User Interface Declarative Models and Development Environments: A Survey

Paulo Pinheiro da Silva

Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, England, UK.
e-mail: pinheirp@cs.man.ac.uk

Abstract

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.

1 Introduction

The model-based user interface development technology aims to provide an environment where developers can design and implement user interfaces (UIs) in a professional and systematic way, more easily than when using traditional UI development tools. To achieve this aim, UIs are described through the use of declarative models. There are three major advantages that derive from the declarative user interface models (UIMs).

- They can provide a more abstract description of the UI than UI descriptions provided by the other UI development tools [51, 35];
- They facilitate the creation of methods to design and implement the UI in a systematic way since they offer capabilities: (1) to model user interfaces using different levels of abstraction; (2) to incrementally refine the models; and (3) to re-use UI specifications.
- They provide the infrastructure required to automate tasks related to the UI design and implementation processes [47].

A major disadvantage of UIMs is the complexity of the models and their notations, which are often hard to learn and use [29, 47]. However, it is expected that an appropriate environment should help to overcome the UIM's complexity, providing features such as graphical editors, assistants and design critics to support UI designers. In fact, the development of model-based user interface development environments (MB-UIDEs) is still challenging since some essential problems related to this technology are not completely solved.

- It is hard to demonstrate that UIMs describe the relevant aspects of the UI required to generate running user interfaces. In fact, there are few examples of running user interfaces generated from declarative UIMs [50, 43].
- The problem of how best to integrate UIs with their underlying applications is introduced in many papers [11, 12] but is not entirely addressed for running user interfaces generated by MB-UIDEs.
- There is no consensus as to which set of models is the most suitable set for describing user interfaces. Indeed, there is no consensus as to which aspects of user interfaces should be modelled.

After more than a decade, research into model-based user interface technologies is achieving a level of maturity that can lead to effective development of good quality UIs integrated with applications. Based on the potential benefits these technologies can provide to UI developers, this survey provides a review of these technologies summarising related information available from the literature. Relevant aspects of fourteen MB-UIDEs, as presented in Table 1, are compared and analysed throughout the paper. Details of how specific MB-UIDEs are implemented are not presented here, and nor are specific notations or tools.

MB-UIDE	References	Local
ADEPT	[27, 23, 52]	Queen Mary and Westfield College, UK
AME	[28]	Fachhochschule Augsburg, Germany
FUSE	[38, 39, 25]	Technische Universität München, Germany
GENIUS	[21]	University of Stuttgart, Germany
HUMANOID	[44, 46, 26]	University of Southern California, USA
JANUS	[2, 3]	Ruhr-Universität Bochum, Germany
ITS	[50, 51]	IBM T.J. Watson Research Center, USA
MASTERMIND	[47, 9, 43]	University Southern California, Georgia Inst. Tech., USA
MECANO	[32]	Stanford University, USA
MODI-D	[35, 33, 34]	Stanford University, USA
TADEUS	[13]	Universität Rostock, Germany
TEALLACH	[16]	U. Manchester, U. Glasgow, U. Napier, UK
TRIDENT	[5, 4, 6]	Facultés Universitaires Notre-Dame de la Paix, Belgium
UIDE	[24, 15, 14]	George Washington University, USA

Table 1: Surveyed MD-UIDEs.

This paper is structured as follows. Section 2 describes MB-UIDE's evolution and presents research efforts. Section 3 introduces a framework for comparing and analysing the architectural components of the UIMs. Section 4 describes the UI development process using a MB-UIDE. Section 5 presents how user interfaces are described through declarative models. Design guidelines are also introduced in this section. Section 6 describes the design environment through the tools used to model, generate and animate the model-based user interfaces. Conclusions are presented in Section 7.

2 Background

The literature contains many papers describing MB-UIDEs and their UIMs. The first generation of MB-UIDE appeared as improvements to the earlier user interface management systems (UIMSs) since they sought to execute user interfaces represented in a declarative way. The main aim of the MB-UIDEs of this generation was to provide a proper way to execute a UI from the UIM. Examples of the the first generation of MB-UIDEs are COUSIN [18], HUMANOID [44], MIKE [31], UIDE [24] and UofA* [42]. However, the UIMs of the first generation of MB-UIDEs did not provide a high-level of abstraction for the description of the UI. For instance, user interface aspects like layouts and widget customisation appeared early during the UI design process. Therefore, a new generation of MB-UIDEs appeared providing mechanisms for describing UIs at a higher level of abstraction [52]. Examples of the second generation of MB-UIDEs are ADEPT [27], AME [28], DIANE+ [48], FUSE [25], MASTERMIND [47], MECANO [32], MOBI-D [35], TADEUS [13], Teallach [16] and TRIDENT [5]. With MB-UIDEs of the second generation, developers have been able to specify, generate and execute user interfaces. Further, this second generation of MB-UIDE has a more diverse set of aims than previous one. Some MB-UIDEs are considering the use of computer-aided software engineering (CASE) tools and notations such as OMT [36] in their development environment. Others are aiming to achieve complete UI development.

Most of the papers describing these MB-UIDEs compare some of their features with other MB-UIDEs, showing the differences among them. However, they are focused more on introducing the new approach than introducing the MB-UIDE technology. There are a few papers that provide overviews of the MB-UIDE technology: Schlungbaum [37], Vanderdonckt [49] and Griffiths [17] provide comparisons among many MB-UIDEs, and Szekely [47] provides an excellent insight into what an MB-UIDE is.

3 User Interface Model Framework

User interfaces convey the output of applications and the input from application users. For this reason, UIs have to cope with the complexity of both the applications and the users. In terms of MB-UIDE's architectures, this problem of conciliating application complexity and user interaction complexity is reflected in parts that MB-UIDEs usually have several models describing different aspects of the UI. These models are referred to in this as *component models* or *models*. Table 2 presents the four models considered in the framework, also presenting which aspects of the user interface are described by each model.

Component Model	Abbrev.	Function
Application model	AM	Describes the properties of the application relevant to the UI.
Task-Dialogue model	TDM	Describes the tasks that users are able to perform using the application, as well as how the tasks are related to each other.
Abstract presentation model	APM	Provides a conceptual description of the structure and behaviour of the visual parts of the user interface. There the UI is described in terms abstract objects.
Concrete presentation model	CPM	Describes in details the visual parts of the user interfaces. There is explained how the UI is composed in terms of widgets.

Table 2: Component models of a user interface.

As the purpose of this framework is to provide a comparison among different UIMs, three points should be considered:

- Task models and dialogue models are classified within a single model called the *Task-dialogue model*. Both, task models and dialogue models describe the possible tasks that users can perform during the interaction with the application, but at different levels of abstraction. The reason for classifying them together is that UIMs often only have one of them. Further, the possible constructors of task and dialogue models may have similar roles.
- User models are supported in some UIMs (i.e. ADEPT, MECANO and TADEUS). Indeed, user models are important for the model-based user interface technologies since they can provide a way to model user interface preferences for specific users or groups of users. However, they are a challenging aspect of the UI not well-addressed in MB-UIDEs, and especially not clearly described in the literature. Moreover, in those UIMs that have a user model it appears that the user model can be replaced by design guidelines. In fact, design guidelines usually contain user preferences that can be considered as a model of a group of users.
- Platform models, or environment models, are not contemplated in the framework for the same reasons that user models are not contemplated.

Models are composed of constructors. Table 3 shows the constructors considered in the framework. The table also shows a possible distribution of these constructors into the component models, a concise description of each constructor, and abbreviations for future reference. The distribution of the constructors into component models, as presented in Table 3, helps to clarify their function in the framework. Definitions of application model constructors are partially extracted from UML [8]. Definitions of task model constructors are partially extracted from Johnson [22]. Definitions of abstract and concrete interaction objects are partially extracted from Bodart and Vanderdonckt [6].

Comp. model	Constructor	Abbrev.	Function
AM	class	CLASS	An object type defined in terms of attributes, operations and relationships.
	attribute	ATTR	A property of the thing modelled by the objects of a class.
	operation	OPER	A service provided by the object of a specific class.
	relationship	RELAT	A connection among classes.
TDM	task	TASK	An activity that changes the state of specific objects, leading to the achievement of a goal. Tasks can be defined at different levels of abstraction, which means that a task can be a sub-task of an abstract class.
	goal	GOAL	A state to be achieved by the execution of a task.
	action	ACTION	A behaviour that can be executed. Actions are the most concrete tasks.
	sequencing	SEQ	The temporal order that sub-tasks and actions must respect for carrying out the related high-level tasks.
	task pre-condition	PRE	Conditions in terms of object states that must be respected before the execution of a task or an action.
	task post-condition	POST	Conditions in terms of object states that must be respected after the execution of a task or an action.
APM	view	VIEW	A collection of AIO's logically grouped to deal with the inputs and outputs of a task.
	abstract interaction object	AIO	A user interface object without any graphical representation and independent of any environment.
CPM	window	WINDOW	A visible and manipulable representation of a a view.
	concrete interaction object	CIO	A visible and manipulable user interface object that can be used to input/output information related to user's interactive tasks.
	layout	LAY	An algorithm that provides the placement of CIOs in windows.

Table 3: User interface model constructors.

One point that should be considered in terms of constructors is that MB-UIDEs do not need to have all constructors presented in the framework. Further, constructors can be distributed in a different manner from that proposed in Table 3.

4 User Interface Development in a MB-UIDE

The UI development process is normally an incremental process in a MB-UIDE. User interface design and implementation can easily be repeated however many times are required to refine the UI specification and code. In reality, some MB-UIDEs are not flexible enough in terms of code refinement. Considering the UI development process, two distinct subprocesses can be identified. The first one is the UI design, that results in the creation of a UIM. The second one is the UI implementation, that results in an executable UI. Section 4.1 presents an overview of how possible UI design processes in a MB-UIDE. Section 4.2 presents how parts of the UI design process can be automated. Section 4.3 presents an overview of the UI implementation process.

4.1 User Interface Design Process

In a MB-UIDE, the UI design is the process of creating and refining the UIM. As stated in Section 3, there is not agreement on which set of models are the best for describing UIs in a declarative manner. In terms of UI design, there is also a lack of agreement as to which is the best method for UI modelling.

According to Figure 1[1], some modelling tools can be provided by MB-UIDEs for editing the models, and modelling assistants can be provided to support UI developers. These modelling tools usually provide a graphical environment that may facilitate the complex work of constructing UIMs. It is expected that these modelling tools can prevent UI developers for worrying about details of the models and their notation, focusing their attention on the design of the UI. Additionally, some MB-UIDEs have modelling assistants that can perform some functions, such as model checking, that provide feedback to developers about the design process.

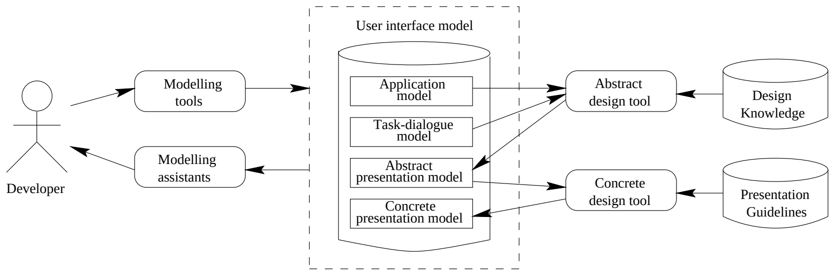

Figure 1: The user interface design in a MB-UIDE.

MB-UIDEs based on textual UIMs optionally may not require any special editor or assistant. This may be an excellent approach for expert developers that could build and refine models using any text editor. In fact, MB-UIDEs based on textual UIMs that offer model editors and assistants give freedom to developers to use or not the model development environment provided. The problem of MB-UIDEs based on textual models are that they may not offer any special facilities for model management. In this case, the cost of constructing UIM descriptions might be higher than the benefits provided by such environments, specially for non-expert developers.

There are MB-UIDEs that use existing graphical editors and CASE tools

[1]Traditional diagram describing the user interface development process in MB-UIDEs, presented in Szekely [45], Schlungbaum [37] and Griffiths *et al.* [16].

as their model editors. In this case, the problem is that tools could not accept modifications to accommodate specific requirements for editing UIMs.

As UIMs can describe UIs at different levels of abstraction, it is expected that the design process should be incremental. Thus, UIs could initially be described by a very abstract model that can gradually be transformed into a concrete model. Considering this iterative design process, developers can edit the models using the modelling tools, and check the model using the design assistants, until the UIM reaches a point where the relevant details are modelled into the UIM. Further, developers can at any time return to the MB-UIDE to refine the model, even after the implementation of the UI. The problem, in this case, is that the UI should be modified only through the MB-UIDE since modifications not described in the UIM are obviously not regenerated, in the new version of the UI.

4.2 Automated Tasks in User Interface Design Process

Some papers claim that the real advantage of the model-based UI technologies is the support they provide to automate the UI design [30]. Indeed, it is a powerful characteristic of the UIMs that they can describe UIs at different levels of abstraction.

Figure 1 shows how UI design automation fits into the development activity. An *abstract design tool* can generate the abstract presentation model from application models or task-dialogue models, using a design knowledge database to supply information required during the UI design process. Additionally, a *concrete design tool* can generate the concrete presentation model from the abstract presentation model, and using a design guideline database. The design guidelines are not part of the user interface model, but they are part of the MB-UIDE.

Most of the research related with UI development concerns the "look and feel" aspects of the UI. For this reason, there are many well-known guidelines concerning the presentation of the UI [40]. On the other hand, there are some research efforts that analyse how to model tasks during human-computer interaction. However, the guidelines provided by these studies are not established enough to be used as a proper design knowledge database [52]. Therefore, part of the automated UI design process related with the task-dialogue model is affected by this lack of well-established task modelling guidelines.

At the same time such automated design facilitated the work of UI developers, it also creates a new problem: how UI developers can interfere in this automated process to design UIs with different characteristics to those provided by design knowledge design guideline databases [47].

4.3 User Interface Implementation Process

Figure 2 illustrates three approaches to generating and executing a user interface, in the context of a MB-UIDE. In the first approach shown in Figure 2a, the source code of the user interface is generated based on the toolkit class library. In this approach, the MB-UIDE generates the source code of the user interface, and sometimes it generates the skeleton of the application. In the second approach shown in Figure 2b, the UI is executed by the UIMS runtime system linked with the application. A UIMS input-file generator is required, in this case, to convert the UIM into the UIMS input-file format. In the third approach shown in Figure 2c, the application can interpret the UIM directly due to the MB-UIDE runtime system being linked to the application.

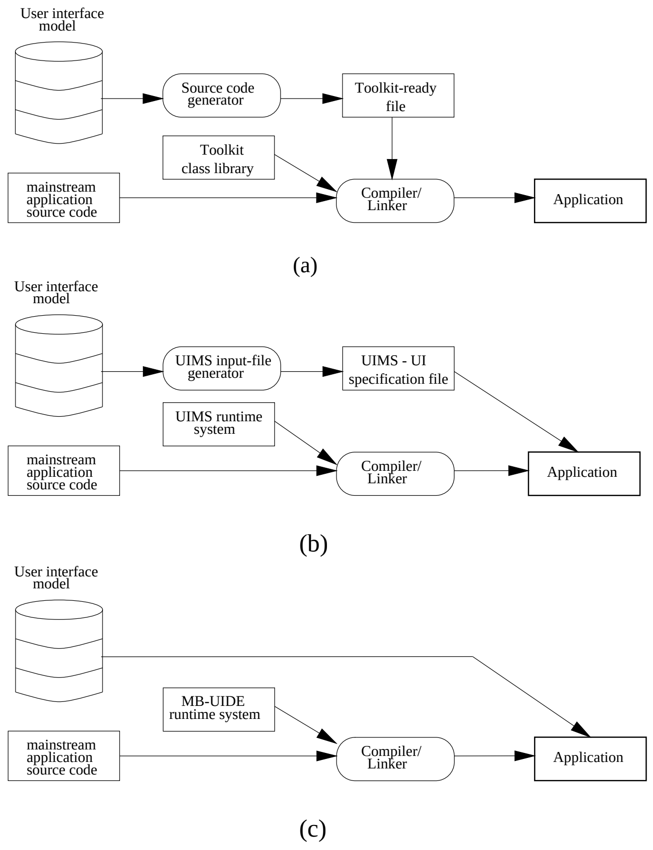

Figure 2: The three approaches for implementing a user interface in a MB-UIDE.

The UI implemented using the first approach has the advantage that it is entirely coded inside the application, providing a natural integration between the UI and the application. However, the UI produced using the first implementation approach is more static than the UIs produced using the other implementation approaches, which are able to be reconfigured more easily at runtime. The UI implemented using the third implementation approach, however, tends to have a performance worse than the UIs implemented using the first and second approaches since it is expensive to interpret the UIM at runtime.

The implementation tools in Figure 2 can be classified into two categories, defined as follows.

- *UI generators*: These are tools that make the application independent of the UIM. In this case, source code generators (Figure 2a) and UIMS input-file generators (Figure 2b) are UI generators. For instance, MB-UIDEs that can execute a UI directly from the UIM do not have UI generators.
- *UI runtime systems*: They are those tools that execute the user interface when the application is running. In this case, the application itself in the first implementation approach (Figure 2a), the UIMS runtime system (Figure 2b) and the MB-UIDE runtime systems are UI runtime systems. UI runtime systems are essential tools for MB-UIDEs.

One relevant observation concerns MB-UIDE terminology. The term *UI generation* can be used to refer to the process of generating the input-file of the UI runtime system, as described above, or to refer to the process of generating the concrete presentation model from abstract presentation models, application models and task-dialogue models. In this paper, we use the term *UI generation* to refer to the first process.

5 Declarative Models

The UIM is certainly the most important element of a MB-UIDE. In fact, the UI is designed in terms of the UIM, generated from the UIM and sometimes executed from the UIM. Therefore, it is important to understand how UIMs are composed in terms of their models and constructors. Further, it is also important to know what notations are used to represent these models.

5.1 Models

Table 4 presents the models of the MB-UIDEs in terms of the framework. The terms used in the table are those used in the literature to identify the models.

The application model is present in every user interface model. In fact, the MB-UIDE technology appeared initially as an improvement in the user interface management systems (UIMS), where a clear distinction between the user interface and the mainstream application is required.

The presentation model, like the application model, is always considered in declarative models. However, there are MB-UIDEs that do not have an abstract presentation model such as MASTERMIND and MECANO. In other MB-UIDEs such as HUMANOID, TADEUS and Teallach, it is not clear the distinction between the abstract and concrete presentation models. In this last case, designers normally have the flexibility to gradually refine the presentation description from an abstract model to a concrete model.

MB-UIDE	Application model	Task-Dialogue model
ADEPT	problem domain	task model
AME	application model	OOD
FUSE	problem domain model	task model
HUMANOID	application semantics design	manipulation, sequencing, action side effects
JANUS	problem domain	(none)
ITS	data pool	control specification in dialog
MASTERMIND	application model	task model
MECANO	domain model	user task model/dialogue model
TADEUS	problem domain model	task model/navigation dialogue
TEALLACH	domain model	task model
TRIDENT	application model	task model

MB-UIDE	Abstract presentation model	Concrete presentation model
ADEPT	abstract user interface model	prototype interface
AME	OOA	prototype
FUSE	logical UI	UI
HUMANOID	presentation	presentation
JANUS	(not surveyed)	(not surveyed)
ITS	frame specification in dialog	style specification
MASTERMIND	(none)	presentation model
MECANO	(none)	presentation model
TADEUS	processing dialogue	processing dialogue
TEALLACH	presentation model	presentation model
TRIDENT	(not surveyed)	presentation model

Table 4: MB-UIDE's component models.

Finally, declarative models also consider the use of a task-dialogue model to describe the possible interactions between users and applications using the presentation and application models. Some MB-UIDEs describe these interactions at a dialogue-level such as HUMANOID, MASTERMIND and ITS. Other MB-UIDEs, especially those developed after ADEPT, describe the interactions at a task-level, more abstract than the dialogue-level. However, there are MB-UIDEs such as MECANO and TADEUS that describe the possible interactions at both dialogue and task levels.

5.2 Constructors

Having identified the models, we need to identify the model constructors. As we did for models, Table 5 presents the model constructors using the terminology available in the literature for the specific proposals. The column *constructor* refers to the abbreviation for constructors introduced in the framework (Table 3). Constructors not present in Table 5 are not used in the specific system, or at least were not identified in the literature.

5.3 Model Notations

While Section 5.1 has indicated what models are present in different proposals, the semantics of the individual models in different contexts has not yet been touched on. Table 6 shows the several different notations used by the models of

MB-UIDE	Constructor	Name
Adept	TASK	task
	GOAL	goal
	SEQ	ordering operator + sequencing
	AIO	user interface object
	CIO	UIO
AME	CLASS	OOA class
	ATTR	slot/OOA attributes
	OPER	OOA operation
	RELAT	relation type
	ACTION	behaviour
	AIO	AIO
	WINDOW	OOD class
	CIO	CIO
	LAY	layout-method
Humanoid	CLASS	object type
	ATTR	slot
	OPER	command
	TASK	data flow constraints
	GOAL	goal
	ACT	behaviour
	SEQ	guard slots' constraints, triggers
	PRE	sequential pre-condition
	POST	action side-effect
	AIO	template
	WINDOW	display
	CIO	display, interaction technique
	LAY	layout
Janus	CLASS	class
	ATTR	attribute
	OPER	operation
	RELAT	association, aggregation
	CIO	interaction object
	WINDOW	dialog widow (UIView)
ITS	CLASS	data table
	ATTR	field
	VIEW	frame
	AIO	dialog object
	EVENT	event
	ACT	action
	WINDOW	root unit
	CIO	unit
	LAY	style attribute
Mastermind	CLASS	interface
	ATTR	attribute
	OPER	method
	TASK	task
	GOAL	goal
	SEQU	connection type
	WINDOW	presentation
	CIO	presentation part
	LAY	guides, grids, conditionals
Teallach	CLASS	class
	ATTR	attribute
	OPER	operation
	TASK	task
	SEQ	task temporal relation
	VIEW	free container
	AIO	AIO
	WINDOW	window
	CIO	CIO

Table 5: MB-UIDE's constructors.

different proposals.

We notice in Table 6 that there are UIs entirely described by models using a single notation. In general, these notations have been developed specifically for the MB-UIDE. They can be completely new as in ITS's Style rules [50, 51], or they can be extensions of other notations, as in MASTERMIND's MDL that is an extension of CORBA IDL [41]. The use of a single notation can be useful to describe how the models collaborate with each other. However, specially due to the requirement of graphical notations, UI models tend to use different notations. For example, JANUS, TADEUS, TRIDENT, Teallach, and Adept models use more than one notation. It is not feasible to provide a categorisation of these UIMs in terms of their notations here because they tend to be specific to each MB-UIDE. For instance, there are many MB-UIDEs that use a hierarchical task notation to model their task-dialogue models, however, the notation may not be precisely formalised, as in Teallach.

The use of standard notations appears to be an aim. For instance, MASTERMIND's notation is based on CORBA IDL, and AME and TADEUS apply OMT [36] in some of their component models, since these are notations available for describing other parts of the application. In fact, OMT can be used during the analysis and design of the mainstream application, and CORBA IDL can be used during the implementation of the mainstream application.

A comprehensive explanation of the semantics of these notations is outside of the scope of this survey. The references required to find out more about these notations are also provided in Table 6 .

MB-UIDE	Notation	Models
ADEPT	task knowledge structures (TKS) [20]	TDM
	LOTOS [7]	TDM
	Communicating Sequential Process (CSP) [19]	TDM, APM
AME	OOA/OOD [10]	AM
	OMT [36]	AM
FUSE	algebraic specification [53]	AM
	HTA [22]	TDM, UM
	Hierarchic Interaction graph Template (HTI)	APM, CPM
HUMANOID	uses a single notation which was not specified	all models
JANUS	JANUS Definition Language (extended CORBA IDL and ODMG ODL)	AM
ITS	Style rule [50, 51]	all models
MASTERMIND	MDL [43] (extended CORBA IDL [41])	all models
MECANO	MIMIC [32] (extended C++)	all models
MOBI-D	MIMIC (see MECANO's notation)	all models
TADEUS	specialised HTA	TDM
	OMT [36]	AM, UM
	Dialogue Graph (specialised Petri net)	TDM
TEALLACH	hierarchical tree with state objects	TDM
	hierarchical tree	AM, APM, CPM
TRIDENT	Entity-Relationship-Attribute (ERA)	AM
	Activity Chaining Graph (ACG)	TDM, APM, CPM

Table 6: Model notations.

5.4 Model Integration

Models are integrated, although it is not unusual for the literature to be unclear on the precise nature of such integration. Indeed, Puerta and Eisenstein [34] said that there is a lack of understanding of UIM integration, denoting this problem as *the mapping problem*.

One strategy to finding out how these models are integrated is through the compilation of the relationships of constructors in different component models. Table 7 shows some of those inter-model relationships, relating the relationship constructors with their multiplicity. The multiplicity between brackets is described in UML notation [8]. Additionally, the figure in Table 7 shows graphically how the models are related to each other in the MB-UIDEs.

The presentation model can be considered as a set composed of the APM, the CPM, and the relationships between the APM and CPM. Our strategy to analyse the figure in Table 7 is based on the identification of how AMs relate to presentation models. There are two approaches to relating AMs and presentation models. The first and most frequent approach is creating direct relationships between the two models, such as in HUMANOID, JANUS, ITS and MECANO. The second approach is using the TDM. In this case, there are relationships between the AM and the TDM, and between the TDM and the presentation model, such as in MASTERMIND and Teallach.

In AME and ADEPT, for instance, there are relationships between the APM and the TDM, but these relationships do not provide a link with the AM that is directly linked with the APM. In this case, the link is more between the AM and the TDM than between the AM and the presentation model.

MB-UIDE	Inter-model relationship	
	Constructor	Constructor
ADEPT	ACTION (1)	AIO (*)
AME	CLASS (1)	AIO (1..*)
	CLASS (1)	WINDOW (0..1)
	ATTR (1)	AIO (1)
	WINDOW (1)	AIO (1..*)
	ACTION (1)	AIO (1)
GENIUS	VIEW (1)	WINDOW (1)
	AIO (1)	CIO (1)
HUMANOID	CLASS (1)	CIO (1)
	AIO (1)	CIO (1)
JANUS	WINDOW (1)	CLASS (*)
	AIO (1)	ATTR (1)
ITS	VIEW (1)	ATTR (*)
	CLASS (1)	AIO (*)
	AIO (1)	CIO (1..*)
MASTERMIND	TASK (1)	OPER (0..1)
	TASK (1)	CIO (0..1)
	root TASK (1)	WINDOW (1)
MECANO	WINDOW (1)	CLASS (1)
	AIO (1)	ATTR (1)
TEALLACH	TASK (1)	CLASS (0..*)
	TASK (1)	AIO (0..*)
	TASK (1)	VIEW (0..1)
	WINDOW (1)	AIO (0..*)
	AIO (1)	CIO (1..*)
TRIDENT	WINDOW (1..*)	VIEW (1)
	AIO (1)	CIO (0..*)

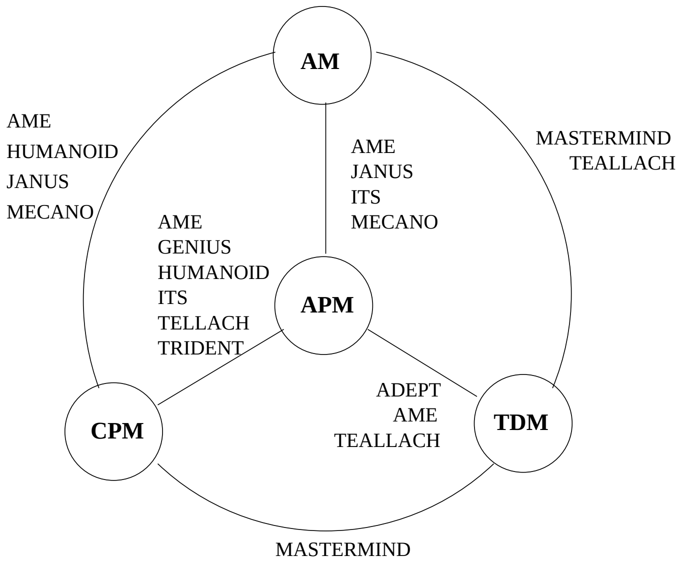

Table 7: Discrete and graphical representation of the inter-model relationships.

6 Environments

MB-UIDEs are composed of tools where users can perform the tasks required to design and generate a user interface, as described in Section 4. Thus, a MB-UIDE architecture can be explained in terms of its tools. In fact, some development environments provide a monolithic tool with which developers perform their tasks. Other environments provide distinct tools where developers perform specific tasks, leading to a complete development of the UI. There is a third kind of environment where developers perform part of their tasks in tools not especially developed for the MB-UIDE, such as CASE tools, and the other part of their tasks in tools especially developed for the MB-UIDE. This section analyses MB-UIDE architectures through a comparison of their tools.

6.1 Design Environment

UI models are generally complex, leading to the modelling process also being a complex task. Thus, modelling tools are usually provided to help the designer to model the user interface. Table 8 produces a classification of environment tools according to our tool classification. It is important to observe that some MB-UIDEs are composed of tools that are responsible for more than one function, then they are classified in more than one category.

MB-UIDE	Modelling Editors	Modelling Assistant
ADEPT	Task model editor AUI editor	Interface generator object browser
AME	OODevelopTool ODE-editor	code generator layout generator
FUSE	FIRE	FLUID
GENIUS	Model Editing Tool	Model Refinement Tool
HUMANOID	(none)	(none)
JANUS	Paradigm Plus (OO CASE tool) Together C++ (OO CASE tool)	(not surveyed)
ITS	not specified	(none)
MASTERMIND	Application Modeling Suite Task Modeling Suite Presentation Modeling Suite	Dialog Critics
TADEUS	Tadeus	Tadeus
TEALLACH	Teallach	code generator

Table 8: Design environment tools.

6.2 Implementation Environment

UI implementation is a key activity in the use of a MB-UIDE. To generate the UI, however, the MB-UIDE depends not only on the UIM, but also on the environment that is being considered. As discussed in Section 4, a MB-UIDE can implement a run-time interpreter for the UIM, generate code for an existing UIMS, or generate code that uses a specific toolkit. Therefore, there are basically three alternatives that can be considered for generating the user interface.

Table 9 summarises the user interface generation tools of the MB-UIDEs. There we see that some MB-UIDEs are based on a UIMS (e.g. TADEUS). Other MB-UIDEs generate code for specific toolkits (e.g. AME, FUSE and MASTERMIND generate code for C++, and Teallach generates code for Java). The others, however, implement the whole environment (e.g. ITS's dialog manager, FUSE's BOSS). One interesting approach is that used by AME and MASTERMIND that provide UI prototyping using UIMSs, but that generate code for C++. That way, these MB-UIDEs can offer to their users the benefits of alternative approaches to generating a user interface.

MB-UIDE	UI Generator	UI Runtime System
ADEPT	interface builder	interpreter
AME	AME/C++ code generator Open Interface code generator not specified	application code Open Interface (UIMS) KAPPA-PC runtime system
FUSE	BOSS[38, 39]	(not surveyed)
GENIUS	(not surveyed)	runtime system
HUMANOID	(none)	Humanoid runtime system
JANUS	C++ code generator	application code
ITS	(none)	UI executed from the UIM
MASTERMIND	Mastermind Prototyping Support C++ code generator	AMULET [30] (UIMS) application code
TADEUS	not specified	ISA Dialog Manager (UIMS)
TEALLACH	Java code generator	application code (Swing Toolkit)

Table 9: Implementation environment tools.

The implementation environment can also be composed of *advisors* and *documentation generators*. However, these tools are not discussed in this survey.

7 Conclusions

MB-UIDEs seek to provide a setting within which a collection of complementary declarative models that can be used as a description of UI functionalities. This survey has compared the models and tools provided by 14 MB-UIDEs. Declarative models, model constructors and model notations were presented using a comparative framework. Design and implementation tools were identified.

The MB-UIDE technology is just now becoming stable enough to be commercialised as products e.g. Systemator [1]. Indeed, this is the result of practical experiences with this technology, e.g. ITS was used by IBM to produce the UI of the visitor information system of EXPO'92 [50, 51], and FUSE has been used by Siemens to simulate an ISDN telephone.

However, there are many aspects of MB-UIDE technology that must be studied in order to increase the acceptance of MB-UIDEs at the level of other specialised UI development tools [29].

- *Mapping between models.* The aspects of UIs that it is relevant to model in UIMs are well-understood. In fact, most of the surveyed MB-UIDEs provide in some way a similar set of UI aspects that they can describe, as observed in Table 4. However, it is unclear how best to model the relationships between the constructors of the models used to describe UIs, as observed in Table 7.

- *UIM post-editing problem.* Automated generated drafts of UI designs may be manually refined in order to generate final designs. However, manual refinements to generated designs are lost when developers regenerate other draft designs. Therefore, it is still a problem how best to cope with post-editing refinements.
- *Standard notations for UIMs.* The use of a standard notation may be useful in order to describe different UIMs using a common set of constructors. In fact, these constructors may facilitate the comparison and the reuse of UIMs and their MB-UIDEs. For instance, the reuse of UIMs may be difficult these days since they are based on several notations, as presented in Table 6. Further, the reuse of UIMs can be essential for make MB-UIDEs scalable for real applications.

Acknowledgements. The author would like to thank Norman W. Paton and the anonymous reviewers of this paper for their valuable comments. The author is sponsored by Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (Brazil) – Grant 200153/98-6.

References

[1] Genera AS. Systemator. http://www.genera.no.

[2] H. Balzert. From OOA to GUI – The JANUS-System. In *Proceedings of INTERACT'95*, pages 319–324, London, UK, June 1995. Chapman & Hall.

[3] H. Balzert, F. Hofmann, V. Kruschinski, and C. Niemann. The JANUS Application Development Environment — Generating More than the User Interface. In *Computer-Aided Design of User Interfaces*, pages 183–206, Namur, Belgium, 1996. Namur University Press.

[4] F. Bodart, A. Hennebert, J. Leheureux, I. Provot, B. Sacre, and J. Vanderdonckt. Towards a Systematic Building of Software Architectures: the TRIDENT Methodological Guide. In *Design, Specification and Verification of Interactive Systems*, pages 262–278, Vienna, 1995. Springer.

[5] F. Bodart, A. Hennebert, J. Leheureux, I. Provot, and J. Vanderdonckt. A Model-Based Approach to Presentation: A Continuum from Task Analysis to Prototype. In *Proceedings of DSV-IS'94*, pages 25–39, Bocca di Magra, June 1994.

[6] F. Bodart and J. Vanderdonckt. Widget Standardisation Through Abstract Interaction Objects. In *Advances in Applied Ergonomics*, pages 300–305, Istanbul - West Lafayette, May 1996. USA Publishing.

[7] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Network ISDN Systems*, 14(1), 1987.

[8] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, MA, 1999.

[9] T. Browne, D. Dávila, S. Rugaber, and K. Stirewalt. *Formal Methods in Human-Computer Interaction*, chapter Using Declarative Descriptions to Model User Interfaces with MASTERMIND. Springer-Verlag, 1997.

[10] P. Coad and E. Yourdon. *Object-Oriented Design*. Prentice-Hall, 1991.

[11] J. Coutaz and R. Taylor. Introduction to the Workshop on Software Engineering and Human-Computer Interaction: Joint Research Issues. In *Proceedings of the Software Engineering and Human-Computer Interaction'94*, volume 896 of *Lecture Notes In Computer Science*, pages 1–3, Berlin, May 1995. Springer-Verlag.

[12] B. Curtis and B. Hefley. A WIMP No More – The Maturing of User Interface Engineering. *ACM Interactions*, 1(1):22–34, 1994.

[13] T. Elwert and E. Schlungbaum. Modelling and Generation of Graphical User Interfaces in the TADEUS Approach. In *Designing, Specification and Verification of Interactive Systems*, pages 193–208, Vienna, 1995. Springer.

[14] J. Foley. History, Results and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based Systems for User Interface Design and Implementation. In *Proceedings of DSV-IS'94*, pages 3–14, Vienna, 1995. Springer-Verlag.

[15] J. Foley, W. Kim, S. Kovacevic, and K. Murray. UIDE – An Intelligent User Interface Design Environment. In *Intelligent User Interfaces*, pages 339–384. Addison-Wesley, ACM Press, 1991.

[16] T. Griffiths, P. Barclay, J. McKirdy, N. Paton, P. Gray, J. Kennedy, R. Cooper, C. Goble, A. West, and M. Smyth. Teallach: A Model-Based User Interface Development Environment for Object Databases. In *Proceedings of UIDIS'99*, pages 86–96, Edinburgh, UK, September 1999. IEEE Press.

[17] T. Griffiths, J. McKirdy, G. Forrester, N. Paton, J. Kennedy, P. Barclay, R. Cooper, C. Goble, and P. Gray. Exploiting Model-Based Techniques for User Interfaces to Database. In *Proceedings of Visual Database Systems (VDB) 4*, pages 21–46, Italy, May 1998. Chapman & Hall.

[18] P. Hayes, P. Szekely, and R. Lerner. Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. In *Proceedings of SIGCHI'85*, pages 169–175. Addison-Wesley, April 1985.

[19] C. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[20] R. Jacob. A Specification Language for Direct Manipulation User Interfaces. *ACM Transactions on Graphics*, 5(4):283–317, October 1986.

[21] C. Janssen, A. Weisbecker, and J. Ziegler. Generating User Interfaces from Data Models and Dialogue Net Specifications. In *Proceedings of InterCHI'93*, pages 418–423, New York, NY, 1993. ACM Press.

[22] P. Johnson. *Human Computer Interaction: Psychology, Task Analysis and Software Engineering*. McGraw-Hill, Maidenhead, UK, 1992.

[23] P. Johnson, H. Johnson, and S. Wilson. Rapid Prototyping of User Interfaces Driven by Task Models. In *Scenario-Based Design*, pages 209–246, London, UK, 1995. John Wiley.

[24] W. Kim and J. Foley. DON: User Interface Presentation Design Assistant. In *Proceedings of UIST'90*, pages 10–20. ACM Press, October 1990.

[25] F. Lonczewski and S. Schreiber. The FUSE-System: an Integrated User Interface Desgin Environment. In *Computer-Aided Design of User Interfaces*, pages 37–56, Namur, Belgium, 1996. Namur University Press.

[26] P. Luo, P. Szekely, and R. Neches. Management of interface design in HUMANOID. In *Proceedings of InterCHI'93*, pages 107–114, April 1993.

[27] P. Markopoulos, J. Pycock, S. Wilson, and P. Johnson. Adept – A task based design environment. In *Proceedings of the 25th Hawaii International Conference on System Sciences*, pages 587–596. IEEE Computer Society Press, 1992.

[28] C. Märtin. Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In *Computer-Aided Design of User Interfaces*, pages 57–74, Namur, Belgium, 1996. Namur University Press.

[29] B. Myers. User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 2(1):64–103, March 1995.

[30] B. Myers, R. McDaniel, R. Miller, A. Ferrency, A. Faulring, B. Kyle, A. Mickish, A. Klimovitsky, and P. Doane. The Amulet Environment: New Models for Effective User Interface Software Development. *IEEE Transactions on Software Engineering*, 23(6):346–365, June 1997.

[31] D. Olsen. A Programming Language Basis for User Interface Management. In *Proceedings of SIGCHI'89*, pages 171–176, May 1989.

[32] A. Puerta. The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development. In *Computer-Aided Design of User Interfaces*, pages 19–36, Namur, Belgium, 1996. Namur University Press.

[33] A. Puerta and J. Eisenstein. Interactively Mapping Task Models to Interfaces in MOBI-D. In *Design, Specification and Verification of Interactive Systems*, pages 261–273, Abingdon, UK, June 1998.

[34] A. Puerta and J. Eisenstein. Towards a General Computational Framework fo Model-Based Interface Development Systems. In *Proceedings of IUI'99*, Los Angeles, CA, January 1999. (to be published).

[35] A. Puerta and D. Maulsby. Management of Interface Design Knowledge with MODI-D. In *Proceedings of IUI'97*, pages 249–252, Orlando, FL, January 1997.

[36] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, Englewood Cliffs, NJ, 1991.

[37] E. Schlungbaum. Model-Based User Interface Software Tools - Current State of Declarative Models. Technical Report 96-30, Graphics, Visualization and Usability Center, Georgia Institute of Technology, 1996.

[38] S. Schreiber. Specification and Generation od User Interfaces with the BOSS-System. In *Proceedings of EWHCI'94*, volume 876 of *Lecture Notes in Computer Sciences*, pages 107–120, Berlin, 1994. Springer-Verlag.

[39] S. Schreiber. The BOSS System: Coupling Visual Programming with Model Based Interface Design. In *Proceedings of DSV-IS'94*, Focus on Computer Graphics, pages 161–179, Berlin, 1995. Springer-Verlag.

[40] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, Reading, MA, second edition, 1992.

[41] J. Siegel. *CORBA: Fundamentals and Programming.* John Wiley, New York, NY, 1996.

[42] G. Singh and M. Green. A high-level user interface management system. In *Proceedings of SIGCHI'89*, pages 133–138, May 1989.

[43] K. Stirewalt. *Automatic Generation of Interactive Systems from Declarative Models.* PhD thesis, Georgia Institute of Technology, December 1997.

[44] P. Szekely. Template-Based Mapping of Application Data to Interactive Displays. In *Proceedings of UIST'90*, pages 1–9. ACM Press, October 1990.

[45] P. Szekely. Retrospective and Challenges for Model-Bases Interface Development. In *Computer-Aided Design of User Interfaces*, pages xxi–xliv, Namur, Belgium, 1996. Namur University Press.

[46] P. Szekely, P. Luo, and R. Neches. Facilitating the Exploration of Interface Design Alternatives: The HUMANOID Model of Interface Design. In *Proceedings of SIGCHI'92*, pages 507–515, May 1992.

[47] P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, and E. Salcher. Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. In *Engineering for Human-Computer Interaction*, pages 120–150, London, UK, 1996. Chapman & Hall.

[48] J. Tarby and M. Barthet. The DIANE+ Method. In *Computer-Aided Design of User Interfaces*, pages 95–119, Namur, Belgium, 1996. Namur University Press.

[49] J. Vanderdonckt. *Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, July 1997.

[50] C. Wiecha, W. Bennett, S. Boies, J. Gould, and S. Green. ITS: A Tool for Rapidly Developing Interactive Applications. *ACM Transactions on Information Systems*, 8(3):204–236, July 1990.

[51] C. Wiecha and S. Boies. Generating user interfaces: principles and use of ITS style rules. In *Proceedings of UIST'90*, pages 21–30. ACM Press, October 1990.

[52] S. Wilson and P. Johnson. Bridging the Generation Gap: From Work Tasks to User Interface Designs. In *Computer-Aided Design of User Interfaces*, pages 77–94, Namur, Belgium, 1996. Namur University Press.

[53] M. Wirsing. Algebraic Specification. In *Handbook of Theoretical Computer Science*, pages 676–788. North Holland, 1990.