Chapter 3
Conceptual Modelling of Interaction

Nathalie Aquino[1], Jean Vanderdonckt[1,2], José Ignacio Panach[1], Oscar Pastor[1]
[1]Centro de Investigación en Métodos de Producción de Software, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain
[2]Université catholique de Louvain, Louvain School of Management (LSM), Place des Doyens, 1 - B-1348, Louvain-la-Neuve, Belgium
{naquino, jpanach, opastor}@pros.upv.es, jean.vanderdonckt@uclouvain.be

Abstract The conceptual model of an information system cannot be considered to be complete after just specifying the structure and behaviour of the system. It is also necessary to specify how end-users will interact with the system. Even though there are several proposals for modelling interaction, none of them have become widely known or widely used in academia and industry. After illustrating the state of the art in this field, this chapter briefly presents a practical approach with the aim of showing how interaction modelling can be faced. The presented approach is called OO-Method, a Model-Driven Engineering method that allows full functional systems to be generated from a conceptual model. The chapter explains how OO-Method supports the interaction modelling by means of its Presentation Model. Apart from this description, the chapter comments on some limitations of the Presentation Model to satisfy end-user interaction requirements related to preferences and different contexts of use. This problem is faced by distinguishing an abstract and a concrete level for interaction modelling. The abstract perspective focuses on what must be presented to end-users in order to allow their interaction with an information system, and the concrete perspective focuses on how those elements are presented. Upon the basis of a whole interaction model, abstract and concrete perspectives are separated. On the one hand, the OO-Method Presentation Model is shown to be an example of abstract interaction modelling. On the other hand, an extension based on Transformation Templates is proposed to cover the concrete interaction modelling perspective. To illustrate how both interaction modelling levels can be used, this chapter models the interaction of a photography agency system.

3.1 Introduction

The idea that the conceptual model is the code is becoming more and more a reality in software engineering and information systems design. Some explicit statements for this perspective can be found in the Conceptual Schema-Centric Development challenge [24], the Extreme Non-Programming initiative [21, 25], and the set of both academic and industrial approaches and tools proposed within the frame of Model-Driven Engineering (MDE), with the intention of providing operative solutions. Conceptually aligned with these ideas and specifically represented in this book under the term Conceptual Modelling Programming (see the manifesto, Chapter 1), we strongly believe that conceptual modelling is programming. As stated in that manifesto, *the conceptual model, with which modellers program, must be complete and holistic*. In practice, this statement requires every necessary aspect of data (structure), behaviour (function), and interaction (both component interaction and user interaction), to be adequately included.

User interaction modelling is the issue in this chapter. We are especially concerned with the answer to an apparently simple question: What are the most relevant conceptual primitives or modelling elements that should guide the construction of a conceptual interaction model? This question arises since the conceptual model community provides widely accepted and widely used data models with strong standards such as the Entity-Relationship Model [10] or UML Class Diagrams as well as widely accepted and widely used behaviour models (from the old Data-Flow Diagrams [34] to the more recent Collaboration, Sequence or Activity UML Diagrams). However, it is surprising that clear and concrete conceptual models to represent interaction have not yet been provided. There are still questions about which interaction models will allow us to face conceptual modelling of user interfaces and how these models can be properly embedded into the whole conceptual model, which includes data, behaviour, and interaction. This is particularly surprising since the answer to these questions are so evident for the data and behaviour perspectives of conceptual modelling, especially when considering the great importance of user interface design in the whole process of building an information system. Everyone accepts that a final software application is much more than a well-defined database and a set of programs that incorporate the needed functionality. If a conceptual model is to be viewed as the code of the system, every essential aspect of software must be considered, and, of course, user interface plays a basic role in this context. Going back to the Conceptual Modelling Programming manifesto, to make the goal of having a conceptual model complete and holistic a reality, the proper specification of user interface conceptual models (not only user interface sketches of the system) is strictly required. Therefore, the conceptual modelling elements behind user interface specification must be defined precisely and must be based on a corresponding ontological agreement that fixes the concepts and their associated representation and notation.

To achieve these goals, this chapter explores two aspects. First, a particular example of what user interface modelling means in terms of modelling primitives and model specification is introduced. The selected approach is the Presentation Model of OO-Method [27]. This approach constitutes a practical case of how interaction modelling from the user interface perspective is joined to data and behaviour modelling in a unified way, and how this conceptual model includes all the relevant information that is needed to face the subsequent conceptual model compilation process to obtain the corresponding software system. Conceptual primitives are introduced to show how user interface modelling can be specifically put in practice, bridging the gap between "conventional" (data- and behaviour-oriented) conceptual modelling and user interface modelling. Second, an important feature that is associated to user interface modelling is dealt with. An interaction model can fix the presentation style, but this presentation style normally needs to be adapted to the end-user's tastes and wishes. Talking about the user interface is not the same as talking about the final data and program structure. In general, end-users want to participate in defining the way in which the human-software interaction is going to be accomplished, and this cannot be done if the user interface model does not allow the conceptual model to be adapted to their particular interaction requirements. Some authors use the term "beautification" to refer to this situation [31].

A common solution for solving this problem consists in explicitly distinguishing two levels in the interaction conceptual model: an abstract level and a concrete level. This approach has been presented in several works ([9, 22, 16, 18, 39, 30], among others), and it is currently being applied in the context of user interface development according to MDE. While the abstract level focuses on the high-level perspective of the interaction, the concrete level identifies several possible representations of the abstract modelling primitives and gives modellers the chance to adapt them according to the target platform and the end-user's preferences. This distinction between abstract and concrete provides a two-level approach that makes it possible to differentiate concerns that are very important within the scope of interaction modelling. On the one hand, there are higher-level abstractions that fix the main relevant user interface properties (e.g., the set of interaction units that should conform the main menu of an application). These abstractions represent which elements are going to be shown in each interface. On the other hand, there is a more concrete level where interfaces are specified for particular software environments. This concrete model represents how the elements of the interface will be presented (e.g., the particular, concrete, presentation style chosen for presenting those main menu options to the end-users).

In accordance with these ideas, this chapter is structured in the following way: in Section 3.2, a related work analysis is presented to understand what other authors have proposed and how the interaction modelling issue is confronted from a conceptual model perspective in current MDE approaches. In Section 3.3, the Presentation Model of OO-Method is introduced as an example of how interaction modelling is properly embedded in an MDE-based software production process where conceptual models are the only key software artefacts. In Section 3.4, we propose an extension to explicitly distinguish between the abstract level and the concrete level, indicating how to accomplish this distinction in practice. The chapter ends with concluding remarks and the list of references used.

3.2 Related Work

Since its inception in the eighties, the domain of Human-Computer Interaction (HCI) has undergone a dramatic increase in research and development, arriving to the point where it is recognized that interaction should also be modelled just like any other aspect of an interactive system. During more than a decade, several model-based approaches have evolved in parallel in order to cope with the different challenges raised by the design and development of user interfaces in continuously evolving technological settings. We can identify various generations of works in this area [36]. The first generation of model-based approaches focused basically on deriving abstractions for graphical user interfaces (see, for example, UIDE [13]). At that time, user interface designers focused mainly on identifying relevant aspects for this kind of interaction modality. Then, the approaches evolved into a second generation that focused on expressing the high-level semantics of the interaction: this was mainly supported through the use of task models and associated tools, which were aimed at expressing the activities that the users intend to accomplish while interacting with the application (see, for example, Adept [15], GTA [42], ConcurTaskTrees (CTT) [29], Trident [5], Humanoid [35]). Since these times, a consensus has been reached in the community to structure interaction modelling according to different levels of abstraction in almost the same way as in other areas (i.e. database engineering and information systems).

In this context, one of the most recent works is the Cameleon Reference Framework [9]. Cameleon structures the development life cycle into four levels of abstraction, starting from task specification to a running interface (see Figure 3.1):

- The Task and Concepts level: This level considers (a) the logical activities (tasks) that need to be performed in order to reach the end-users' goals; and (b) the domain objects manipulated by these tasks.
- The Abstract User Interface (AUI): This level represents the user interface in terms of interaction spaces (or presentation units), independently of which interactors are available and even independently of the modality of interaction (e.g., graphical, vocal, haptic).
- The Concrete User Interface (CUI): This level represents the user interface in terms of "concrete interactors", which depend on the type of platform and media available and which have a number of attributes that more concretely define how the user interface should be perceived by the end-user.
- The Final User Interface (FUI): This level consists of source code, in any programming or mark-up language (e.g., Java, HTML5, VoiceXML, X+V). It can then be interpreted or compiled.

These levels are structured with both a relationship of reification going from a more abstract level to a more concrete one and a relationship of abstraction going from a more concrete level to a more abstract one. There can also be a relationship of translation between models at the same level of abstraction, but conceived for different contexts of use. These relationships are depicted in Figure 3.1.

There are other approaches for representing the interaction based on UML models (http://www.uml.org/). Wisdom [23] is a UML-based software engineering method that proposes an evolving use-case-based method in which the software system is iteratively developed by incremental prototypes until the final product is obtained. The UML notation has been enriched with the necessary stereotypes,

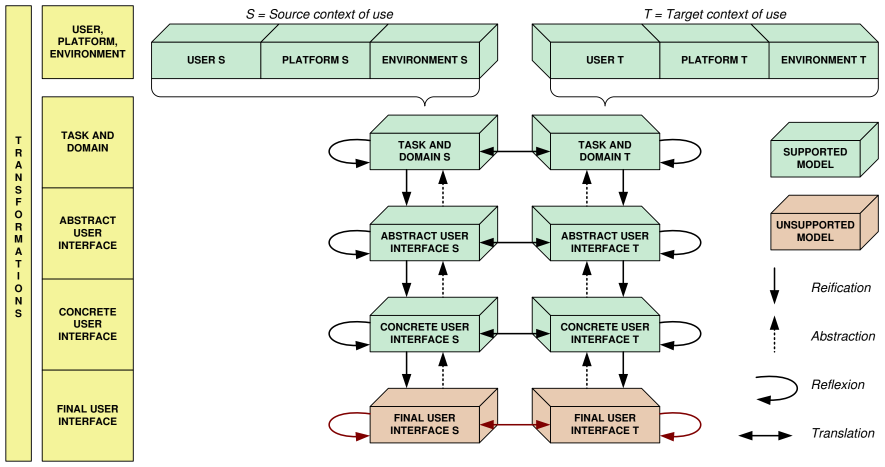


Fig. 3.1 Relationships between components in the Cameleon Reference Framework

labelled values, and icons to allow user-centered development and a detailed user interface design. Three of its models are concerned with interaction modelling at different stages: the Interaction Model, at the analysis stage; and the Dialog and Presentation models during the design stage, as refinements of the Interaction Model. Another important proposal is UMLi [12], which is a set of user interface models that extends UML to provide greater support for user interface design. UMLi introduces a new diagram: the User Interface Diagram, which can be considered to be the first reliable proposal of UML to formally capture the user interface. However, the models are so detailed that the modelling turns out to be very difficult. Middle-sized models are very hard to specify, which may be the reason why UMLi has not been adopted in industrial environments.

In addition, there are several proposals that model the interaction abstractly by means of the above-mentioned ConcurTaskTrees (CTT) notation [29]. Examples of these types of proposals are TERESA [22] and SUIDT [4]. TERESA (Transformation Environment for inteRactivE Systems representAtions) is a tool that supports transformations in a top-down manner, providing the possibility of obtaining interfaces for different types of devices from logical descriptions. This tool starts with an overall envisioned task model and then derives concrete and effective user interfaces for multiple devices. SUIDT (Safe User Interface Design Tool) is a tool that automatically generates interfaces using several models that are related to each other: a Formal Functional Core, an Abstract Task Model, and a Concrete Task Model. CTT notation is used in the Abstract Task Model and in the Concrete Task Model.

We have mentioned different types of approaches for representing the interaction in an abstract manner; however, a suitable language that enables integration within the development environment is still needed. For this purpose, the notion of User Interface Description Language (UIDL) has emerged to express any of the aforementioned models. A UIDL is a formal language used in HCI to describe a particular user interface independently of any implementation technology. As such, the user interface might involve different interaction modalities (e.g., graphical, vocal, tactile, haptic, multimodal), interaction techniques (e.g., drag and drop), or interaction styles (e.g., direct manipulation, form fillings, virtual reality). A common fundamental assumption of most UIDLs is that user interfaces are modelled as algebraic or model-theoretic structures that include a collection of sets of interaction objects together with behaviours over those sets.

The design process for a UIDL encompasses the definition of the following artefacts:

- Semantics: This expresses the context, meaning and intention of each abstraction captured by the underlying meta-models on which the UIDL is based.
- Abstract Syntax: This is a syntax that makes it possible to define user interface models (in accordance with the UIDL semantics) independently of any representation formalism.
- Concrete Syntax/es: These are (one or more) concrete representation formalisms intended to syntactically express user interface models.
- Stylistics: These are graphical and textual representations of the UIDL abstractions that maximize their representativity and meaningfulness in order to facilitate understanding and communication among different people.

In conclusion, there are a lot of proposals to represent the interaction abstractly, as we have seen in this section. Each proposal is based on a specific notation, like UML or CTTs. However, as far as we know, none of these proposals supports interaction modelling together with persistence and functionality. Existing proposals can generate interfaces but not fully functional systems. Moreover, all the works mentioned in this section have seldom been used in industrial environments. In the next section, we present an approach that has solved both of these limitations: the modelling of interaction in a holistic conceptual modelling approach and the practical applicability of interaction modelling in an industrial context. Furthermore, we show how the interaction can be represented by means of conceptual primitives.

3.3 The Interaction Model of OO-Method

OO-Method [27] is an object-oriented method which allows the automatic generation of software applications from conceptual models. These conceptual models are structured in four system views: (1) the Object Model specifies the static properties of the interactive application by defining the classes and their relationships; (2) the Dynamic Model controls the application objects by defining their life cycle and interactions; (3) the Functional Model describes the semantics of object state changes; and (4) the Presentation Model specifies the user interface.

OO-Method is supported by a commercial software suite named OlivaNOVA that was developed by CARE Technologies (http://www.care-t.com). OlivaNOVA edits the various models involved and applies subsequent transformations until the final code of a fully functional application (persistence, logic, and presentation) is generated for different computing platforms: C# or ASP running on .NET or .NET 2.0; and EJB, JSP, or JavaServer Faces running on Java. Thus, OO-Method defines a holistic conceptual model which includes the interaction perspective as well as the structural and behavioural ones. Furthermore, it is currently being used successfully in an industrial environment.

This section presents the conceptual primitives of the OO-Method Presentation Model, which allow a user interface to be modeled in a convenient way. These primitives have enough expressiveness to represent any management information system interface. Furthermore, an illustrative example related to a photography agency system is presented throughout this section and the next one. This agency manages illustrated reports for distribution to newspaper editorials. The agency operates with photographers who work as independent professionals.

The OO-Method Presentation Model is structured with a set of interaction patterns that were defined in [20]. These interaction patterns are ordered in three levels (see Figure 3.2):

- Level 1 - Hierarchical Action Tree (HAT): organizes the access to the system functionality through a tree-shaped abstraction.
- Level 2 - Interaction Units (IUs): represent the main interactive operations that can be performed on the domain objects (executing a service, querying the population of a class, and visualizing the details of a specific object).
- Level 3 - Elementary Patterns (EPs): constitute the building blocks from which IUs are constructed.

In the next three subsections, we provide more details about the interaction patterns from the three levels, going from the most specific to the most general ones.

3.3.1 Elementary Patterns

Elementary Patterns (EPs) constitute the primitive building blocks to build IUs. They represent specific aspects of the interaction between a human and a system and cannot be combined in an arbitrary way; on the contrary, each of them are applicable in specific IUs.

In the current OO-Method Presentation Model, there are eleven EPs that can be related to their corresponding relevant IUs (see Figure 3.2). These EPs are the following:

- Introduction: captures the relevant aspects of data to be entered by the end-user. Interaction aspects that can be specified include edit masks and valid value ranges.
- Defined selection: enables the definition (by enumeration) of a set of valid values for an associated model element.

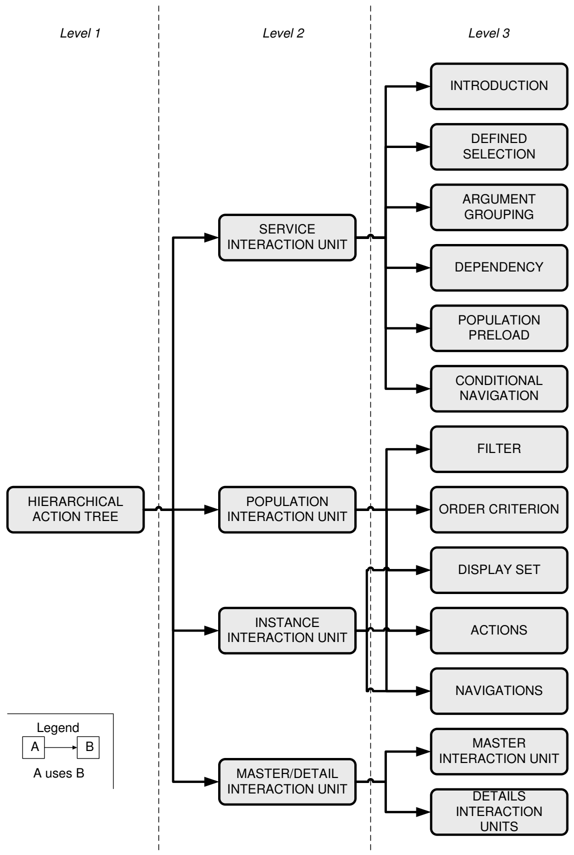


Fig. 3.2 OO-Method Presentation Model

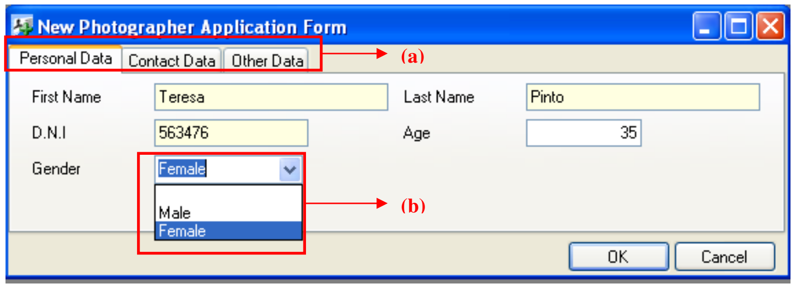
- Argument grouping: defines the way in which input arguments for a given service are presented to the end-user allowing these input arguments to be arranged in groups and subgroups.
- Dependency: enables dependency relationships to de defined between the value or state of an input argument of a service and the value or state of other input argument of the same service. The definition is based on ECA-type rules (event, condition, action).
- Population preload: allows the designer to specify that the selection of an object as an input argument of a service will be carried out with or without changing the interaction context.
- Conditional navigation: allows navigation to different IUs after the successful or failed execution of a service. In order to specify which IU to navigate to, it is also necessary to establish a condition that must hold after the execution of the service.

- Filter: defines a selection condition over the population of a class, which can be used to restrict the object population of the class, thereby facilitating further object search and selection operations.
- Order criterion: defines how the population of a class is to be ordered. Ordering is done on the values of one or more properties of the objects, taking into account ascending/descending options.
- Display set: determines which properties of a class are to be presented to the user and in what order.
- Actions: define the set of available services that can be performed on the objects of a given class.
- Navigations: determine the information set that can be accessed via navigation of the structural relationships found in an initial class.

3.3.2 Interaction Units

An Interaction Unit (IU) describes a particular scenario of the user interface through which users are able to carry out specific tasks. In the OO-Method approach, there are three different basic kinds of interaction scenarios: execution of a service, manipulation of one object, and manipulation of a collection of objects. For each one of these basic interaction scenarios, the OO-Method approach proposes a specific IU that is appropriate for handling it. A fourth IU is proposed to combine the other IUs. The four IUs are the following (see Figure 3.2):

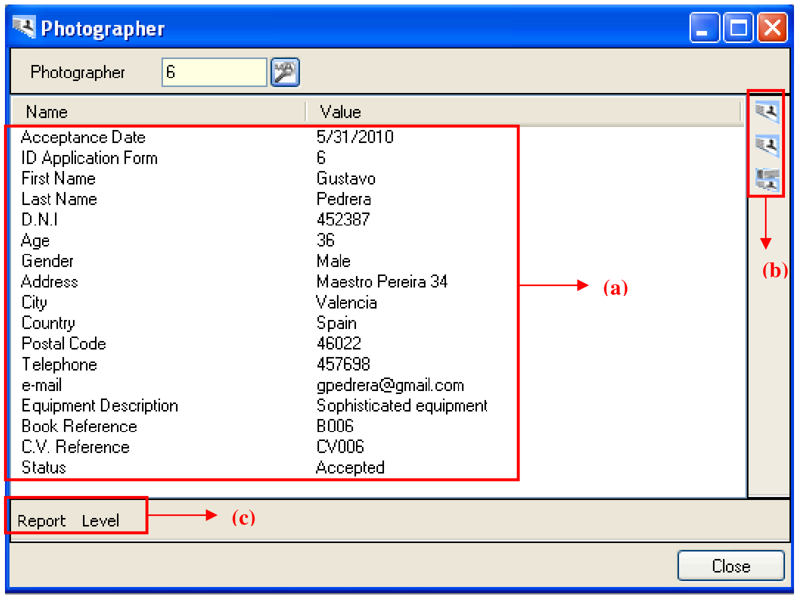
- Service IU: enables a scenario to be defined in which the user interacts with the system in order to execute a service. The user must provide the arguments and launch the service. Six of the EPs can be used to complete the specification of a Service IU: introduction, defined selection, argument grouping, dependency, population preload, and conditional navigation (see Figure 3.2). Figure 3.3 shows the final user interface generated from a Service IU. The user interface for this Service IU allows a photographer to fill in an application form for working in a photography agency. The photographer must provide personal and contact data as well as data related to its profesional equipment.
- Instance IU: represents a scenario in which information about a single object is displayed, including the list of services that can be executed on it, as well as the scenarios of related information to which the user can navigate. All this information is structured by means of three EPs: display set, actions, and navigations (see Figure 3.2). Figure 3.4 shows the final user interface generated from an Instance IU. The user interface for this Instance IU shows data related to a photographer of the agency.
- Population IU: represents an interaction scenario where multiple objects are presented. Includes the appropriate mechanisms to do the following: select and sort objects, choose the information and available services to be shown, and list other scenarios that can be reached. All this information is structured by means of five



First Name	Teresa	Last Name	Pinto
D.N.I	563476	Age	35
Gender	Female		

Male
Female

Fig. 3.3 User interface generated from a Service IU with argument groupings (a), and defined selection (b)



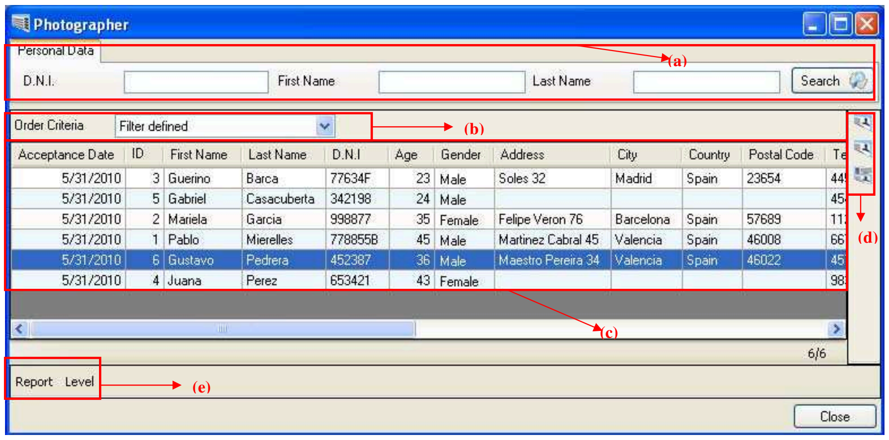
Photographer 6

Name	Value
Acceptance Date	5/31/2010
ID Application Form	6
First Name	Gustavo
Last Name	Pedrera
D.N.I	452387
Age	36
Gender	Male
Address	Maestro Pereira 34
City	Valencia
Country	Spain
Postal Code	46022
Telephone	457698
e-mail	gpedrera@gmail.com
Equipment Description	Sophisticated equipment
Book Reference	B006
C.V. Reference	CV006
Status	Accepted

Fig. 3.4 User interface generated from an Instance IU with display set (a), actions (b), and navigations (c)

EPs: filter, order criteria, display set, actions, and navigations (see Figure 3.2). Figure 3.5 shows the final user interface generated from a Population IU. The user interface for this Population IU shows data related to multiple photographers of the agency at the same time.

- Master/Detail IU: presents the user with a scenario for the interaction with multiple collections of objects that belong to different interrelated classes. This forms a composite scenario in which two kinds of roles can be defined: a master role, which represents the main interaction scenario; and detail roles, which represent



Acceptance Date	ID	First Name	Last Name	D.N.I	Age	Gender	Address	City	Country	Postal Code	Te
5/31/2010	3	Guerino	Barca	77634F	23	Male	Soles 32	Madrid	Spain	23654	44
5/31/2010	5	Gabriel	Casacuberta	342198	24	Male					45
5/31/2010	2	Mariela	Garcia	998877	35	Female	Felipe Veron 76	Barcelona	Spain	57689	11
5/31/2010	1	Pablo	Mierelles	778855B	45	Male	Martinez Cabral 45	Valencia	Spain	46008	66
5/31/2010	6	Gustavo	Pedrera	452387	36	Male	Maestro Pereira 34	Valencia	Spain	46022	45
5/31/2010	4	Juana	Perez	653421	43	Female					98

Fig. 3.5 User interface generated from a Population IU with filter (a), order criterion (b), display set (c), actions (d), and navigations (e)

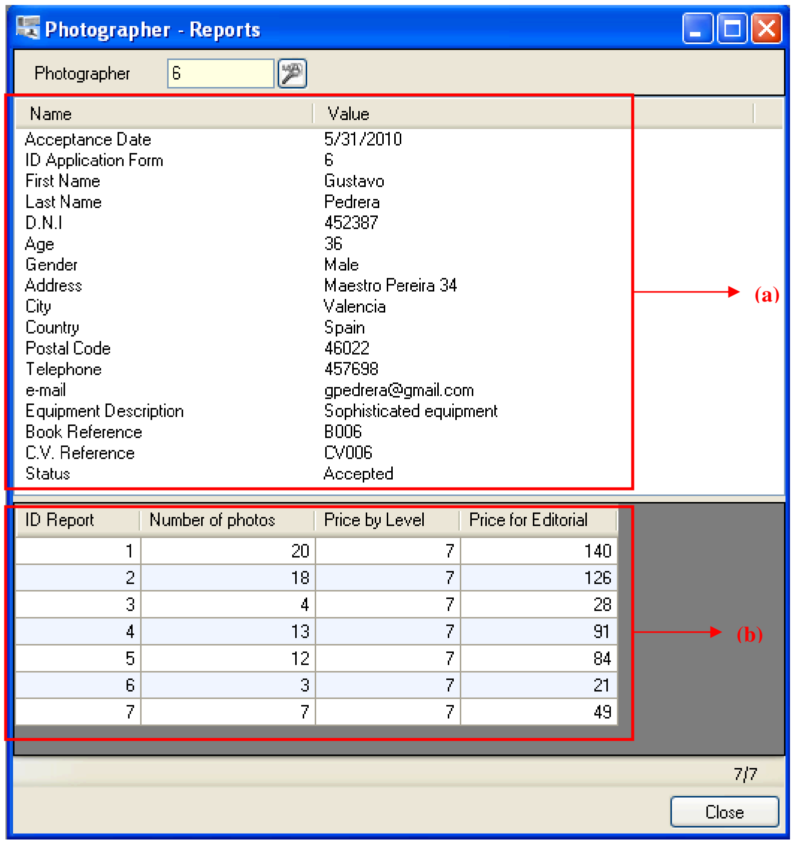
secondary, subordinated interaction scenarios that are kept synchronized with the master role (see Figure 3.2). Figure 3.6 shows the final user interface generated from a Master/Detail IU in which the master role corresponds to an Instance IU, which shows data related to a photographer of the agency, and the detail role corresponds to a Population IU, which shows the list of reports related to the photographer.

The user interfaces depicted in Figure 3.3, Figure 3.4, Figure 3.5, and Figure 3.6 have been generated by OlivaNOVA for the desktop .NET platform.

3.3.3 Hierarchical Action Tree

Once the interaction scenarios have been described through the corresponding IUs, it is necessary to determine how these IUs are to be structured, organized, and presented to the user. This structure will characterize the top level of the user interface, establishing what could be described as the main menu of the application. The Hierarchical Action Tree (HAT) serves this purpose.

The HAT defines an access tree that follows the principle of gradual approximation to specify the manner in which the interactive user can access system functionality. This is achieved by arranging actions into groups and subgroups by using a tree-abstraction, from the most general to the most detailed. Intermediate (i.e., non-leaf) nodes in the tree are simply grouping labels, whereas tree leaves reference pre-existing IUs (see Figure 3.2).



Name	Value
Acceptance Date	5/31/2010
ID Application Form	6
First Name	Gustavo
Last Name	Pedrera
D.N.I	452387
Age	36
Gender	Male
Address	Maestro Pereira 34
City	Valencia
Country	Spain
Postal Code	46022
Telephone	457698
e-mail	gpedrera@gmail.com
Equipment Description	Sophisticated equipment
Book Reference	B006
C.V. Reference	CV006
Status	Accepted

ID Report	Number of photos	Price by Level	Price for Editorial
1	20	7	140
2	18	7	126
3	4	7	28
4	13	7	91
5	12	7	84
6	3	7	21
7	7	7	49

Fig. 3.6 User interface generated from an Master/Detail IU with master role (a), and detail role (b)

3.4 Explicitly Distinguishing Abstract and Concrete Interaction Modelling in OO-Method

The OO-Method Presentation Model constitutes a unified interaction model in which there is no explicit distinction between an abstract level and a concrete level. This model can be considered a good starting point for adequately modelling interaction since it provides a good basis to include user interface generation in the conceptual model compilation process. However, it still presents an important problem: the interaction style of the resultant software application is fixed by the model compiler, and there is no way to adapt the presentation style to the particular needs and individual tastes of end-users. In this section, we show how to make this distinction feasible. We also extend the above approach in this direction, and add a concrete level that incorporates decisions related to platforms and users. In particular, the Transformation Templates approach is presented as a means for concrete interaction modelling.

3.4.1 Abstract Interaction Modelling

As explained in Section 3.3, the OO-Method Presentation Model provides primitives that allow the designer to define user interfaces in a homogeneous and platform-independent way. All of its interaction patterns, from the three levels, capture the necessary aspects of the user interface without delving into implementation issues. In other words, the OO-Method Presentation Model focuses on what type of user interaction is desired, and not on how this interaction will be implemented in the resulting software product. Therefore, the OO-Method Presentation Model can be considered an abstract model from which the model compiler can automatically generate a user interface for different interaction modalities and platforms.

3.4.2 Concrete Interaction Modelling: Transformation Templates

At the abstract level, the OO-Method Presentation Model does not provide primitives that allow the structure, layout and style of user interfaces to be expressed. These decisions are delegated to the model compiler and are hard-coded in it. Thus, design knowledge and presentation guidelines are implicit and fixed in the tool that performs the model-to-code transformation and cannot be edited or customized. Thus, even though different final user interface implementations are potentially valid when moving from the abstract to the final user interface, it is not possible to adapt the user interface generation according to end-user requirements and preferences. This results in the generation of predetermined user interfaces, all of which look alike, and which may not always satisfy the end-user.

Because of these issues, it has been necessary to extend the OO-Method Presentation Model with a new concrete level that provides the required expresiveness in order to enable the customization of user interfaces before their generation. An approach based on Transformation Templates has been defined for this purpose.

A Transformation Template [3, 2] aims to specify the structure, layout and style of a user interface according to preferences and requirements of end-users as well as according to the different hardware and software computing platforms and environments in which the user interface will be used.

A Transformation Template is composed of parameters with associated values that parameterize the transformations from the OO-Method Presentation Model to code. Figure 3.7 illustrates the use of a Transformation Template with OO-Method. The model compiler takes a Presentation Model and a Transformation Template as input. The Transformation Template provides specifications that determine how to transform the Presentation Model to code. The specifications are expressed by

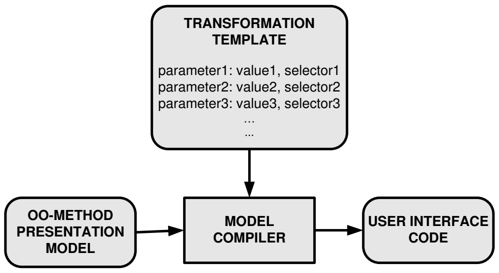

TRANSFORMATION TEMPLATE
parameter1: value1, selector1
parameter2: value2, selector2
parameter3: value3, selector3
...
...
OO-METHOD PRESENTATION MODEL
MODEL COMPILER
USER INTERFACE CODE

Fig. 3.7 An OO-Method Presentation Model and a Transformation Template are inputs for the model compiler

means of parameters with values and selectors. Selectors define the set of elements of the OO-Method Presentation Model that are affected by the value of the parameter. The transformation engine follows the specifications to generate the code.

In this way, Transformation Templates externalizes the design knowledge and presentation guidelines and makes them customizable according to the characteristics of the project that is being carried out. Transformation Templates can then be reused in other projects with similar characteristics.

Even though the idea behind the Transformation Templates is based on Cascading Style Sheets [6], there are significant differences between the two approaches with the main one being that Transformation Templates are applied to user interface models and not directly on the code. Another difference is that Transformation Templates are supposed to be used in an MDE process for user interface generation for different contexts of use, not only for web environments.

The main concepts or primitives that characterize the Transformation Templates approach are depicted in Figure 3.8. There are concepts related to context, to user interface models, and to the Transformation Templates themselves. All these concepts are explained bellow.

3.4.2.1 Context

- Context (see Figure 3.8): refers to the context of use of an interactive system. We have defined context according to the Cameleon Reference Framework [9], which is widely accepted in the HCI community. According to this framework, a context of use is composed of the stereotype of a user who carries out an interactive task with a specific computing platform in a given surrounding environment. The purpose of conceptualizing context is that we want it to be possible to define different Transformation Templates for different contexts of use.

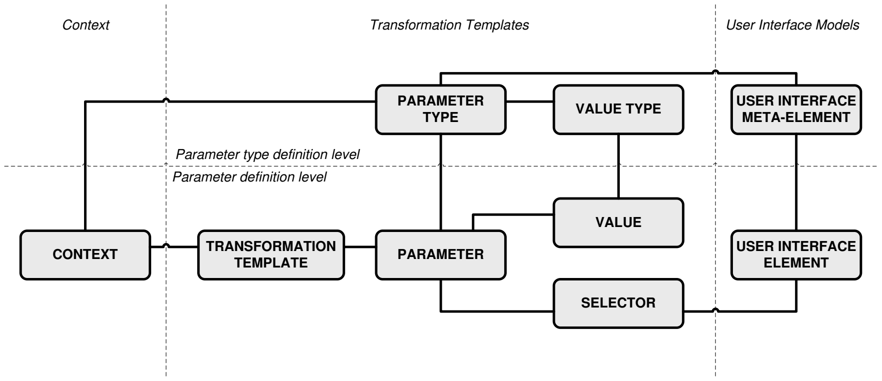


Fig. 3.8 Main concepts of the Transformation Templates approach

3.4.2.2 User Interface Models

The Transformation Templates approach makes use of two concepts related to user interface models (see Figure 3.8):

- User Interface Meta-Element: represents, in a generic way, any of the OO-Method interaction patterns presented in Section 3.3.
- User Interface Element: represents an element of the OO-Method Presentation Model, that is, a specific instance of any of the above mentioned interaction patterns.

It is important to note that even though in this chapter we are presenting the Transformation Templates approach as an extension of OO-Method, it can also be used with other MDE approaches related to user interface development. In fact, the User Interface Meta-Element is a generic representation of any meta-element of a user interface meta-model just as the User Interface Element is a generic representation of any element of a user interface model.

3.4.2.3 Transformation Templates

With regard to concepts specifically related to the Transformation Templates approach, we distinguish two levels: one in which parameter types are defined, and another one in which the previously defined parameter types are instantiated as parameters in a Transformation Template.

In the parameter type definition level, there are two concepts (see Figure 3.8):

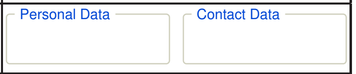
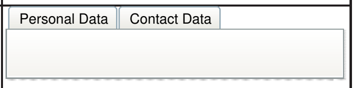
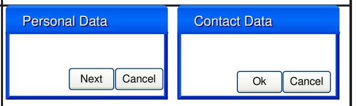
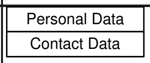
- Value Type: refers to a specific data type (e.g., integer, URI, colour, etc.) or to an enumeration of the possible values that a parameter type can assume.
- Parameter Type: represents a design or presentation option related to the structure, layout, or style of the user interface. We can distinguish between low-level

and high-level parameter types. Low-level ones operate at the attribute level of user interfaces; for instance, colour or font type are low-level parameter types related to style. High-level parameter types operate at the concept level of user interfaces and can be used to specify the structure of the user interface, the type of components (containers, widgets) that will be used, or the alignment of the components. Defining a parameter type subsumes specifying the list of user interface meta-elements that are affected by, as well as its value type. A parameter type, with all or a set of its possible values, can be implemented in different contexts of use. In order to decide about these implementations, we propose that each possible value receive an estimation of its importance level and its development cost for different relevant contexts of use. In this way, possible values with a high level of importance and a low development cost can be implemented first in a given context, followed by those with a high level of importance and a high development cost, and so on. Possible values with a low level of importance and a high development cost would not have to be implemented in the corresponding context. For each relevant context of use, usability guidelines can be assigned to each possible value of a parameter type. These guidelines will help user interface designers in choosing one of the possible values by explaining under what conditions the values should be used.

Table 3.1 shows an example of the definition of a parameter type named *grouping layout for input arguments*. This paramter type is useful for deciding how to present the input arguments of a service that have been grouped using the Argument Grouping interaction pattern presented in Section 3.3.1. Table 3.1 (a) shows that this parameter type affects two interaction patterns of the OO-Method Presentation Model. It also shows that four different possible values have been defined. Table 3.1 (b) shows that the parameter type has been associated to two contexts of use: a desktop platform and a mobile one. For each context of use and each possible value the importance level and development cost have been estimated. Table 3.1 (c) presents a list of usability guidelines for the desktop context and each possible value of the parameter type. These usability guidelines have been proposed from an extraction from [14].

In the parameter definition level, there are four concepts (see Figure 3.8):

- Transformation Template: gathers a set of parameters for a specific context of use.
- Parameter: each parameter of a Transformation Template corresponds to a parameter type and has both a value and a selector.
- Value: is an instance of a value type. The value of a parameter corresponds to a possible value of the corresponding parameter type.
- Selector: delimits the set of user interface elements that are affected by the value of a parameter. We have defined different types of selectors that allows the designer to choose: a specific user interface element; all the user interface elements of a certain type; the first or last element contained in a specific type of user interface element; other options.

Parameter Type			
Name	**Affects**	**Possible values enumeration**	
		Value	**Graphical description**
Grouping layout for input arguments	Two patterns of the OO-Method Presentation Model: Service Interaction Unit and Argument Grouping	group box	
		tabbed dialog box	
		wizard	 Personal Data Next Cancel Contact Data Ok Cancel
		accordion	

(a)

	Contexts			
	SW: C# on .NET - HW: laptop or PC		SW: iPhone OS - HW: iPhone	
Possible value	**Importance level**	**Development cost**	**Importance level**	**Development cost**
group box	high	low	high	low
tabbed dialog box	high	low	medium	medium
wizard	medium	medium	low	high
accordion	low	medium	medium	medium

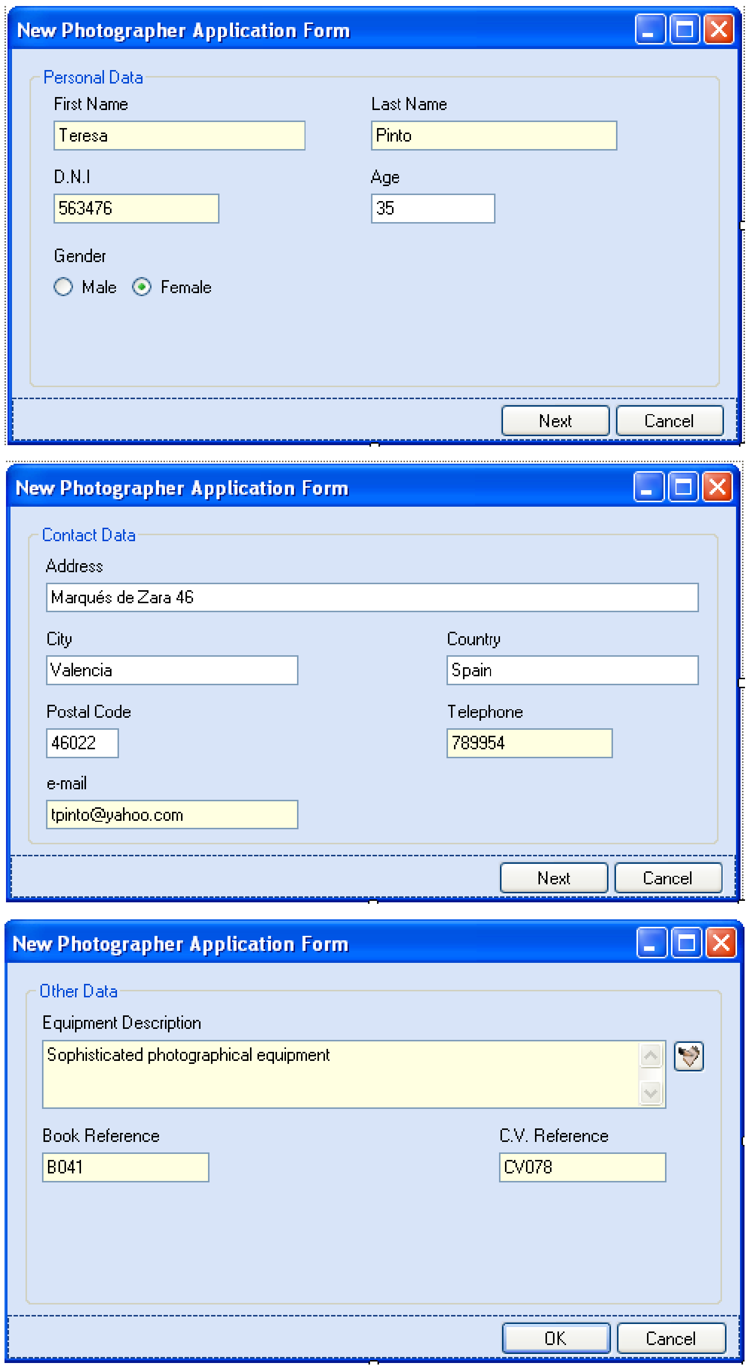
(b)

Possible value	**Usability guidelines (for desktop context)**
group box	Visual distinctiveness is important. The total number of groups will be small.
tabbed dialog box	Visual distinctiveness is important. The total number of groups is not greater than 10.
wizard	The total number of groups is between 3 and 10. The complexity of the task is significant. The task implies several critical decisions. The cost of errors is high. The task must be done infrequently. The user lacks the experience it takes to complete the task efficiently.
accordion	Visual distinctiveness is important. The total number of groups is not greater than 10.

(c)

Table 3.1 Parameter type: grouping layout for input arguments

Figure 3.9 represents the user interface that could be obtained for the Service IU that was presented before, in Figure 3.3, if the parameter *grouping layout for input arguments* is applied with value *wizard* (see Table 3.1) and if the following three parameters are also applied: a parameter for specifying the widget to be used to display defined selections with value *radio button*; a parameter for specifying the alignment of labels with value *vertical*; and a parameter for specifying the background colour.



New Photographer Application Form

Personal Data

First Name: Teresa	Last Name: Pinto
D.N.I: 563476	Age: 35

Gender: Male [✓ Female]

New Photographer Application Form

Contact Data

Address: Marqués de Zara 46

City: Valencia	Country: Spain
Postal Code: 46022	Telephone: 789954
e-mail: tpinto@yahoo.com	

New Photographer Application Form

Other Data

Equipment Description: Sophisticated photographical equipment

Book Reference: B041	C.V. Reference: CV078

Fig. 3.9 User interface that could be generated from a Service IU after applying different parameters

3.5 Conclusion

This chapter emphasizes the importance of interaction modelling on the same level of expressiveness as any other model involved in the development life cycle of an interactive application. In the same way a conceptual model of the domain could be used to derive a database for a future application, a conceptual model of the interaction could be used to derive a user interface for this same application [37]. A system with a suitable functionality and persistence may be rejected by end-users if the interface does not satisfy their expectations. Therefore, the designer must be provided with the suitable conceptual primitives to represent every relevant characteristic of the final interface; otherwise, a complete code generation from a conceptual model cannot become a reality.

Today, the community has reached a level of progress in which this has now become a reality that goes beyong mere prototypes. In the past, *model-based approaches* have been exploited to capture the essence of a user interface into a conceptual model of this user interface that will be later used for design, specification, generation, and verification. More recently, *model-driven engineering* (MDE) approaches have been introduced in order to make the user interface development life cycle more precise, rigorous, and systematic. The main difference between model-based approaches and model-driven engineering approaches [40, 41] is that in the former, only models are used while in the latter all models obey to a meta-model that is itself defined according to a same meta-meta-model. Similarly, all operations are captured through transformations that are themselves compliant with the same meta-model as opposed as no meta-model in the former approaches. Not all model-based approaches for user interface development could be considered as compliant with Model-Driven Architecture (MDA) [40].

Indeed, the following MDE/MDA definition was approved unanimously by 17 participants of the ORMSC -Object and Reference Model Subcommittee of the Architecture Board of the Object Management Group (OMG)- plenary session meeting in Montreal on 23-26 August 2004. The stated purpose of this paragraph was to provide principles to be followed in the revision of the MDA guide:

"MDA is an OMG initiative that proposes to define a set of non-proprietary standards that will specify interoperable technologies with which to realize model-driven development with automated transformations. Not all of these technologies will directly concern the transformation involved in MDA. MDA does not necessarily rely on the UML, but, as a specialized kind of MDD (Model-Driven Development), MDA necessarily involves the use of model(s) in development, which entails that at least one modeling language must be used. Any modeling language used in MDA must be described in terms of the MOF (MetaObject Facility) language to enable the metadata to be understood in a standard manner, which is a precondition for any activity to perform automated transformation".

This definition is now completely applicable to some MDE approaches for interaction modelling, such as OO-Method and its Presentation Model presented in this chapter. Taking this Presentation Model as input, we state that the interaction modelling must be divided into two views: abstract [27, 39, 38] and concrete [3, 2].

The abstract view represents *what* will be shown in each interface. This view corresponds to the Presentation Model of OO-Method, which represents the interface independently of the platform and the design. The concrete view represents *how* the elements will be shown in each interface. This model is built by means of Transformation Templates. At first glance, designers might be concerned that more effort on their part is required for modelling the concrete level. However, this problem can be resolved thanks to the use of default Transformation Templates for a specific context of use. Once the abstract interaction model has been specified, the concrete interaction model can be determined by just choosing the default Transformation Template for the context of use in which the information system is going to be used. These default Transformation Templates must be designed only once, and then can be reused. Designers might only has to change the value and/or scope of some parameters in order to adequate the concrete modelling to end-user requirements.

Future avenues of this work include:

- Integration with requirements engineering: we plan to develop a method to capture interaction requirements that is compliant with holistic development based on conceptual models. These requirements would help the designer to determine the users needs and preferences in order to guide the interaction modelling. The capture of requirements would be based on tasks, which is the notation that is most commonly used in the HCI community.
- Inclusion of a usability model in the transformation process: we will include usability characteristics in both the abstract and concrete interaction models. These characteristics will help the designer to build quality systems according to usability guidelines and heuristics. This will be helpful not only for evaluating usability during the transformation process, but also to guarantee to some extent that user interfaces issued by this approach are somewhat usable by construction [1] so as to provide a general computational framework for user interfaces [32].
- Building various transformation sets for various development paths: we will build new transformation sets that would support other development paths [17] than merely forward engineering. For instance, ReversiXML [8, 7] performs reverse engineering of web pages into a concrete interface model expressed in UsiXML [18] by using derivation rules, but not transformation rules. Similarly, MultimodaliXML [33] generates multimodal user interfaces based on the same conceptual models, but involve other sets of transformation rules.
- Building multi-fidelity editors for each model: we plan to develop model editors that enable modelers to rely on different levels of fidelity, not just high-fidelity [19], for instance by sketching the model [11], ranging from low fidelity to high fidelity.

As for any MDA approach, it is crucial to develop any work that contributes to obtain a low threshold, a high ceiling, and wide walls as much as possible to expand the capabilities of expressiveness and their transformation into a larger gamma of user interfaces. This is reflected in Figure 3.10: the first generation of MDA software usually suffered from a high threshold (they required a high amount of resources to get some results), a low ceiling (the capabilities of the user interface generated

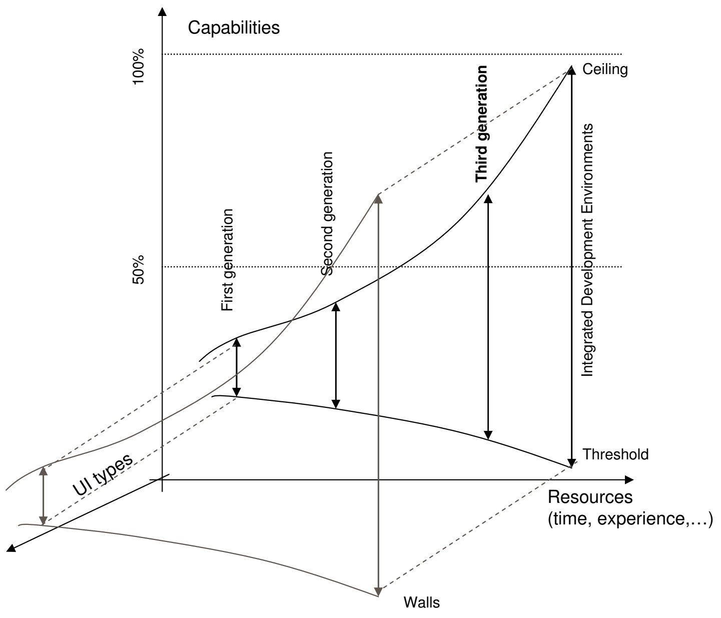


Fig. 3.10 Low threshold, high ceiling, and wide walls as determinants of a MDA approach

were limited), and narrow walls (there was only one user interface generated for one computing platform). The second generation improved this situation by lowering the threshold, increasing the ceiling, and enlarging the walls. Right now, we are in the third generation where user interface capabilities have been expanded for multiple computing platforms and contexts of use.

This race is to be continued.

Acknowledgements We gratefully acknowledge the support of the ITEA2 Call 3 UsiXML project under reference 20080026; the MITYC under the project MyMobileWeb, TSI-020301-2009-014; the MICINN under the project SESAMO, TIN2007-62894, co-financed with ERDF; the Generalitat Valenciana under the project ORCA, PROMETEO/2009/015, and the grant BFPI/2008/209. Jean Vanderdonckt also thanks the FP7 Serenoa project supported by the European Commission.

References

1. Silvia Abrahão, Emilio Iborra, and Jean Vanderdonckt. Usability Evaluation of User Interfaces Generated with a Model-Driven Architecture Tool. In E. Law, E. Hvannberg, and G. Cockton, editors, *Maturing Usability: Quality in Software, Interaction and Value*, volume 10 of *HCI Series*, pages 3–32. Springer, London, 2008.
2. Nathalie Aquino, Jean Vanderdonckt, and Oscar Pastor. Transformation Templates: Adding Flexibility to Model-Driven Engineering of User Interfaces. In Sung Y. Shin, Sascha Os-

sowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, *Proc. of the 25th ACM Symposium on Applied Computing, SAC 2010 (Sierre, Switzerland, March 22-26, 2010)*, pages 1195–1202. ACM Press, New York, 2010.

3. Nathalie Aquino, Jean Vanderdonckt, Francisco Valverde, and Oscar Pastor. Using Profiles to Support Model Transformations in the Model-Driven Development of User Interfaces. In V. López Jaquero, F. Montero Simarro, J.P. Molina Masso, and J. Vanderdonckt, editors, *Computer-Aided Design of User Interfaces VI, Proc. of 7th Int. Conf. on Computer-Aided Design of User Interfaces, CADUI 2008, (Albacete, Spain, June 11-13, 2008)*, pages 35–46. Springer, Berlin, 2009.
4. Mickaël Baron and Patrick Girard. SUIDT: A task model based GUI-Builder. In Costin Pribeanu and Jean Vanderdonckt, editors, *Task Models and Diagrams for User Interface Design: Proc. of the First Int. Workshop on Task Models and Diagrams for User Interface Design, TAMODIA 2002 (18-19 July 2002, Bucharest, Romania)*, pages 64–71. INFOREC Publishing House Bucharest, 2002.
5. François Bodart, Anne-Marie Hennebert, Isabelle Provot, Jean-Marie Leheureux, and Jean Vanderdonckt. A Model-Based Approach to Presentation: A Continuum from Task Analysis to Prototype. In Paternò [28], pages 77–94.
6. Bert Bos, Tantek Çelik, Hakon Wium Lie, and Ian Hickson. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Technical report, World Wide Web Consortium (W3C), July 2007.
7. L. Bouillon, Q. Limbourg, Jean Vanderdonckt, and B. Michotte. Reverse Engineering of Web Pages Based on Derivations and Transformations. In *Proc. of 3rd Latin American Web Congress LA-Web 2005 (Buenos Aires, Argentina, October 31 - November 2, 2005)*, pages 3–13, Los Alamitos, CA, USA, 2005. IEEE Computer Society Press.
8. Laurent Bouillon, Jean Vanderdonckt, and Kwok Chieu Chow. Flexible Re-engineering of Web Sites. In *Proc. of 8th ACM Int. Conf. on Intelligent User Interfaces, IUI 2004 (Funchal, 13-16 January 2004)*, pages 132–139. ACM Press, New York, 2004.
9. Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. A Unifying Reference Framework for Multi-Target User Interfaces. *Interacting with Computers*, 15(3):289–308, June 2003.
10. Peter P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
11. Adrien Coyette and Jean Vanderdonckt. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In Maria Francesca Costabile and Fabio Paternò, editors, *Proc. of 10th IFIP TC 13 Int. Conf. on Human-Computer Interaction, INTERACT 2005 (Rome, 12-16 September 2005)*, volume 3585 of *Lecture Notes in Computer Science*, pages 550–564. Springer-Verlag, Berlin, 2005.
12. Paulo Pinheiro da Silva and Norman W. Paton. User Interface Modeling in UMLi. *IEEE Software*, 20(4):62–69, 2003.
13. James D. Foley and Piyawadee Noi Sukaviriya. History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-based System for User Interface Design and Implementation. In Paternò [28], pages 3–14.
14. Wilbert O. Galitz. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
15. Peter Johnson, Stephanie Wilson, Panos Markopoulos, and James Pycock. ADEPT: Advanced Design Environment for Prototyping with Task Models. In Stacey Ashlund, Kevin Mullet, Austin Henderson, Erik Hollnagel, and Ted N. White, editors, *Human-Computer Interaction, Proc. of INTERACT '93, IFIP TC13 International Conference on Human-Computer Interaction, 24-29 April 1993, Amsterdam, The Netherlands, jointly organised with ACM Conference on Human Aspects in Computing Systems CHI'93*, page 56. ACM, 1993.
16. Quentin Limbourg and Jean Vanderdonckt. USIXML: A User Interface Description Language Supporting Multiple Levels of Independence. In Maristella Matera and Sara Comai, editors, *Engineering Advanced Web Applications: Proc. of Workshops in connection with the 4th Int. Conf. on Web Engineering, ICWE 2004 (Munich, Germany, 28-30 July, 2004)*, pages 325–338. Rinton Press, 2004.

17. Quentin Limbourg and Jean Vanderdonckt. Multi-Path Transformational Development of User Interfaces with Graph Transformations. In A. Seffah, Jean Vanderdonckt, and M. Desmarais, editors, *Human-Centered Software Engineering*, HCI Series, pages 109–140. Springer, London, 2009.
18. Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, and Víctor López-Jaquero. USIXML: A Language Supporting Multi-path Development of User Interfaces. In Rémi Bastide, Philippe A. Palanque, and Jörg Roth, editors, *Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS 2004 (Hamburg, July 11-13, 2004)*, volume 3425 of *Lecture Notes in Computer Science*, pages 200–220. Springer-Verlag, Berlin, 2005.
19. Benjamin Michotte and Jean Vanderdonckt. GrafiXML, a Multi-target User Interface Builder Based on UsiXML. In D. Greenwood, M. Grottke, H. Lutfiyya, and M. Popescu, editors, *Proc. of 4th Int. Conf. on Autonomic and Autonomous Systems, ICAS 2008 (Gosier, 16-21 March 2008)*, pages 15–22, Los Alamitos, 2008. IEEE Computer Society Press.
20. Pedro J. Molina, Santiago Meliá, and Oscar Pastor. Just-UI : A User Interface Specification Model. In Christophe Kolski and Jean Vanderdonckt, editors, *Computer-Aided Design of User Interfaces III, Proc. of the 4th Int. Conf. on Computer-Aided Design of User Interfaces, CADUI 2002, (Valenciennes, France, May 15-17, 2002)*, pages 63–74. Kluwer, 2002.
21. Tony Morgan. Doing IT Better. Keynote address at the 3rd Conf. on Information Systems Technology and its Applications, ISTA 2004 (Salt Lake City, UT, USA, 15-17 July, 2004), July 2004.
22. Giulio Mori, Fabio Paternò, and Carmen Santoro. Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions. *IEEE Trans. Software Eng.*, 30(8):507–520, 2004.
23. Nuno Jardim Nunes and João Falcão e Cunha. Wisdom: A Software Engineering Method for Small Software Development Companies. *IEEE Software*, 17(5), 2000.
24. Antoni Olivé. Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In Pastor and e Cunha [26], pages 1–15.
25. Oscar Pastor. From Extreme Programming to Extreme Non-programming: Is It the Right Time for Model Transformation Technologies? In Stéphane Bressan, Josef Küng, and Roland Wagner, editors, *Proc. of 17th Int. Conf. on Database and Expert Systems Applications, DEXA 2006 (Krakow, Poland, 4-8 September 2006)*, volume 4080 of *Lecture Notes in Computer Science*, pages 64–72. Springer, 2006.
26. Oscar Pastor and João Falcão e Cunha, editors. *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*, volume 3520 of *Lecture Notes in Computer Science*. Springer, 2005.
27. Oscar Pastor and Juan Carlos Molina. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.
28. Fabio Paternò, editor. *Design, Specification and Verification of Interactive Systems'94, Proceedings of the First International Eurographics Workshop, June 8-10, 1994, Bocca di Magra, Italy*. Springer, 1994.
29. Fabio Paternò. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK, 1999.
30. Fabio Paternò, Carmen Santoro, and Lucio Davide Spano. MARIA: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4), 2009.
31. Inés Pederiva, Jean Vanderdonckt, Sergio España, José Ignacio Panach, and Oscar Pastor. The Beautification Process in Model-Driven Engineering of User Interfaces. In Maria Cecilia Calani Baranauskas, Philippe A. Palanque, Julio Abascal, and Simone Diniz Junqueira Barbosa, editors, *Proc. of 11th IFIP TC 13th Int. Conf. on Human-Computer Interaction, INTERACT 2007 (Ro de Janeiro, September 10-14, 2007)*, volume 4662 of *Lecture Notes in Computer Science*, pages 411–425. Springer-Verlag, Berlin, 2007.

32. Angel R. Puerta and Jacob Eisenstein. Towards a General Computational Framework for Model-Based Interface Development Systems. *Knowl.-Based Syst.*, 12(8):433–442, 1999.
33. Adrian Stanciulescu, Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, and Francisco Montero. A Transformational Approach for Multimodal Web User Interfaces based on UsiXML. In Gianni Lazzari, Fabio Pianesi, James L. Crowley, Kenji Mase, and Sharon L. Oviatt, editors, *Proc. of the 7th Int. Conf. on Multimodal Interfaces, ICMI 2005 (Trento, Italy, October 4-6, 2005)*, pages 259–266, New York, 2005. ACM Press.
34. Wayne P. Stevens, Glenford J. Myers, and Larry L. Constantine. Structured Design. *IBM Systems Journal*, 13(2):115–139, 1974.
35. Pedro A. Szekely. Template-based mapping of application data interactive displays. In Scott E. Hudson, editor, *Proc. of the 3rd Annual ACM Symposium on User Interface Software and Technology, UIST 1990 (Snowbird, Utah, USA, October 3-5, 1990)*, pages 1–9. ACM, 1990.
36. Pedro A. Szekely. Retrospective and Challenges for Model-Based Interface Development. In François Bodart and Jean Vanderdonckt, editors, *Design, Specification and Verification of Interactive Systems'96, Proc. of the 3rd Int. Eurographics Workshop, June 5-7, 1996, Namur, Belgium*, pages 1–27. Springer, 1996.
37. Ismael Torres, Oscar Pastor, Quentin Limbourg, and Jean Vanderdonckt. Una experiencia prctica de generacin de interfaces de usuario a partir de esquemas conceptuales. In Angel R. Puerta and Miguel Gea, editors, *Proc. of VI Congreso Interaccin Persona Ordenador, Interaccin 2005 - CEDI 2005 (Granada, Spain, 13-16 September 2005)*, pages 401–404, Madrid, 2005. Thomson Paraninfo.
38. Francisco Valverde, José Ignacio Panach, Nathalie Aquino, and Oscar Pastor. *New Trends on Human-Computer Interaction. Research, Development, New Tools and Methods*, chapter Dealing with Abstract Interaction Modelling in an MDE Development Process: a Pattern-Based Approach, pages 119–128. Springer London, April 2009.
39. Francisco Valverde, José Ignacio Panach, and Oscar Pastor. An Abstract Interaction Model for a MDA Software Production Method. In John C. Grundy, Sven Hartmann, Alberto H. F. Laender, Leszek A. Maciaszek, and John F. Roddick, editors, *Challenges in Conceptual Modelling. Proc. of tutorials, posters, panels and industrial contributions at the 26th Int. Conf. on Conceptual Modeling, ER 2007 (Auckland, New Zealand, November 5-9, 2007)*, volume 83 of *CRPIT*, pages 109–114. Australian Computer Society, 2007.
40. Jean Vanderdonckt. A MDA-Compliant Environment for Developing User Interfaces of Information Systems. In Pastor and e Cunha [26], pages 16–31.
41. Jean Vanderdonckt. Model-Driven Engineering of User Interfaces: Promises, Successes, and Failures. In S. Buraga and I. Juvina, editors, *Proc. of 5th Annual Romanian Conf. on Human-Computer Interaction ROCHI'2008, (Iasi, 18-19 September 2008)*, pages 1–10. Matrix ROM, Bucarest, 2008.
42. Gerrit C. Van Der Veer, Bert F. Lenting, and Bas A. J. Bergevoet. GTA: Groupware Task Analysis - Modeling Complexity. *Acta Psychologica*, 91:297–322, 1996.