Chapter 1
Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool

Silvia Abrahão[1], **Emilio Iborra**[2] **and Jean Vanderdonckt**[3]

[1] Department of Information Systems and Computation,
Valencia University of Technology, Spain, e-mail: sabrahao@dsic.upv.es
[2] Ami2 – Ambiental Intelligence & Interaction, Spain
[3] Belgian Lab. of Computer-Human Interaction (BCHI),
Université Catholique de Louvain, Belgium

Abstract Model-driven architecture (MDA) has recently attracted the interest of both the research community and industry corporations. It specifies an automated process for developing interactive applications from high-level models to code generation. This approach can play a key role in the fields of software engineering (SE) and human-computer interaction (HCI). Although there are some MDA-compliant methods for developing user interfaces, none of them explicitly integrates usability engineering with user interface engineering. This chapter addresses this issue by showing how the usability of user interfaces that are generated automatically by an industrial MDA-compliant CASE tool can be assessed. The goal is to investigate whether MDA-compliant methods improve software usability through model transformations. To accomplish this, two usability evaluations were conducted in the code model (final user interface). Results showed that the usability problems identified at this level provide valuable feedback on the improvement of platform-independent models (PIM) and platform-specific models (PSM) supporting the notion of *usability produced by construction*.

1.1 Introduction

The object management group (OMG) launched an initiative called model-driven architecture (MDA) to support the development of large, complex, interactive software applications providing a standardized architecture with the following features (MDA 2005):

- Interactive applications can easily evolve to address constantly evolving user requirements
- Old, current, and new technologies can be harmonized
- Business logic can be maintained constantly or can evolve independently of technological changes, or of the rest of the interactive application
- Legacy systems can be integrated and unified with new systems

E. Law et al. (eds.), *Maturing Usability.*

© Springer-Verlag London Limited 2008

In this approach, models are applied in all the steps of development up to a target computing platform. This provides a complete software production process where model transformation at different levels of abstraction becomes the basic strategy for obtaining a software application from models.

An MDA development process basically transforms a PIM into one or more PSMs, which then are transformed into code (code model–CM). The CM is just the actual code generated from PSMs through transformation. Here, the goal is to decouple the way in which interactive applications are currently defined, which is dependent on the technology they use. The purpose of this decoupling is to ensure that the investments made in building systems can be preserved even when the underlying technological platforms change. MDA has been applied to many kinds of business problems and integrated with a wide range of other common computing technologies, including user interfaces (UIs). It makes sense, therefore, to assess the usability of UIs in an interactive application resulting from a MDA process.

The MDA paradigm is a recent manifestation of the old tradition of model-based interface design environments (MB-IDEs) (Puerta 1997), which are aimed at generating the UI of an interactive application as automatically as possible from a conceptual model. Technicians have long observed that the usability of such an automatically generated UI is not known and should be compared with respect to the usability of a manually produced UI. The goal of this chapter, therefore, is to show how to assess the usability of UIs that are automatically generated by a MDA-compliant CASE tool. We also show how the usability evaluation process provides feedback to improve the models that are obtained in an MDA process.

The research question addressed by this study is the following: *Can MDA-compliant methods improve software usability through model transformations?* Specifically, our motivations are the following:

- Information systems (online or offline) probably represent a very significant portion of the total portfolio of today's interactive applications and are used by the widest and mostly diversified population of users. Therefore, it is important to assess the usability of this type of system.
- Because MDA is a modern method for developing information systems, and because it represents a family of development methods that is largely applied in the field of SE, we will focus on this family of methods by selecting a representative member of this family.
- Some works have been conducted in the field of SE (OMG 2006), as well as in HCI (Jespersen and Linvald 2003), to show how MDA can effectively be applied. However, we are not aware of any existing usability assessment of an UI obtained by a MDA-compliant method.
- We select a representative MDA-compliant CASE tool to support the complete development life cycle of the systems we are considering. This CASE tool enables us to automatically generate a completely running interactive application from conceptual models.
- If the usability of an automatically generated UI could be assessed, we would be able for the first time to predict (to some extent) the usability of any future UI produced by this CASE tool. In other words, we will talk about a UI that is (to some extent) *usable by construction*, at least.

We structure the remainder of this chapter as follows: the next section summarizes a literature review used to identify MDA-compliant methods and tools for developing interactive applications. The presentation of our strategy to integrate usability into MDA processes follows. Next, we describe an experimental study to evaluate the usability of a UI generated by a MDA-compliant CASE tool at its code model. The following section presents a comparative analysis of results. We also analyze the identified usability problems and their implications for the PIM and PSM. We then answer the stated research question. Finally, the chapter provides some conclusions and a discussion on future trends in the last section.

1.2 Related Work

According to OMG, there are today around 40 commercial or open source tools (OMG 2006) that follow the MDA paradigm (e.g., Rational Software Architect, ArcStyler, Together Architect, AndroMDA, REP ++ Studio, and OlivaNova). They normally use a UML (OMG 2003) class diagram to capture the system structure and a process model for its functionalities.

These tools support one or more features of MDA, such as UML modeling support, transformation rules, model integration, and code generation. Interaction modeling, however, is not a key issue when requirements and conceptual modeling are represented in a software production process. An exception to this is the OlivaNova tool (CARE 2005). This tool allows the representation of user interaction through a pattern-based approach. A presentation model that is built upon a UML-like class diagram represents user interaction, and other models represent user behavior. All these models, which act as a PIM, automatically transform into code for different target platforms.

Akhter and Tariq (2005) present a comparison of nine selected MDA-based tools (i.e., Optimal J, Arcstyler, Together Architect, MDE Studio, XMF-Mosaic, Ameos, Objecteeing, Constructor, and Codagen Architect). The comparison is done on the basis of identified evaluation criteria organized into three sections: UML and modeling support, a *core models* (PIM, PSM, and CM) approach, and a *transformation* approach. Results show that these tools are not fully compliant with the MDA specification. Most of them provide good support for modeling with UML but need to improve their transformation capabilities.

In the HCI field, many techniques exist in MB-IDEs to automatically generate the UI code: model transformation (Vanderdonckt 2005), static analysis (Eisenstein, Vanderdonckt, and Puerta 2001), skeleton or template-based programming (Janssen, Weisbecker, and Ziegler 1993; Vanderdonckt and Bodart 1993), generative programming, hierarchy construction (Lonczewski and Schreiber 1996), and pattern-based design (Molina, Meliá, and Pastor 2002), to name a few. Although we could identify the differences between these techniques in terms of strengths and weaknesses, it is still impossible to know today which technique is the most efficient.

In Silva's survey (2001), most of the MB-IDEs are reviewed, distinguishing two generations of tools. The goal of the first generation was to provide a runtime environment for UI models—for example, HUMANOÏD (Szekely 1996). The goal of the second generation was to provide more support for interface modeling at a high level of abstraction. Examples of these environments are FUSE (Lonczewski and Schreiber 1996), GENIUS (Janssen, Weisbecker, and Ziegler 1993), TRIDENT (Bodart et al. 1995), MASTERMIND (Szekely et al. 1996), and MECANO (Puerta, 1996). Most environments of this second generation rely on a domain model. This model is often a description of the domain entities and relationships among them, which are represented as a declarative data model (as in MECANO), an entity-relationship data model (as in GENIUS and TRIDENT), or an object-oriented data model (as in FUSE). Some proposals like FUSE, TRIDENT, and TEALLACH (Griffiths et al. 2001) add task models as a primary way for abstracting the user interaction, from which the abstract interface models (or their equivalent dialog models) are later derived.

Other proposals appear with the advent of the MDA initiative. Jespersen and Linvald (2003) proposed a model-based approach to UI engineering that generates domain-specific J2EE applications from a declarative application description. More recently, Mori et al. (2004) proposed a method and the associated tool (TERESA) to support the development of multiplatform user interfaces from conceptual models and transformations between them. Mori's work also briefly reports on the usability evaluation of the user interfaces generated by TERESA done at Motorola.

Vanderdonckt (2005) proposed a UI engineering methodology that adheres to the principles of MDA. It is structured on three axes: models and their specification language, method, and the tools that support the method based on the underlying models. Relevant aspects are stored in UsiXML (www.usixml.org) files to generate different types of UIs (e.g., graphical, vocal, multimodal, and virtual) on different types of computing platforms.

Even though the industry has launched many attempts to establish a comprehensive model-based approach for developing UIs, no consensus has been reached and no method has really emerged from these initiatives—perhaps because the resulting usability remains uncharacterized. In addition, only a few of these environments are MDA-compliant in a genuine way.

Although the need for generating more than merely the UI has been raised (Balzert et al. 1995), the MB-IDEs proposed in the HCI field also do not allow obtaining an entire software system (only its UI).

While experts in the fields of SE and HCI have conducted several works to see how MDA could effectively and efficiently be applied, we are not aware of any existing work that actually performs any explicit usability evaluation of the UI obtained by such a model-based software production method. There are some methods to develop a UI according to a MDA-compliant method, but none of them explicitly integrates usability engineering with UI engineering. This is particularly important because the code portion devoted to the UI could be very important in terms of lines of code: Myers and Rosson (1992) observed that the UI could range from 20 percent for almost non-interactive applications to 85 percent for highly-interactive applications, with a mean of 50 percent.

1.3 Integrating Usability into a MDA Process

The usability of an interactive application obtained from a transformation process can be evaluated at several stages of a MDA process: i) in the CIM, by evaluating the requirements models (i.e., use cases), task models or domain models which represent the user requirements and tasks; ii) in the PIM, by evaluating the models that represent the abstract user interface such as presentation models or dialog models; iii) in the PSM, by evaluating the concrete interface models (if they exist); and iv) in the CM, by evaluating the final user interface.

Figure 1.1 shows the correspondence between the models obtained in a generic MDA process and the usability evaluation activities.

It should be noted that the entire MDA process is driven by the PIM, which is automatically transformed into a PSM, and from there to code; therefore, modeling becomes, in fact, programming on a higher level of abstraction, where the PIM specifies the code that needs to be produced. The aim of a usability evaluation done at the PIM[1] (e.g., presentation models, dialog models) is to produce a *platform-independent usability report*. This report provides feedback to improve the PIM (1A in Figure 1.1) and the PSM (1B in Figure 1.1).

By means of model transformations and the explicit traceability between models, the changes performed in the PIM directly reflect into the PSM, avoiding usability problems in the interactive application (CM) obtained as part of the transformation process. The evaluation at the PIM or PSM should be done in an iterative way until these models have the required level of usability. The UEMs that can be applied in this stage include heuristic evaluation, standards inspection, and action analysis.

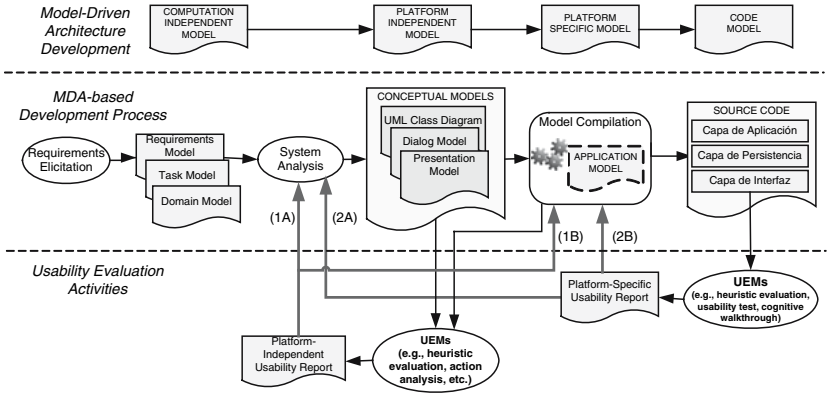


Fig. 1.1 Correspondence between the activities in a MDA process and the usability evaluation activities

[1] Although the evaluation can also be done at the CIM (e.g., requirement and task models), we focus on the transformations between the PIM-PSM-CM models.

On the other hand, other usability factors (i.e., efficiency, satisfaction) can only be evaluated on a specific platform and taking into account the concrete interface components that constitute the resulting user interface (CM). At this stage, we can apply usability inspection methods (e.g., heuristic evaluation) or other empirical methods that include the active participation of users (e.g., user testing, cognitive walkthrough). As a result, we obtain a *platform-specific usability report*.

Note that instead of relating a given usability problem with changes in the interface tier (code model level), as is usual in other approaches, we relate it to the conceptual primitives of the PIM that are affected by it (2A in Figure 1.1). Another benefit, which has no precedents in the MDA literature, is the opportunity to provide feedback to the transformation rules (2B in Figure 1.1). This allows improving of the model compilers and guaranteeing that the generated user interfaces will be usable to some extent.

We believe that construction can estimate the usability of an interactive application resulting from a MDA process. This means that each generated user interface satisfies a certain level of usability resulting from the application of the transformations. Each transformation may improve or diminish the whole usability depending on how they produce their output. This level of usability is no longer resulting from a manual programming process, but from the automated application of transformations. In the domain of theorem-proving, experts say that a program could be proven correct by construction by proving that the different instructions of this program correctly transform an initial precondition into a final valid postcondition. The principle is similar here for usability produced by construction: if we estimate the usability of a user interface produced by any transformation, then we can predict the usability of the final user interface produced by the whole set of transformations that has been used.

1.4 Experimental Study

The goal of the experimental study was to evaluate the usability level of UIs that are automatically generated using a MDA tool. The research question addressed by this study is the following: Can MDA-compliant methods improve software usability through model transformations?

To accomplish this, we evaluated the usability of an interactive application without users and with users using two UEMs at its code model level (implementation) as shown in Figure 1.1.

The methods chosen were action analysis (Olson and Olson 1990) and user testing (Dumas and Redish, 1999). These two UEMs were chosen because of their complementary character. An evaluation without users can catch usability problems (UPs) that an evaluation with only a few users may not reveal. Action analysis allows us to predict the time to complete tasks and the time to learn the interface. Testing with end users provides information about how people use the application and their problems with a specific user interface. We produced a list of usability problems

from the two evaluations. These problems were addressed by changing the PIM, the model transformation strategy (PSM), or both.

1.4.1 Context of the Experiment

Instead of considering any MDA-compliant method, we relied on the process adopted by the OlivaNova – The Programming Machine tool (CARE 2005). We selected this CASE tool for the following reasons:

- It adheres to MDA principles
- It automatically generates an entire information system (including its user interface) based on conceptual models
- It is one of the tools available in the industry that is referenced on the OMG website

Figure 1.2 shows the methodological process followed by OlivaNova. In the following section, we describe in more detail the set of models and the transformation strategy on which the OlivaNova tool is based.

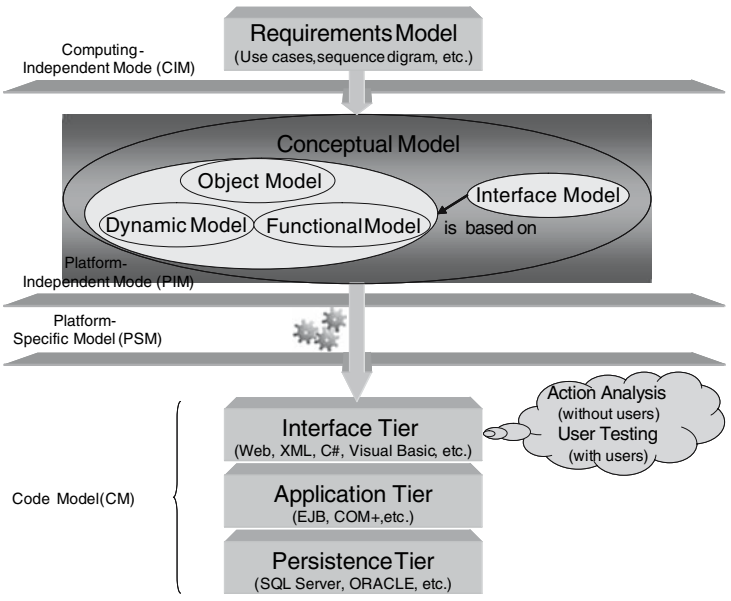


Fig. 1.2 An MDA-based development environment

Models

After specifying the requirements model, a PIM called a conceptual model represents the system data and behavior issues. It consists of the following: an *object model* to represent the structure of the interactive application in terms of classes and their relationships; a *dynamic model* to model the behavior of the system in terms of valid object lives and interaction between objects; and a *functional model* to represent the semantics associated with the state changes of the objects.

To support the conceptual modeling of user interfaces, the conceptual model has been enriched with a *presentation model*. It is composed with a pattern language called Just-UI (Molina, et al. 2002), which decomposes the user interaction in three main levels:

- *Level 1: Hierarchy of Actions Tree (HAT)*—organizes the semantic functions that will be presented to the different users who access the system. The HAT is a tree where intermediate nodes are labels for organizing the functionality and leaf nodes containing the functionality with links to the corresponding *interaction units*.
- *Level 2: Interaction Units (IUs)*—represents abstract interface units (Vanderdonckt and Bodart 1993) with which the users will interact to carry out their tasks. There are four types of interaction units: the Service IU, the Instance IU, the Population IU, and the Master/Detail IU. A *Service IU* models a dialog whose objective is to help the user executing a service. An *Instance IU* models the data presentation of a class instance. A *Population IU* models the presentation of a set of instances of a class. Finally, a *Master/Detail IU* models a complex user interaction that deals with master/slave presentations.
- *Level 3: Elementary Patterns*—are the primitive building blocks of the user interface and allow the restriction of the behavior of the different interaction units. Some examples are *introduction*, which represents data entry and validation in the system; *filter*, which defines selection criteria and obtains the instances that fulfill the constraint expressed; and *action*, which allows the user to execute object services.

Model Transformations

A model compiler that implements a set of mappings between conceptual primitives (PIM) and their software representations in the code model automates the development process. Nowadays, the OlivaNova tool provides model compilers that automatically transform the conceptual model into a full interactive application for the following target computing platforms: Visual Basic, C #, ASP and NET, Cold Fusion and JSP using SQL Server 2000, Oracle, or DB2 as a repository. The resulting application consists of three-layers that include the interface tier, the application tier, and the persistence tier. This unique capability of generating code for multiple targets is the cornerstone of model-based approaches for multi-platform UIs

(Eisenstein, Vanderdonckt, and Puerta 2001) as they represent instances relevant for various technological spaces whose variety is recognized (Lyytinen and Yoo 2002).

1.4.2 Experimental Object

The chosen application is the information management system of Aguas del Bullent S.A., a water supply service company located in Oliva, Spain. The Aguas del Bullent information management software (AdB) supports management for clients, orders, invoices, etc. It is important to note that AdB is a Microsoft-certified application and that it was completely generated from the conceptual model without any manual-based activity. From this application, we selected two representative tasks for conducting the UEMs. These tasks were selected through a questionnaire for the AdB application users to identify the most commonly used and representative tasks. This questionnaire resulted in the following two tasks: *create a subscriber*, and *create a product in stock*. The next section describes both tasks in different levels according to the MDA approach.

1.4.2.1 Task Description in the Computation Independent Model

At this level, we describe the user tasks to be carried out and the domain-oriented concepts that use the ConcurTaskTrees notation (CTT, http://giove.cnuce.cnr.it/ConcurTaskTrees.html). This is done in the Requirements Model step of the process shown in Figure 1.2.

The task model representing the end user's viewpoint of the *create a subscriber* task is shown in Figure 1.3. This is a complex domain-dependent task. As a water supply service company, AdB handles a strongly regulated set of concepts. The local administration controls the physical unit called a *water meter*. It is installed in homes and buildings, and is referred to as a *subscriber*. When the same *client* has more than one water meter installed (for example, if the same person owns

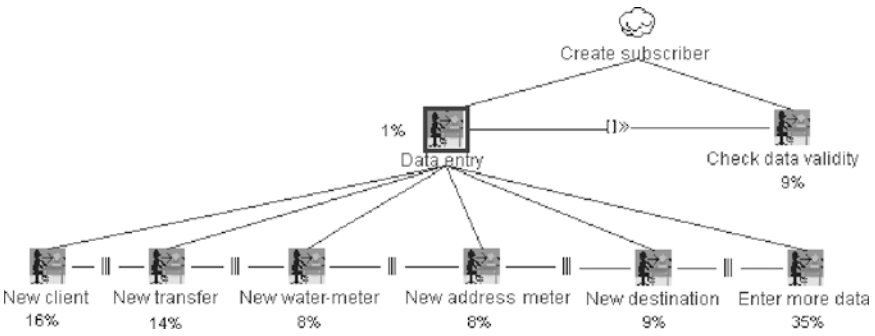


Fig. 1.3 A task model for the *create a subscriber* task

two apartments), each meter may be a different size, may be used for a different purpose (private or industrial use, etc.) or may be in a different area. In this case, we will use the concept of *subscriber* for each unit, and the service is invoiced separately.

When new areas need to be supplied with water, AdB is notified in advance by municipal administrations. There are no streets (nor streets names), but there are maps that are divided into lots. These lots are known as *destinations* and are introduced in the application when the infrastructure is done. It then creates and names streets. Some contractors ask AdB to install all the water meters before selling the properties. This is when most *subscribers* and *water meter addresses* are created. The property is later sold, and a new *client* is created and the corresponding subscriber updated to be linked to the new client.

This task is decomposed into the following subtasks: create a *new client*, create a *new transfer,* create a *new water meter*, create a *new address meter*, create a *new destination*, *enter more data*, and *check the validity* of the entered data. Given the complexity of the task, we assigned a percentage of task completeness for each subtask to indicate how much of task has been properly achieved. This could be improved in the future by assigning a weight that is a function of the amount of the subtask, but also of the importance of the subtasks.

Similarly, the task model representing the end user's viewpoint of the *create a product in stock* task is shown in Figure 1.4. This is a moderately complex domain-independent task that allows the introduction of a certain number of products in a warehouse. It is carried out in the following sub-tasks: create a *new product*, *select warehouse*, and *introduce number of products in stock*. As in Task 1, we assigned a percentage of completeness for each subtask.

1.4.2.2 Task Description in the Platform-Independent Model

At this level, the tasks to be performed are described as abstract user interfaces using the Presentation model. This corresponds to the last step of building the system conceptual model, as shown in Figure 1.2. An abstract user interface is a canonical expression of the rendering of the domain concepts and tasks in a way that is independent from any modality of interaction (Vanderdonckt 2005). It is considered an abstraction of a concrete user interface with respect to interaction modality.

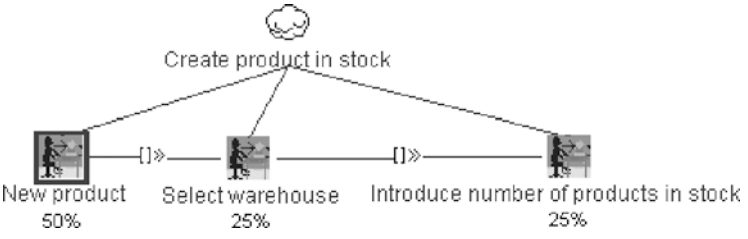


Fig. 1.4 A task model for the *create a product in stock* task

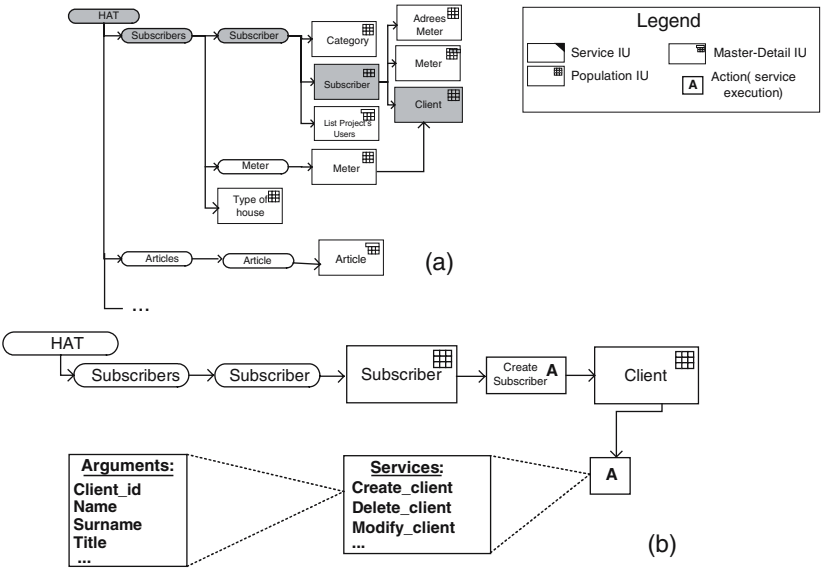


Fig. 1.5 Excerpt of the Presentation model for the AdB application

Figure 1.5 shows a piece of the Presentation model for the AdB application using the Just-UI patterns (Molina, et al. 2002)—(a) shows the HAT of the application, including the intermediate nodes (the menu/submenu options) and the *interaction units* that the users will interact with; and (b) shows all the *actions* (class services) accessible from the Client Population IU, accessed by following the path highlighted in (a) and the arguments of the *create_client* service.

1.4.2.3 Task Description in the Code Model

The model compilation shown in the process shown in Figure 1.2 defines an application model that is equivalent to the Platform-Specific model in MDA. However, as this model is implicit in ONME, there is no need to describe the tasks at this level. Consequently, the last level is the Code model. It corresponds to the final user interface that allows the tasks to be described by explaining the navigation between windows and the expected interaction between the user and the system.

Figure 1.6 shows part of the final UI for the AdB application running in the .NET platform (CARE 2006b), generated from the Presentation model shown in Figure 1.6. In the figure, (a) shows a form obtained for a *Service IU* (NewSubscriber). Once the data is entered, the user can launch the service or cancel it. Also in the figure, (b) shows a form obtained for the *Client Population IU* that is accessed by selecting the first lens icon in the NewSubscriber Service IU. This form is delimited by three zones—the central zone shows the client instances, the upper zone is used to perform searches using filters, and the right side zone is a toolbar to access the

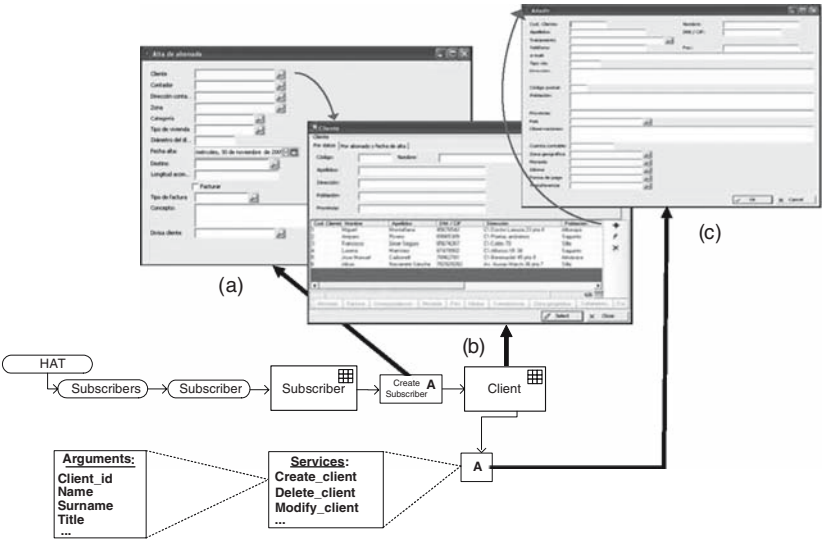


Fig. 1.6 Final user interfaces of the AdB application

create_client, *modify_client* and *delete_client* actions. In the figure, (c) shows a form obtained for the *NewClient Service IU* that is accessed by selecting the *create_client* action in the Client Population IU.

1.4.3 Usability Evaluation without Users

In this section, we describe how to conduct the usability evaluation of the AdB application without users using the two selected tasks. This kind of evaluation is useful for identifying usability problems and improving the usability of an interface by checking it against a standard. In this study, the results of applying the *action analysis* method (Olson and Olson 1990), based on the recommendations of Holzinger (2005), are used as standard. It is divided into *formal* and *back-of-the-envelope* action analysis. We used the formal approach that is often called *keystroke-level* analysis. It allows more accurate predictions of the time it takes a skilled user to complete the tasks. In addition, this prediction is done within a 20 percent margin of error, as reported by Lewis and Rieman (1993).

The keystroke-level analysis requires close inspection of the action sequences that a user performs to complete a task. It has two phases: the first phase determines the physical and mental steps a user performs to complete one or more tasks with the interface to predict the time that the user needs to do the task; and the second phase analyzes these steps, looking for problems. Some UPs that the analysis might reveal are that it takes too many steps to perform a simple task, or that it takes too long to perform the task, or that there is too much to learn about the interface, etc.

1.4.3.1 Keystroke-Level Analysis

We decomposed the two selected tasks into a sequence of actions that the user must perform to complete each task in order to predict the tasks' performance times. To do this, a time is associated with each one of these actions or sequence of operators (physical or mental), and they are then totaled. The time is calculated by using the average time that it takes a skilled user to complete the action, as suggested by Lewis and Rieman (1993). For example, one keystroke on a standard keyboard is considered a physical movement that takes 0.28 seconds, whereas learning a single *step* in a procedure is considered a mental movement that takes 0.25 seconds. Below, we describe some of the low-level physical and mental actions required to perform the *create a subscriber* task.

Subtask 1: Data Entry

1.1.	Recognize the menu entry "Create subscriber"	(0.34 s)
	This is a combination of moving the eyes, reading the name of the label, and performing the mental step of matching.	
1.2.	Move the mouse to the option "Create subscriber" and click it	(1.5 s)
	This is a standard mouse movement time. We know from the selection rule that the hand is already on the mouse.	
1.3.	See the action result. The screen is similar to Figure 1.6 (a)	(0.23 s)

Subtask 2: New Client

2.1	Recognize the client label	(0.34 s)
2.2	Move the mouse to the client lens icon and click it	(1.5 s)
2.3	See the action result. The screen is similar to Figure 1.6 (b)	(0.23 s)
2.4	Recognize the add client label	(0.34 s)
2.5	Click on "add client"	(1.5 s)
2.6	See the action result. The screen is similar to Figure 1.6 (c)	(0.23 s)
2.7	Recognize the client code label	(0.23 s)
2.8	Remember the client code	(1.2 s)
	This is just recalling an item from long-term memory	
2.9	Write the client code	0.84 s (0.28 * 3)
2.10	Recognize the client name label	(0.34 s)
2.11	Remember the client name	(1.2 s)
2.12	Write the client name	2.52 s (0.28 * 9)
...		
2.19	Recognize the *title* label	(0.34 s)
2.20	Move the mouse to the title lens icon and click it	(1.5 s)
2.21	See the action result	(0.23 s)
2.22	Remember the desired title	(1.2 s)
2.23	Find the desired title	12.58 s (0.34 * 37)
2.24	Move the mouse to click the desired title	(1.5 s)
2.25	Return the focus to add client screen	(0.23 s)
...		

Table 1.1 shows the time prediction for the eight subtasks (ST) of the *create a subscriber* task. It means that on average, a skilled user will take about 467.34 seconds (7.79 min.) to complete the task.

Table 1.1 Time prediction for Task 1

ST 1	ST 2	ST 3	ST 4	ST 5	ST 6	ST 7	ST 8	Total
40.65 s	157.9 s	77.05 s	28.48 s	30,25 s	40,51 s	67.19 s	25.34 s	467.34 s

Similarly, the keystroke-level predictions for the performance times of the *create a product in stock* task are as follows:

Subtask 1: New Product

1.1.	Recognize the "Articles" menu entry	(0.34 s)
1.2.	Move the mouse to the "Articles" menu	(1.5 s)
1.3.	See the action result	(0.23 s)
1.4.	Recognize the "Article" menu entry inside "Articles"	(0.34 s)
1.5.	Move the Mouse to the submenu entry called "Article"	(1.5 s)
1.6.	See the action result	(0.23 s)
1.7.	Recognize the add article with supplier label	1.02 s (0.34 * 3)
1.8.	Click on "add article" with supplier	(1.5 s)
1.9.	See the action result	(0.23 s)
1.10.	Recognize the *family* label	(0.34 s)
1.11.	Move the mouse to the family lens icon and click it	(1.5 s)
1.12.	See the action result	(0.23 s)
1.13.	Remember the desired family	(1.2 s)
1.14.	Find the desired family	4.08 s (0.34 * 12)
1.15.	Move the mouse to click the desired family	(1.5 s)
1.16.	Return the focus to add article with supplier	(0.23 s)
...		

Table 1.2 shows the time prediction for *create a new product in stock* (Task 2). The three subtasks (ST) are the following: *new product*, *select warehouse*, and *introduce number of products in stock*. The total time is 152.76 seconds (7.55 min.).

1.4.3.2 Analysis and Interpretation of Results

Even though the action analysis method is mainly used to predict task time, it can also highlight other UPs. One issue revealed by the analysis was that Task 1 might take too long because it is a complex task composed of several subtasks and steps. Specifically, Subtask 2, *New client*, is the subtask that takes the longest.

One of the causes of this problem is the use of the lens icon. For instance, the first field of the window shown in Figure 1.6(a) is dealt with by executing Steps 2.1 to 2.12. Similarly, the other eight lens icons must also be dealt with. This means executing 96 steps to accomplish Task 1. Note that the same interaction pattern

Table 1.2 Time prediction for Task 2

ST 1	ST 2	ST 3	Total
78.75 s	32.62 s	41.39 s	152.76 s

occurs during the execution of Task 2 (Steps 1.10 to 1.16). A possible solution for making Task 1 easier is to split it into smaller tasks, in contrast with Task 2, which is already well divided into subtasks (the subtasks have similar times).

Another question that the analysis highlights is the user workload. The higher the workload is, the higher the probability of making errors will be. During the interaction, the user needs to select the lens icon to enter the information in another window. For instance, in the third field of the window shown in Figure 1.6(c), a lens icon is provided to help the user enter the client title (e.g., Mr. or Mrs.). To do this, Steps 2.19 to 2.25 must be executed. This requires navigation to a new window, in which the user must select a title that already exists or create a new title.

The analysis also calls into question the large number of steps needed to do this fairly simple subtask. It is widely accepted that the shorter the required actions are, the faster the interactions will be. It can be noted that this potential problem is recurrent, because the other six lens icons (i.e., country, geography zone, and currency) must be dealt with in the same way. This means executing 36 additional steps. Of course, because any analysis at this level produces a *large* number of steps, it must first be determined whether or not there is a problem by comparing the task to other tasks of similar complexity.

Therefore, the workload must be reduced to increase the dialog efficiency. To do this, some of the lens icons should be changed into a list box or a combo box. This requires some further thought, because the data selection technique used should be consistent as well as quick. Another feature that could be included is the option to write and recognize the data entry. Also during data entry, the application should display default values in their appropriate data fields.

A related issue is the long-term memory requirements of the task. The analysis makes it clear that the user must recall several pieces of data during the task execution. This can be observed in Step 2.8 and Step 2.11 of Task 1. To avoid this, the application should not require data entry by the user when the data can be derived. We also noticed that, in some contexts, no feedback for user selection (i.e., color change) was provided. The problems found in the analysis facilitated the drafting of design usability requirements.

1.4.4 Usability Evaluation with Users

In December 2005, a user test was conducted in CARE Technologies. The goal was to evaluate how well or poorly the AdB application and its user interface performed, and how satisfied the users were in completing the two selected tasks. Specifically, we measured user *effectiveness*, *efficiency*, and *satisfaction* (ISO 1998). Effectiveness relates the goals of using the product to the accuracy and completeness with which these goals are achieved. It was measured using the following measures proposed in the Common Industry Format (CIF) for usability tests: the completion rate and the frequency of assists. The *completion rate* is the percentage of participants who completed each task correctly. We distinguished

between two types of rates: i) *unassisted completion rate,* which is the rate achieved without intervention from the testers, and the ii) *assisted completion rate,* which is the rate achieved with tester intervention. The *frequency of assists* is the number of times that the experimenter assisted the user. Efficiency relates the level of effectiveness achieved to the quantity of resources expended. It was measured in terms of *goal achievement* divided by *task time*. Finally, satisfaction was measured as Questionnaire for User Interaction Satisfaction (QUIS) scores (see http://lap.umd.edu/QUIS).

1.4.4.1 Subjects

The test involved eight[2] expert and inexperienced end-users. They were internal employees of Aguas del Bullent and CARE Technologies. Five of them were male and three were female. Their expertise in the application domain and technology was considered as an independent variable.

Table 1.3 shows the distribution of users into four groups: novice in domain experience (DE) and technology experience (TE) (Group 1), novice in DE and expert in TE (Group 2), expert in DE and novice in TE (Group 3), and expert in DE and TE (Group 4). The decomposition of the user population into two dimensions (i.e., DE and TE) results from characterizing a user population along the two dimensions of syntactical knowledge and semantic knowledge, as recommended in (Jarke and Vassiliou 1985). The syntactic knowledge refers to the user's ability to interact technically with an interactive system through its devices, and is independent of any domain. The semantic knowledge refers to the user's knowledge in a particular domain of human activity. These two dimensions are independent.

1.4.4.2 Procedure and Instrumentation

We set up a usability laboratory in CARE Technologies to conduct the test. The interaction between the user and the AdB application was video-recorded and logged. An instructor and one independent observer participated in the study. The

Table 1.3 Distribution of subjects

Individual Testers (N = 8)		Domain Experience (DE)	
		Novices	Experts
Technology Experience (TE)	Novices	User7	User5
		User8	User6
	Experts	User3	User1
		User4	User2

[2] Prior research has shown that using four subjects in a usability test maximizes the ratio of the number of UPs found to the effort involved in running the test (Nielsen and Landauer 1993).

instructor interacted with the user while the observer took a record of observations and assists during the user test. Users were informed that the usability of the AdB application was going to be tested to check whether the system met their needs. No pretask training was scheduled. The design of the test followed a logical sequence of events for each user and across tests. The testing procedure was as follows:

- Users were given information about the goals and objectives of the test. They were told that it was not a test of their abilities. They were also informed that their interaction would be recorded.
- Users were asked to fill in a *demographic questionnaire* prior to the testing. It was used to confirm their job description, time in their job (years), education level, and age. They also scored their attitude towards technology in general, the use of computers, the use of the AdB application, and their experience with the OlivaNova technology (on a scale of 1 to 5).
- Users were then given a series of clear *instructions* that were specific for the test. They were advised to try to accomplish the tasks without any assistance, and that they should only ask for help if they felt unable to complete the task on their own.
- Users were asked to complete the two tasks. To avoid a possible ceiling effect, there was no time limit to complete the tasks.
- Users were then asked to complete a *QUIS* after completing the last task. This questionnaire is arranged in a hierarchical format and contains: 1) a demographic questionnaire, 2) six scales that measure overall reaction ratings of the system, and 3) four measures of specific interface factors—screen factors, terminology and system feedback, learning factors, and system capabilities. Each of the four specific interface factors has a main component question followed by related subcomponent questions. Each item is rated on a scale from one to seven with positive statements on the right side and negative statements on the left side. In addition, *not applicable* is listed as a choice. Additional space that allows the users to make comments is also included in the questionnaire. The comment space is headed by a statement that prompts the user to comment on each of the specific interface factors. Users are tracked by a number.
- Users were given a small reward for their participation.

After the experimental session, a we conducted a debriefing to collect information about how to improve the application.

1.4.4.3 Operation and Data Collection

We also followed a strict protocol for the usability testing to ensure consistency of the data collected. The test user was expected to accomplish the two tasks without being assisted. In any case, if the user got stuck on one task and felt unable to continue, he/she could ask for help—the instructions given to the users clearly stated that they should only ask for assistance as a last resort. When a user asked for assistance, the task was marked as *assisted*. Therefore, that instance of the task was computed in the assisted completion rate variable and not in the unassisted completion rate variable.

We recorded the data times using the Allnetic Time Tracking software (www.allnetic.com). Times were collected for each user, task, and subtask. We also included remarks about the type of help or assistance given to the user. The instructor was aware of any point where the user got stuck, because he/she was following the users' actions on a screen and checking the actions made by the user in a follow-up list based on the user manual. When the user asked for help, the instructor explained the next step that the user needed to take. The user performed the next step in the presence of the instructor. The assistance given was documented by annotating the step where the user got confused or lost.

1.4.4.4 Analysis and Interpretation of Results

In this section, we discuss the results and findings of the usability study. Table 1.4 shows the characteristics and capabilities of the users.

The demographic data for subjects revealed that 62 percent of the subjects were male; 62 percent of them ranged in age from 20–39 years; and 25 percent of the subjects had experience with the application they were evaluating.

Table 1.5 shows the participants' effectiveness and efficiency in executing Task 1. It illustrates that, on average, the users took 14 minutes to complete the task. The *new client*, *new water meter*, and *new address meter* subtasks took longer to perform. The data also shows that three out of eight users successfully completed the task without assistance and that three out of eight users completed the task with assistance.

In terms of frequency of assists, Subtasks 4 and 6 are the ones for which the users required more assistance (three assists), followed by Subtask 5 with two assists, and Subtask 7 (*enter more data*) and 8 (*check the validity of the data entered*) with one assist each. User 3 was the only user that required assistance on Subtasks 3 to 8. However, even with these assists, this user was not able to complete the task. This indicates that the most difficult or problematic subtasks are *new water meter* (Subtask 4) and *new address meter* (Subtask 5).

Looking at the videotapes, we could verify that the users got confused and lost in these subtasks. In Subtask 4, the reason was that, when the users selected the action to create the new water meter, a meaningless error message appeared: *"There is an open DataReader that corresponds to this connection. You must close it*." In Subtask 5, the user must select a *subscriber* that has not yet been created because the creation of an address meter is part of the creation of the subscriber. This is because the application does not maintain the reference of the new subscriber that is being created. In terms of efficiency, Users 4, 5, and 8 performed better than the others. These users completed the tasks without any assistance. This suggests that the efficiency is independent of the domain and technology experiences as they pertain to different categories. Indeed, Users 4 and 8 were considered novices in domain experience.

Table 1.6 shows the effectiveness and efficiency of the users executing Task 2. As can be observed, the users took an average of eight minutes to complete the task. The *new product* subtask took longer than the others. In terms of effectiveness,

Table 1.4 Characteristics of users

	Gender	Age	Education level	Job	Time in Job (Years)	Computer Usage Freq. (Windows / Others)	Attitude Towards Technology	AdB App. Usage Frequency	Expertise Level in Water Supply	OlivaNova Tech. Experience (End User / Analyst)
1	M	35	B	A	13	5 / 0	2	4	2	5 / 0
2	M	33	C	A	10	5 / 0	2	5	2	5 / 0
3	M	62	D	M	11	5 / 0	1	0	4	0 / 0
4	M	45	D	M	6	4 / 0	2	0	3	1 / 1
5	F	29	D	AN	6	5 / 0	2	0	2	4 / 5
6	F	32	D	S	6	5 / 0	1	0	1	5 / 3
7	F	26	C	R	0,25	5 / 1	2	0	1	0 / 3
8	M	43	C	P	–	5 / 0	2	0	1	0 / 0

Education level: A = school-leaving certificate, B = secondary graduate, C = qualified / technical engineer, D = graduate / engineer
Job: A = administration staff, M = management, AN = analyst, S = support, R = research, P = programming

Table 1.5 Task 1: Effectiveness and efficiency measures

User #	Unassisted Task Completion Rate[a] (%)	Assisted Task Completion Rate (%)	Task Time[b] (min)	Completion Rate / Task Time	Assists
1	39%	8%	21.0	1.86%	1
2	31%	17%	10.0	3.10%	2
3	31%	69%	41.0	0.76%	5
4	100%	0%	17.0	5.88%	0
5	100%	0%	09.0	11.11%	0
6	47%	9%	16.0	2.94%	1
7	31%	8%	9.0	3.44%	1
8	100%	0%	20.0	5.00%	0
Mean	64%	6%	14.57	4.76%	1.25
Std. Error	12.8%	2.4%	1.96	1.17	0.59
Std Dev.	34.1%	6.4%	5.1	3.10	1.66
Min	31%	0%	9.0	1.86%	0
Max	100%	17%	21.0	11.11%	5

[a] Rate of subtasks achieved without testers' intervention
[b] Period of time that the user executes the task without assistance (until the first time s/he asks for assistance). Summary data are given for the seven participants who completed the task.

four out of eight users successfully completed the task without assistance, two users completed the task with assistance, and the other two could not complete the task. In terms of assistance, two assists were given in each subtask. Again, User 3 was the user that required the most assistance and was not able to complete the task.

With respect to efficiency, Users 4, 5 and 6 performed better than the others. This shows that users who are domain experts but technology novices do not experience significant trouble in finding their way in the system generated by OlivaNova.

Descriptive statistics for overall user reaction ratings of the AdB application, the four measures of specific interface factors, and the general user satisfaction are

Table 1.6 Task 2: Effectiveness and efficiency measures

User #	Unassisted Task Completion Rate (%)	Assisted Task Completion Rate (%)	Task Time (min)	Completion Rate / Task Time	Assists
1	50%	25%	7.0	7.14%	1
2	75%	25%	9.0	8.33%	1
3	0%	100%	12.0	0%	3
4	100%	0%	7.0	14.29%	0
5	100%	0%	4.0	25%	0
6	100%	0%	6.0	16.67%	0
7	100%	0%	15.0	6.67	0
8	0%	50%	0	0%	1
Mean	87.5%	8.33%	8.0	13.01%	0.33
Std. Error	8.5	5.2%	1.5	2.91%	0.21
Std. Dev.	20.9	12.9%	3.7	7.14%	0.51
Min	50%	0%	4.0	6.67%	0
Max	100%	25%	15.0	25%	1

Summary data are given for the six participants who completed the task.

Table 1.7 Descriptive statistics for perception-based variables

	Minimum	Maximum	Mean	Median	Std. Deviation
Overall User Reactions	2.00	5.33	3.97	4.33	1.145
Screen Factors	2.25	6.75	4.00	3.75	1.42
Terminology and System Feedback	1.83	5.83	4.27	4.1	1.31
Learning Factors	1.83	6.40	4.57	4.80	1.39
System Capabilities	4.60	6.0	5.35	5.40	.57
Satisfaction	2.35	5.59	4.36	4.75	1.07

presented in Table 1.7. A participant's score for a perception-based variable was calculated as the mean of the scores assigned to the different items of that variable. Higher scores indicated greater user satisfaction with the use of the application. With the exception of the overall user reactions' measurement, the mean scores for all the other measures were higher than the middle score (i.e., a score of 4) on the Likert scale. The mean question scores varied from 3.97 to 5.35, with standard deviations ranging between 0.57 and 1.42. Novice users in the domain and technology (first-time users) judged the application more satisfactory than other types of users.

For these measures to be considered valid, however, an analysis of the construct validity and reliability of the QUIS are needed. The construct validity was carried out through an *inter-item correlation analysis* of the questionnaire. To do this, we used the concepts of convergent and discriminant validity, as proposed by Campbell and Fiske (1988).

An item is considered valid if its convergent validity (CV) is higher than its discriminant validity (DV). We assumed that all items associated with a particular construct have equal weights. As a result, Items Q4, Q15, and Q16, as well as the system capability items, were invalid (the DV for these items was higher than their CV). For this reason, these items were excluded from the analysis. With the exclusion of these items, the results of the validity analysis improved. The mean correlations between each main item and a general satisfaction scale ranged between 0.64 and 0.94. This suggests that there is a strong correlation between the items of the QUIS and general user satisfaction.

We also conducted a reliability analysis on the items used to measure the perception-based variables. Reliability of an instrument describes the consistency (or repeatability) that the instrument gives in measuring the same phenomenon over time or by different people. For this reliability analysis, items Q4, Q15, and Q16, as well as the system capability construct, were excluded. The results obtained for each construct using Cronbach alpha are: overall user reactions $= 0.842$; screen factors $= 0.814$; terminology and system feedback $= 0.922$; and learning factors $= 0.805$. These values are all 0.7 or above as required for constructs to be deemed reliable, as suggested by Nunally (1978). In addition, the general Cronbach alpha obtained for the instrument was 0.8. As a result, we conclude that the items on the survey (except the excluded items) are reliable and valid measures of the underlying perception-based constructs.

1.5 Correlation between Results

We performed a Spearman's rank order correlation analysis to assess the relationships that exist among usability measures. The results in Table 1.8 show that the unassisted task completion rate (UTCR) and task time are significantly negatively correlated (r = 0.910). This indicates (as expected) that when task time decreases, UTCR increases. Similarly, UTCRs are significantly negatively correlated with assists (r = 0.877). There is also a significant positive correlation between task time and assists (r = 0.898). As the assists increase, the task time also increases. Exceptions are the perception-based variables where most of the correlations are not significant at the 95 percent confidence level. The exception are screen factors that have a positive correlation with task time (r = 0.548). These results are in accordance with other studies that reveal that the aforementioned aspects of usability are not well understood for complex tasks (Frøkjær et al. 2000). Frøkjær's study also suggests that a weak correlation between usability measures is to be expected, and that efficiency, effectiveness, and satisfaction should be considered independent aspects of usability.

On the other hand, one of the reasons for conducting user testing was to get a baseline for the evaluation of the AdB application by comparing the users' performances with the prediction obtained with the action analysis. Because the action analysis provides quantitative estimates of task execution time, it can properly be considered as a benchmark. Table 1.9 shows a comparison between the predicted and measured time spent for the users to perform the two selected tasks. In this analysis, we included only the users that successfully completed the task without assistance. As noted, the efficiency of the users performing the tasks did not match the task time predictions. On average, the time it took users to complete the two tasks were 96.4 percent and 389 percent, respectively. This can be explained by the fact that the keystroke-level analysis model does not take into account individual differences among users. In addition, the accuracy of a model of this kind is highly dependent on the estimates of the operator time values for a specific user. As the prediction is made for expert users who do not make any errors, the novices who need to learn the application are not considered.

Table 1.8 Spearman's correlation between user performance and perceptions

		UTCR	Task Time	Assists	Learning Factors	Terminology and System Feedback	Screen Factors
UTCR	p < 0.0001	1.000	−0.910[a]	−0.877[a]	,205	−0.084	−0.359
	Sig. 1-tailed		0.001	0.002	0.313	0.422	0.191
Task	p < 0.0001	−.910[a]	1.000	0.687[b]	−0.335	0.143	0.548
Time	Sig. 1-tailed	0.001	.	0.030	0.208	0.368	0.080
Assists	p < 0.0001	−.877[a]	0.898	1.000	−, 111	−0.184	0.172
	Sig. 1-tailed	0.002		.	0.397	0.331	0.342

[a] Correlation is significant at 0.01
[b] Correlation is significant at 0.05

Table 1.9 Comparison between measured and predicted performance times

User	$T1_{pred.}$ [sec]	$T1_{mes.}$ [sec]	Diff. [%]	$T2_{pred.}$ [sec]	$T2_{mes.}$ [sec]	Diff. [%]
4	467.34	1020	118%	152.76	420	274%
5	467.34	540	15%	152.76	240	157%
6	467.34	–	–	152.76	360	235%
7	467.34	–	–	152.76	1020	667%
8	467.34	1200	156%	152.76	–	–

1.6 Usability Problems and their Implications for PIM and PSM

This section discusses how the identified usability problems relate to the models obtained in a MDA process. We also show how the two different evaluation methods produce problems originated in different models.

Section 1.4.3 discusses some of the most common usability problems identified in the evaluation without users. They are mainly related to long tasks that require long-term memory requirements and cause user workload problems. In addition, it was detected that the application does not provide feedback for user selection in some specific contexts.

In contrast, the evaluation with users revealed a set of usability problems that were mainly identified through the videotapes and observation records. Some of these problems were: lack of descriptive labels, difficult navigation (cyclic and unnecessary steps), inflexible/invalid search capabilities, inconsistent menu items and window labels, no feedback for user selection, lack of descriptive labels in some fields, and meaningless error messages. We also verified that some of the identified problems during the user test were further supported by the comments made by the subjects in the post-task questionnaire. Some answers for the question "List the three things that you least liked in the application" included:

1. The organization of the window in frame: the functionality of one of them appears hidden. [#2]
2. The navigation through lens icons: I lost the source window.
3. The data that was just entered does not appear: it is needed to search it manually when the application could provide it. [#3]
4. Error messages lack clarity and understanding. [#2]

On the other hand, some answers for the question "List the three things that you most liked in the application" included:

1. The application structure and menu are very intuitive and clear. I have no prior knowledge in the use of the application.
2. The possibility to find the information from different places of the application. [# 2]
3. The interface is simple (visually)
4. The application is very fast in searches and in all the actions that I executed. [#4]

Table 1.10 Usability problems by usability evaluation method

Usability Problem	Action Analysis	User Testing		Source
		Videotape	QUIS	
Inconsistent menu item and window labels		√		PIM
User workload (unnecessary searching)	√		√	PIM/PSM
Lack of descriptive labels		√	√	PIM
Difficult navigation	√	√	√	PIM
Inflexible/invalid search capabilities		√	√	PSM
No feedback for user selection	√	√	√	PIM/PSM
Invalid filters		√	√	PIM
Meaningless error messages		√	√	PIM/PSM
Long-term memory requirements	√		√	PIM

Table 1.10 shows a classification of UPs by evaluation method. As can be observed, the problems identified with the action analysis are related to procedural issues of usability (i.e., too many steps to complete the task, difficult navigation) as this method only represents the procedural aspects of a task. In contrast, the problems identified in the user test are more concerned with perceptual (i.e., meaningless error messages) and conceptual issues than verifying whether the user has an appropriate *mental model*. Another important issue is determining the source of the identified UPs (model level, pattern level, code transformation level, or platform level). This information will be used to create a plan for design recommendations, modifications, and improvements in the AdB conceptual model or transformation engines.

The issue of meaningless messages is related to the quality of error messages. An error message that appeared when one of the users finished Task 1 was the following: "*Maximum cardinality for Subscriber is not satisfied. At most 1 instances of subscriber are allowed*." Although the message includes prompting about the way to correct the error, the indications are not meaningful to end-users. The source of this problem is in the model level (Presentation model). It was the property *validation message* of the *Introduction* pattern of a Service UI (see Section 1.4.1). This pattern captures the main features of data types that the user introduces into the system. It includes the definition of edition masks, value by default, rank of values, help messages, and validation messages.

The source for the lack of descriptive labels and the inconsistent page problems is also in the model/pattern level. The former is related to the lack of *supplementary information* patterns, which is used to define additional information that will be shown when a user selects an object (i.e., field). The latter problem is related to consistency. This happened because the analyst gave a different name to a leaf node of the HAT (i.e., menu item of the application) and the title of the interaction unit (window) associated with it (see Section 1.4.1).

We also observed other UPs in the code transformation level. They were related to the implementations of the OlivaNova transformation engines. A frequent problem in Task 2 was that the user could not find the action to create an article (Subtask 1).

This occurred because the *action* icon was hidden, even if the window was maximized. Other problems in this level were the following: the help reference was not linked to the generated application and the F1 functionality on controls was not implemented, and the data of a display set (e.g., grid) was not refreshed when a service (i.e., create new client) was executed. A possible solution was to refresh data after a service execution.

1.7 Answering our Research Question

Thanks to the model transformations that implement the relationships between abstract UI, concrete UI, and the final UI, the usability of an application can be produced by construction. Depending on the transformation rules that are applied, the abstract presentation patterns (abstract interaction objects, AIOs) are instantiated in the corresponding software components in the target platform architecture (concrete interaction objects, CIO) (Vanderdonckt and Bodart 1993). This strategy is called reification (Vanderdonckt 2005). In this way, the changes made to fix usability problems will be always preserved during transformation.

The duality between COIs and AOIs has allowed us to abstract usability attributes and other relevant properties from the user interface elements. We have verified that some usability attributes are dealt with during the reification process. For instance, the following user guidance attributes related to visual feedback are directly ensured—visual feedback in menus or dialog boxes where choices can be selected or where the cursor is pointing, visual feedback where options are already selected for multiple options, and visual feedback when objects are selected. In addition, other attributes related to the legibility of the UI are also ensured—prompts and error messages always appear in the same place(s), and error messages displayed in pop-up windows indicate the field(s) where the error has occurred, etc.

The results of this study not only provide feedback to improve the AdB conceptual model, but also provide feedback to improve the .NET model compiler (CARE 2006b). For instance, one usability problem revealed by the action analysis was that Task 1 might take too long. A possible cause is the use of *lens icons*, which imply additional navigation. This occurs because the object-valued service arguments of a Service UI (Table 1.11) are implemented using generic control selectors that are, in turn, translated into the fields using an associated *lens icon* (one-to-one mapping). To address this problem, the code generator should be changed to allow different implementations for the same generic control selector (concrete object interface).

All these results provide evidence that the usability of an interactive application can be produced by construction. It is our belief that model-driven development provides the basis for tight integration of usability evaluation in the MDA development process, allowing usability issues to be addressed as an integrated part

Table 1.11 Correspondences between the presentation model, the transformation engine component, and the final user interface

Presentation Model (PIM)	.NET Code Generator (PSM)	Final Interface (CM)
Hierarchical action tree (HAT)	Application menu	File Changes Others Window
Service IU	MDI child form with entry controls	The whole window
Service throw and Cancellation	"accept" and 'cancel' buttons and code to validate, invoice and close the service	OK Cancel
Simple type service argument	Entry control	Identifier:
Object-valued service argument	Generic control OIDSelector	Employee:
Introduction pattern for a date field	Generic control OIDSelector	Code for validating date format

of the system design and not just as an ad-hoc solution after most of the development has been completed. It also provides the basis for platform-independent usability assessment, where the generic part of a system is assessed at the platform level, and not from scratch, for each platform-specific technology. This is a promising result and supported our research question. We are aware, however, that these are only the first results in this direction and that more experimentation is needed.

1.8 Conclusion and Perspectives

We have presented an empirical study to evaluate the usability of user interfaces that were automatically generated by a MDA-compliant tool. The generated application was evaluated without users and with users using two UEMs at its Code model level. The results were collected using both quantitative and qualitative data, ranging from manually capturing user behavior to measuring user satisfaction using a post-test questionnaire.

The evaluation without users revealed UPs that are more procedurally oriented, while the evaluation with users revealed problems that are more perceptual and cognitively oriented. It is encouraging, however, that the kinds of UPs identified with different methods are quite similar, which suggests convergent validity. Although the evaluations were conducted in the Code model (final user interface), they provided valuable feedback on the improvement of the Platform-Independent model and the Platform-Specific model, as well as the Code model, as a consequence of the model transformation process. Our study also provides some empirical evidence of the intrinsic quality of the user interfaces generated by a tool, and to the notion of *usability proven by construction*. Usability by construction is analogous to the concept of *correctness by construction* (Hall and Chapman 2002) introduced to

guarantee the quality of a safety-critical system. In this development method, the authors argue that to obtain software with almost no defect (0.04% per KLOC), each step in the development method should be assessed with respect to correctness. Figure 1.7 depicts when defects are introduced and then removed in a development method that does not rely on correctness. For instance, 38 defects were introduced during the Specification step and removed later on during the Code step. If we can maintain proof of the correctness of a software from its inception until its delivery, it would mean that we can prove that it is correct by construction. Similarly, if we can maintain proof of the usability of a user interface from its specifications until the final code, it would mean that we can prove that it is usable by construction.

Of course, in the experiment conducted here, we can only show that each applied pattern reaches a certain level of usability. Therefore, we may predict the global usability of an entire interface by estimating the relative usability levels of the patterns and transformations involved in the development method. We cannot prove that a UI is entirely usable, but we can prove that it is usable at a certain level.

As a consequence, the evaluation of the UIs obtained by such an MDA tool poses new challenges. For example, the usability problems discussed here could be used to define design guidelines or antipatterns for analysts (e.g., van Welie et al. 2000). The explicit operationalization of these guidelines will help to obtain usable interactive applications from conceptual models. This could be done by using these guidelines or anti-patterns as the driving force for model transformation, supporting the concept of usability-driven model transformation.

The conclusions regarding the applicability of usability evaluation methods drawn from this experiment should, therefore, be generalized to a wide range of MDA-based tools. To address this issue, we plan to compare the usability of the UIs obtained with the OlivaNova tool with those obtained with other MDA-compliant tools, such as VisualWade (http://www.visual wade.com). Other future work includes the execution of several experiments to compare a UI designed

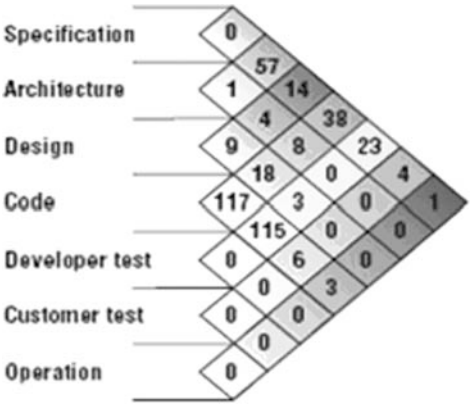


Fig. 1.7 Life-cycle phases where defects were introduced and then removed. (Source: Hall & Chapman, 2002)

by several experienced designers, and a UI generated by a MDA tool for the same interactive application starting from the same models. This would enable us to identify the differences between a manually produced UI and an automatically generated UI.

Acknowledgment The authors would like to thank Sergio España, Ignacio Panach, and Inés Pederiva for their full support while conducting this experiment. We would also like to thank the users who evaluated the system for their time and efforts. This work was partially supported by the COST Action n°294 (www.cost294.org), by the SIMILAR network of excellence (www.similar.cc). Jean Vanderdonckt would also like to thank Valencia University of Technology for its support.

References

Balzert, H., Hofmann, F., Kruschinski, V., & Niemann, Ch. (1995). The JANUS application development environment – Generating more than the user interface. In Vanderdonckt, J. (Ed.), *Computer-Aided Design of User Interfaces, Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (pp 183–208). Presses Universitaires de Namur, Namur.

Bodart, F., Hennebert, A.M., Leheureux, J.M., Vanderdonckt, J. (1995). A model-based approach to presentation: A continuum from task analysis to prototype. In *Focus on Computer Graphics Series* (pp. 77–94). Springer-Verlag.

Campbell, D.T., & Fiske, D.W. (1959). Convergent and discriminant validation by the multitrait-multimethod matrix. *Psychological Bulletin, 56*, 81–105

Care Technologies (2005) http://www.care-t.com/products/index.html. Last access: Nov. 2005.

Care Technologies (2006a) http://www.care-t.com/downloads, *OlivaNova Modeller User Manual*. Last update: Jan. 2006.

Care Technologies (2006b) http://www.care-t.com/downloads, *OlivaNova .NET Business Logic for Scalable Transactional Architecture Manual*. Last update: Feb. 2006.

Dumas, J.S., & Redish, J.C. (1999). A *practical guide to usability testing*. Intellect Ltd (UK).

Eisenstein, J., Vanderdonckt, J., & Puerta, A.R. (2001). Model-based user-interface development techniques for mobile computing. In *Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'01* (pp. 69–76). New York: ACM Press.

Frøkjær, E., Hertzum, M., & Hornbæk, K. (2000). Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated? In *Proc. of the ACM Conference on Human Factors in Computing Systems CHI'00* (pp. 345–352). New York: ACM Press.

Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J.B., Gray, P.D., Cooper, R., Goble, C.A., & Silva, P.P. (2001). Teallach: A Model-based user interface development environment for object databases. *Interacting with Computers, 14(1),* 31–68

Hall, A., & Chapman, R. (2002). Correctness by construction: Developing a commercial secure system. *IEEE Software, 19*(1), 18–25

Holzinger, A. (2005) Usability engineering methods for software developers. *Communications of the ACM, 48(1)*, 71–74

ISO 9241-11 (1998). *Ergonomic requirements for office work with visual display terminals (VDTs) Part 11: Guidance on usability*.

Janssen, C., Weisbecker, A., & Ziegler, J. (1993). Generating user interfaces from data models and dialogue net specifications. In *Proc. of the ACM Conference on Human Aspects in Computing Systems CHI'93* (pp. 418–423), Amsterdam. New York: ACM Press

Jarke, M., & Vassiliou, Y. (1985). A framework for choosing a database query language. *ACM Computing Surveys, 17(3),* 313–370

Jespersen, J.W., Linvald, J. (2003). Investigating user interface engineering in the model driven architecture. In M.B. Harning & J. Vanderdonckt (Eds.), *Closing the gap: Software engineering*

and human-computer interaction, Proc. of the 2nd IFIP TC 13 WG2.7/13.4 Workshop on Integrating Software Engineering and Human-Computer Interaction, IFIP (pp. 63–66). Accessible at http://www.se-hci.org/bridging/interact/proceedings.htm

Lewis, J., & Rieman, J. (1993). *Task-centered user interface design: A practical introduction*. Accessible at ftp.cs.colorado.edu in /pub/cs/distribs/clewis/HCI-Design-Book). <*>document</*>

Lonczewski, F., & Schreiber, S. (1996). The FUSE-system: an integrated user interface design environment. In J. Vanderdonckt (Ed.), *Computer-aided design of user interfaces, Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (pp. pp 37–56). Namur: Presses Universitaires de Namur.

Lyytinen, K., & Yoo, Y. (2002). Issues and challenges in ubiquitous computing. *Communications of the ACM,45*(12), 62–65

MDA: http://www.omg.org/mda. Last access: Dec. 2005

Molina, P.J., Meliá, S., & Pastor, O. (2002). JUST-UI: A user interface specification model. In Ch. Kolski & J. Vanderdonckt (Eds.), *Proc. of 4th Int. Conf. on Computer Aided Design of User Interfaces CADUI'02* (pp. 63–74). Dordrecht: Kluwer Academics Publishers.

Mori, G., Paternò, F., & Santoro, C. (2004). Design and development of multi-device user interfaces through multiple logical descriptions. *IEEE Transactions on Software Engineering, 30*(8), 507–520

Myers, B.A., Hudson, S.E., & Pausch, R.F. (2000). Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction, 7(1)*, 3–28.

Myers, B.A., & Rosson, M.B. (1992). Survey on user interface programming. In *Proc. of ACM Conference on Human Factors in Computing Systems CHI'92,* Monterey (pp. 195–202). New York: ACM Press.

Nielsen, J., & Landauer, T.K. (1993). A mathematical model of the finding of usability problems. In *Proc. of Conf. on Human Factors in Computing Systems INTERCHI'93* (pp. 206–213), Amsterdam. New York: ACM Press.

Nunally, J. (1978). *Psychometric Theory*(2nd *ed.*) New York: McGraw-Hill

Olson, J.R., & Olson, G.M. (1990). The growth of cognitive modeling in human-computer interaction since GOMS. *Human-Computer Interaction, 5*, 221–265.

OMG (2006). *Committed companies and their products*. Accessible at http://www.omg.org/mda/committed-products.htm

OMG (2003) *Unified Modeling Language v1.5*. http://www.omg.org/cgi-bin/doc? formal/03-03-01. Last access: Feb. 17, 2005.

Pastor, O., Gómez, J., & Insfran, E., & Pelechano, V. (2001). The OO-method approach for information systems modelling: from object-oriented conceptual modeling to automated programming. *Information Systems, 26(7):* 507–534

Puerta, A.R. (1996). The mecano project: Comprehensive and integrated support for model-based interface development. In J. Vanderdonckt (Ed.), *Computer-aided design of user interfaces, Proc. of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96* (pp 19–36). Namur: Presses Universitaires de Namur.

Puerta, A.R. (1997). A model-based interface development environment. *IEEE Software, 14(4)*, 41–47.

Silva, P.P. (2000). User interface declarative models and development environments: A survey. In *Proc. of 7th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'00, Lecture Notes in Computer Science 1946* (pp. 207–226). Berlin: Springer-Verlag.

Szekely, P.A. (1996). Retrospective and challenges for model-based interface development. In F. Bodart & J. Vanderdonckt (Eds.), *Proc. of the 3rd Int. Eurographics Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'96* (pp. 1–27), Namur. Berlin: Springer-Verlag.

Szekely, P.A., Sukaviriya, P., Castells, P., Muthukumarasamy, J., & Salcher, E. (1996). Declarative interface models for user interface construction tools: the MASTERMIND approach. In L.J. Bass & C. Unger (Eds.), *Engineering for human-computer interaction, Proc. of the IFIP TC2/WG2.7 Working Conf. on Engineering for HCI EHCI'95* (pp. 120–150), Yellowstone Park.

Tariq, N.A., & Akhter, N. (2005). *Comparison of model driven architecture (MDA) based tools.* Master Thesis, Royal institute of Technology (KTH), Stockholm, Sweden, URL: http://dis.dsv.su.se/∼emis-nat/CMDA/MainPage.htm

Vanderdonckt, J. (2005). A MDA-compliant environment for developing user interfaces of information systems. In O. Pastor & J. Cunha (Eds.), *Proc. of* 17^{th} *Int. Conference on Advanced Information Systems Engineering CAiSE'05*, Porto. Lecture Notes in Computer Science n°3520 (pp. 16–31). Berlin: Springer-Verlag.

Vanderdonckt, J., & Bodart, F. (1993) Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proc. of the ACM Conf. on Human Factors in Computing Systems INTERCHI'93* (pp. 424–429), Amsterdam. New York: ACM Press.

van Welie, M., van der Veer, G.C., & Eliëns, A. (2000). Patterns as tools for user interface design. In *Proc. of the Int. Workshop on Tools for Working with Guidelines* (pp. 313–324). London: Springer.