Models and their possibility for integration with recent CASE tools

	<b>Domain model</b>	<b>Possibility for integration</b>
<b>UIDE</b> [Sukaviriya93b]	C++ classes	
<b>MASTERMIND</b> [Szekely96a]	CORBA IDL	
<b>Mobi-D</b> [Puerta96a]	MIMIC Object model	
<b>TRIDENT</b> [Bodart95,96]	ER model	
<b>AME</b> [Märting96]	OO-Object model	
<b>GENIUS</b> [Janssen93]	ER model	
<b>FUSE</b> [Lonczewski96]	algebraic specification	
<b>JANUS</b> [Balzert96]	OO-Object model	
<b>TADEUS</b> [Elwert94,95]	OO-Object model	
	domain model is equivalent to the model of recent CASE tools	
	integration of this domain model requires additional transformations	
	a direkt integration is difficult	

### 3 Declarative models in future MB-IDEs

The hitherto existing model-based user interface software tools could be classified into two generations:

- **1. Generation**

The first generation tools like UIDE, MECANO, AME, JANUS used one universal declarative model as the Interface Model. They emphasized the fully automatic generation of the final user interface instead of a user interface design process.

- **2. Generation**

The Interface Model of the second generation tools like MASTERMIND, Mobi-D, FUSE, TADEUS is much improved. Their Interface Model is structured into many declarative models (see Fig. 1). The second generation MB-IDEs support an incremental user interface design. During the user interface design process the designer has to specify different aspects of the desired user interface for what these tools provide specific declarative models like task, domain, user, dialogue, and presentation models.

The current MB-IDEs are criticized because they mostly support the generation of form-based user interfaces only. One of the reason of this situation is that most MB-IDEs generate the final user interface for UI toolkits or UIMS which provide a large-scale set of UI widgets<sup>1</sup> supporting form-based user interfaces only. The use of UI elements with a standard behavior is reflected in the declarative models. For example, the TADEUS dialogue model includes the rudimentary specification of the input token (end user interactions - lexical elements of the end user's input language) and output tokens (layout parameter - lexical elements of the computer's output language). Other MB-IDEs like Mobi-D include a separate presentation model in order to specify the output token more exactly. However, no one of the current MB-IDEs includes an interaction model in order to specify the input token.

But a comprehensive Graphical User Interface needs to represent application specific objects and to attach to them an interactive behavior. This fact requires a rethinking about the

---

<sup>1</sup>A UI widget consists of a layout and an standard interactive behavior whereby both parts tightly interconnected. The user interface developer can change characteristics of a widget using its attributes.

ontology<sup>2</sup> of the Interface Model. The following suggestion is based on the current discussion in the MASTERMIND project team [Browne96].

The Interface Model ontology of future MB-IDEs is influenced by two ideas. The first is programming language design, which provides foundations for any modeling process. The second is object-oriented user interface toolkit technology, which provides abstractions and control mechanisms. User interface design is analogous to programming language design [Foley90]. Interfaces can be thought of as being composed of two languages: one in which the end-user communicates to the computer, and one in which the computer communicates to the end-user. The act of engineering user interfaces can be therefore thought of as the simultaneous design of these two languages. Independent of this is the emergence of new abstractions in the object oriented user interface toolkit community [Myers90a]. Two constructs, constraints and interactors, permit sufficiently declarative models to instantiate the design ontology and compile into executable code.

The language analogy suggests applying the phases of programming language design to the simultaneous design of the input and output languages. There are three phases in this process:

- **Semantic Design**

Semantics refers to the meaning or intentions of the end user. User task analysis [Diaper89] is applied to codify a system at this level. This is the understanding the end user will have in his or her mind when using the system. This conceptual model is refined into a detailed semantic model of the system by incorporating the functional requirements of the application.

- **Syntactic Design**

A syntactic model is defined which denotes the detailed semantic model. Forms in the syntactic model represent procedures that, when executed, cause some semantically prescribed effect to occur. The primary difference between syntactic and semantic models is that syntactic models are specified procedurally (as an ordered series of steps); whereas semantic models are non-procedural (declarative). Dialogue techniques are syntactic; whereas the user tasks that these dialogue techniques are employed to accomplish are semantic.

---

<sup>2</sup>Ontology: a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents [Gruber].

- **Lexical Design**

A lexical model, describing low-level tokens, is defined alongside the syntactic model. For the input language, these tokens include keystrokes, mouse clicks, or mouse motion. For the output language, these tokens include output characters, beeps, or graphical widgets. Sometimes lexical tokens are organized into gestures<sup>3</sup>.

The language analogy, while conceptually elegant, is not feasible if the lexical level embodies keystroke level input events and atomic output events like beeps, or flashes. This is analogous to a compiler whose lexical analyzer feeds single characters to its parser. Compiler theory teaches us that single characters are inappropriate tokens because they inject too much complexity into the corresponding parsers. To see how this problem manifests itself in UI design, consider a drag-and-drop interface. A burst of activity begins with an end user clicking down on an icon. Immediately a shadow icon appears, and as the end-user moves the mouse, the shadow icon changes its position on the screen. The burst of activity ends when the end-user releases the mouse button signifying a drop over some other icon. If we consider each mouse move event and icon redraw to be tokens, then the syntactic description of drag and drop will be extremely complex. Compiler theory suggests we simplify this problem by making tokens more abstract than just mere input and output events. In compilers, tokens embody patterns of input characters. Lexical UI design should, therefore, identify patterns of input/output events and treat these patterns as tokens. User-computer interfaces are characterized by bursts of tight, high volume, feedback intensive, input/output event sequences, and these sequences can be described by patterns. To represent lexical patterns, we pull ideas from object oriented UI toolkits.

Object oriented toolkits like Garnet [Myers90b], and Amulet [Myers96] provide interactors which encapsulate these tight input/output protocols into implemented units that may be selected from a library and specialized to a particular use. In both toolkits, the number of interactors is fixed and relatively small. The *interaction model* is an abstraction of interactors, and we consider it a lexical model. As another example, consider how window sizes might need to change dynamically to accommodate the insertion/deletion of graphical objects. Object oriented toolkits use constraints to declare that an object's attributes be dynamically recomputed when another object's attributes change. This mechanism abstracts away a great deal of sequencing that would otherwise have to be implemented in the two

---

<sup>3</sup>gesture: a stereotypical sequence of end-user input actions such as clicking on an object and then moving the mouse.

languages. The *presentation model* allows designers to use constraints when specifying presentation layout, and we consider it a lexical model. The final lexical specification is the *application wrapper*. The application wrapper is currently nothing more than an application program interface for invoking behavior in a (possibly distributed) application. Treating it at the lexical level allows us to consider method invocations as tokens.

The syntactic component of design is captured by one model, the *dialogue model*. Syntactic models abstract a meaningful ordering structure over tokens described in the lexical models. In programming languages, syntactic models are usually specified by context-free grammars whose terminal symbols correspond to tokens in a lexical model. Unfortunately, there are orderings mechanisms in dialogue models that are not expressible using context free grammars. The essence of grammars, however, are hierarchical ordering constraints over tokens. The dialogue model declares hierarchical ordering constraints over interaction tokens.

Task organization and application functionality are semantic concepts. Interaction, presentation, and application invocation are lexical concepts, and dialogue is a syntactic concept. These models must precisely define the input and output languages and express their interleaving. This ontology is depicted in the figure below:

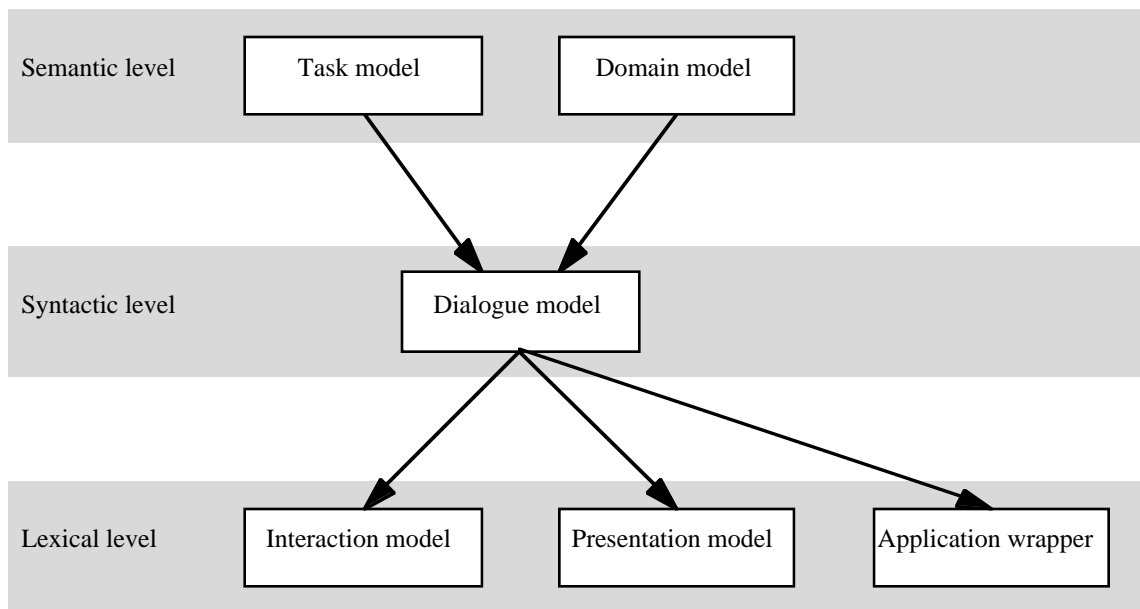


Fig. 2 Ontology of an Interface Model of future MB-IDEs

## References

- [Balzert96] H. Balzert, F. Hofmann, V. Kruschinski, C. Niemann: The Janus Application Development Environment Generating More than the User Interface. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 183-205.
- [Bodart95a] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, J. Vanderdonckt: A Model-based Approach to Presentation: A Continuum from Task Analysis to Prototype. In: F. Paterno (ed.): Interactive Systems: Design, Specification and Verification. Berlin: Springer, 1995, 77-94.
- [Bodart95b] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, B. Sacre, J. Vanderdonckt: Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. In P. Palanque, R. Bastide (eds.): Design, Specification and Verification of Interactive Systems. Wien: Springer, 1995, 262-278.
- [Bodart96] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, J. Vanderdonckt, G. Zucchini: Key Activities for a Development Methodology of Interactive Applications. In: D. Benyon, P. Palanque (eds.): Critical Issues in User Interface Systems Engineering. London: Springer, 1996, 109-134.
- [Browne96] T. Browne, D. Davila, S. Rugaber, K. Stirewald: The MASTERMIND User Interface Generation Project. Research report, Georgia Institute of Technology, Graphics, Visualization & Usability Center, GIT-GVU-96-31, 1996.
- [Castells97] P. Castells, P. Szekely, E. Salcher: Declarative Models of Presentation. In Proceedings of Intelligent User Interfaces '97, New York: ACM Press, 1997, in press.
- [Diaper89] D. Diaper (ed.): Task Analysis for Human-Computer Interaction. Chichester: Ellis Horwood, 1989.
- [Elwert94] T. Elwert, P. Forbrig, E. Schlungbaum: Meta Models for Task-oriented User Interface Development. In: Proceedings 1. Workshop on Cognitive Modelling and Interface Development, Wien, 1994, S. 163-172.

- [Elwert95] T. Elwert, E. Schlungbaum: Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In: P. Palanque, R. Bastide (eds.): Designing, Specification, and Verification of Interactive Systems. Wien: Springer, 1995, 193-208.
- [Foley88] J. Foley, C. Gibbs, W. Kim, S. Kovacevic: A Knowledge-based User Interface Management System. In: E. Soloway, D. Frye, S. Sheppard (eds.): Human Factors in Computing Systems. Proceedings CHI'88 (Washington, May 1988). New York: ACM Press, 1988, 67-72.
- [Foley89] J. Foley, W. Kim, S. Kovacevic, K. Murray: The User Interface Design Environment - A Computer Aided Software Engineering Tool for the User Computer Interface. IEEE Software 6 (January 1989), 1, 25-32. (Special Issue on User Interface Software)
- [Foley90] J. Foley, A. van Dam, S. Feiner, A. Hughes: Computer Graphics: Principles and Practice. (2nd Ed.) Reading: Addison Wesley, 1990.
- [Foley91] J. Foley: User Interface Software Tools. Research report GIT-GVU-91-29, Graphics, Visualization & Usability Center, Georgia Institute of Technology, 1991.
- [Foley95] J. Foley, P. Sukaviriya: History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Implementation. In: F. Paterno (ed.): Interactive Systems: Design, Specification and Verification. Berlin: Springer, 1995, 3-14.
- [Gorny95] P. Gorny: An HCI-Counseling for User Interface Design. In: Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT '95 (Lillehammer, June 1995). London: Chapman & Hall, 1995, 297-304.
- [Gruber] Tom Gruber. What is an Ontology? <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>.
- [Hsi95] I. Hsi, C. Potts: Towards Integrating Rationalistic and Ecological Design Methods for Interactive Systems. Research report, Georgia Institute of Technology, Graphics, Visualization & Usability Center, GIT-GVU-95-27, 1995.
- [Janssen93] C. Janssen, A. Weisbecker, J. Ziegler: Generating User Interfaces from Data Models and Dialogue Net Specifications. In: S. Ashlund, et.al. (eds.): Bridges between Worlds. Proceedings InterCHI'93 (Amsterdam, April 1993). New York: ACM Press, 1993, 418-423.

- [Johnson92] P. Johnson: Human-Computer Interaction. London: McGraw-Hill, 1992.
- [Johnson95] P. Johnson, H. Johnson, S. Wilson: Rapid Prototyping of User Interfaces Driven by Task Models. In: J. Carroll (ed.): Scenario-Based Design. London: John Wiley & Son, 1995, 209-246.
- [Lonczewski96] F. Lonczewski, S. Schreiber: The FUSE-System: An Integrated User Interface Design Environment. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 37-56.
- [Märting96] C. Märting: Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 57-74.
- [Myers90] B. A. Myers: A New Model for Handling Input. ACM Transactions on Information Systems 8 (1990), 3, 289-320.
- [Myers90b] B. A. Myers, et. al.: Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces. IEEE Computer 23 (1990), 11, 71-85.
- [Myers92] B. A. Myers, M. B. Rosson: Survey on User Interface Programming. In: P. Bauersfeld, J. Bennett, G. Lynch (eds.): Striking a Balance. Proceedings CHI'92 (Monterey, May 1992), New York: ACM Press, 1992, 195-202.
- [Myers95] B. A. Myers: User Interface Software Tools. ACM Transactions on Computer-Human Interaction 2 (1995), 1, 64-103.
- [Myers96] B. A. Myers, A. Ferreny, R. McDaniel, R. C. Miller, P. Doane, A. Mickish, A. Klimovitski: The Amulet V2.0 Reference Manual. Carnegie-Mellon University, School of Computer Science, Technical Report CMU-CS-95-166-R1, 1996.
- [Neches93] R. Neches, J. Foley, P. Szekely, P. Sukaviriya, P. Luo, S. Kovacevic, S. Hudson: Knowledgeable Development Environments Using Shared Design Models. In: W. Gray, W. Hefley, D. Murray (ed.): Proceedings of the 1993 International Workshop on Intelligent User Interfaces (Orlando, January 1993). New York: ACM Press, 1993, 63-70.
- [Olsen93] D. Olsen, J. Foley, S. Hudson, J. Miller, B. Myers: Research directions for user interface software tools. Behaviour & Information Technology 12 (1993), 2, 81-97.

- [Puerta94a] A. Puerta, P. Szekely: Model-based Interface Development. CHI'94 Tutorial Notes, 1994.
- [Puerta94b] A. Puerta, H. Eriksson, J. Gennari, M. Musen: Beyond Data Models for Automated User Interface Generation. In: G. Cockton, S. Draper, G. Weir (eds.): People and Computers IX. Proceedings British HCI'94 (Glasgow UK, August 1994). Cambridge: Cambridge University Press, 1994, 353-366.
- [Puerta96a] A. Puerta: The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 19-36.
- [Puerta96b] A. Puerta: Issues in Automatic Generation of User Interfaces in Model-Based Systems. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 323-325.
- [Reiterer94] H. Reiterer: A user interface design approach. In: K. Brunnstein, E. Raubold (eds.): Applications and Impacts, Information Processing '94. Proceedings of the IFIP 13th World Computer Congress (Hamburg, August 1994), Vol. 2, IFIP Transactions A-52, Amsterdam: North-Holland, 1994, 180-187.
- [Rumbaugh91] J. Rumbaugh, M. Blaha, W. Premerlain, F. Eddy, W. Lorensen: Objectoriented Modelling and Design. Englewood Cliffs: Prentice Hall, 1991.
- [Schlungbaum96a] E. Schlungbaum, T. Elwert: Automatic User Interface Generation from Declarative Models. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 3-18.
- [Schlungbaum96b] E. Schlungbaum, T. Elwert: Dialogue Graphs - a Formal and Visual Specification Technique for Dialogue Modelling. In: C.R. Roast and J.I. Siddiqi (eds): BCS-FACS Workshop on Formal Aspects of the Human Computer Interface, Sheffield Hallam University, 10-12 September 1996. Electronic Workshops in Computing, Springer-Verlag, Booklet ISBN: 3-540-76105-5, URL: <http://www.springer.co.uk/eWiC/Workshops/FAHCI.html>.
- [Schlungbaum96c] E. Schlungbaum: Individual User Interfaces and Model-based User Interface Software Tools. Research report, Georgia Institute of Technology, Graphics, Visualization & Usability Center, GIT-GVU-96-28, November 1996.

- [Schreiber95] S. Schreiber: The BOSS System: Coupling Visual Programming with Model Based Interface Design. In: F. Paterno (ed.): Interactive Systems: Design, Specification and Verification. Berlin: Springer, 1995, 161-179.
- [Storrs95] G. Storrs: The Notion of Task in Human-Computer Interaction. In: M. Kirby, A. Dix, J. Finlay (eds.): People and Computers X. Proceedings British HCI'95 (Huddersfield UK, August 1995). Cambridge: Cambridge University Press, 1995, 357-365.
- [Sukaviriya90] P. Sukaviriya, J. Foley: Coupling a UI Framework with Automatic Generation of Context-Sensitive Animated Help. In: Proceedings of the 3rd Annual Symposium on User Interface Software and Technology UIST'90 (Snowbird, October 1990). New York: ACM Press, 1990, 152-166.
- [Sukaviriya93a] P. Sukaviriya, J. Foley: Supporting Adaptive Interfaces in a Knowledge-based User Interface Environment. In: W. Gray, W. Hefley, D. Murray (ed.): Proceedings of the 1993 International Workshop on Intelligent User Interfaces (Orlando, January 1993). New York: ACM Press, 1993, 107-114.
- [Sukaviriya93b] P. Sukaviriya, J. Foley, T. Griffith: A Second Generation User Interface Design Environment. In: S. Ashlund, et.al. (eds.): Bridges between Worlds. Proceedings InterCHI'93 (Amsterdam, April 1993). New York: ACM Press, 1993, 375-382.
- [Sukaviriya95] P. Sukaviriya, J. Muthukumarasamy, M. Frank, J. Foley: A Model-Based User Interface Architecture: Enhancing a Runtime Environment with Declarative Knowledge. In: F. Paterno (ed.): Interactive Systems: Design, Specification and Verification. Berlin: Springer, 1995, 181-197.
- [Szekely93] P. Szekely, P. Luo, R. Neches: Beyond Interface Builders: Model-Based Interface Tools. In: S. Ashlund, et.al. (eds.): Bridges between Worlds. Proceedings InterCHI'93 (Amsterdam, April 1993). New York: ACM Press, 1993, 383-390.
- [Szekely96a] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, E. Salcher: Declarative interface models for user interface construction tools: the MASTERMIND approach. In: L. Bass, C. Unger (eds.): Engineering for Human-Computer Interaction. Proceedings of the IFIP TC2/WG2.7 working conference on engineering for human-computer interaction (Yellowstone Park, August 1995). London: Chapman & Hall, 1996, 120-150.

- [Szekely96b] P. Szekely: Retrospective and Challenges for Model-Based Interface Development. In: F. Bodart, J. Vanderdonckt (eds.): Design, Specification, and Verification of Interactive Systems. Berlin: Springer, 1996, 1-27.
- [UML96] Unified Modeling Language for Real-Time Systems. November, 1996.  
([http://www.rational.com/pst/tech\\_papers/uml\\_rt.html](http://www.rational.com/pst/tech_papers/uml_rt.html))
- [Walsh89] P. Walsh: Analysis for task object modelling (ATOM): towards a method of integrating task analysis with Jackson System Development for user interface software design. In: D. Diaper (ed.): Task Analysis for Human-Computer Interaction. Chichester: Ellis Horwood, 1989, 186-209.
- [Wiecha90] C. Wiecha, W. Bennett, S. Boies, J. Gould, S. Greene: ITS: A Tool for Rapidly Developing Interactive Applications. ACM Transactions on Information Systems 8 (1990), 3, 204-236.
- [Wilson93] S. Wilson, P. Johnson, C. Kelly, J. Cunningham, P. Markopoulos: Beyond Hacking: A Model Based Approach to User Interface Design. In: J. Alty, D. Diaper, S. Guest (eds.): People and Computers VIII. Proceedings British HCI'93 (Loughborough, September 1993). Cambridge: Cambridge University Press, 217-231.
- [Wilson94] S. Wilson, P. Johnson: From Work Tasks to Interactive System Designs (Tutorial Notes British HCI'94). Technical Report 693, Department of Computer Science, Queen Mary and Westfield College, 1994.
- [Wilson96a] S. Wilson, P. Johnson: Bridging the Generation Gap: From Work Tasks to User Interface Designs. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 77-93.
- [Wilson96b] S. Wilson: Reflections on Model-Based Design: Definitions and Challenges. In: J. Vanderdonckt (ed.): Computer-Aided Design of User Interfaces. Namur: Namur University Press, 1996, 327-333.