Chapter 1

INTEGRATING MODEL-BASED AND TASK-BASED APPROACHES TO USER INTERFACE GENERATION

Sergio España, Inés Pederiva, Jose Ignacio Panach
Department of Information Systems and Computation
Valencia University of Technology, Camino de Vera s/n, 46071 Valencia, Spain

Abstract: Software Engineering community has been interested in defining methods and processes to develop software by specifying its data and behaviour, but disregarding user interaction. Human-Computer Interaction community has defined techniques oriented to the modelling of the interaction between the user and the system, proposing user-oriented software constructions. In this paper, we show how to lay proper bridges between both visions, by integrating a CTT task model into a sound, model-based software development process. This proposal is underpinned by the MDA-based technology OlivaNova Method Execution, which makes software generation a reality, while still taking the user interaction needs into account.

Key words: Functional Requirements, User Interaction, Model-based, Task-based, User Interface, Code Generation.

1. INTRODUCTION

Software Engineering (SE) is considered to be strong in specifying functional requirements, while Human-Computer Interaction (HCI) is centred on defining user interaction at the appropriate level of abstraction. In either case, software production methods that combine the most functional-oriented, conventional requirements specification with the most interaction-oriented, user interface modelling are strongly required.

From an HCI point of view, there is a number of model-based user interface development environments (MB-UIDEs) reported in the literature

[2]. The first generation aimed to provide run-time environment, as in COUSIN [4] and HUMANOID [14]. The second generation increased the abstraction level, as in MASTERMIND [15]. More recently, some broader frameworks have been proposed, like USIXML [16] and TERESA [10]. SE has proposed UML-based approaches, for example WISDOM [11] and UMLi [3].

Model transformation technologies (i.e. MDA [7]) make it possible to provide a global software process where the requirements model includes all the relevant aspects of the analysed problem. These are first projected onto a Conceptual Model and onto the final software product later.

The intended contribution of this paper is to extend a sound software production process with an interaction requirements elicitation. Two basic principles remain constant in the paper:

- Model Transformation is used to automate the conversion of the Requirements Model into the Conceptual Model and then convert this Conceptual Model into the final software application.
- Each modelling step provides appropriate methods to deal properly with the specification of structural, functional and interaction properties.

The approach presented here has been successfully implemented in OlivaNova Model Execution (ONME), an MDA-based tool that generates a software product that corresponds to the source Conceptual Model. This tool should later be enhanced to support our requirements level proposal.

The paper is structured as follows. Section 2 introduces a software production process that combines model-based and task-based approaches. This process is explained using a case study. And section 3 presents the conclusions, and future work.

2. MODEL-BASED INTERFACE DEVELOPMENT WITH OLIVANOVA MODEL EXECUTION

In this section, we present a complete software production process that combines functional requirements specification, user interaction design, and implementation. It is defined on the basis of OlivaNova Model Execution (ONME) [1], a model-based environment for software development that complies with the MDA paradigm [7] by defining models of a different abstraction level. Figure 1-1 shows the parallelism between the models proposed by MDA and the models dealt with in OO-Method [12] (the methodology underlying ONME).

At the most abstract level, a Computation-Independent Model (CIM) describes the information system without considering if it will be supported by any software application; in OO-Method, this description is called the

Requirements Model. The Platform-Independent Model (PIM) describes the system in an abstract way, still disregarding the underpinning computer platform; this is called the *Conceptual Model* in OO-Method. ONME implements an automatic transformation of the Conceptual Model into the *source code* of the final user application. This is done by a *Model Compilation* process, with knowledge about the target platform. This step is equivalent to the Platform Specific Model (PSM) defined by MDA.

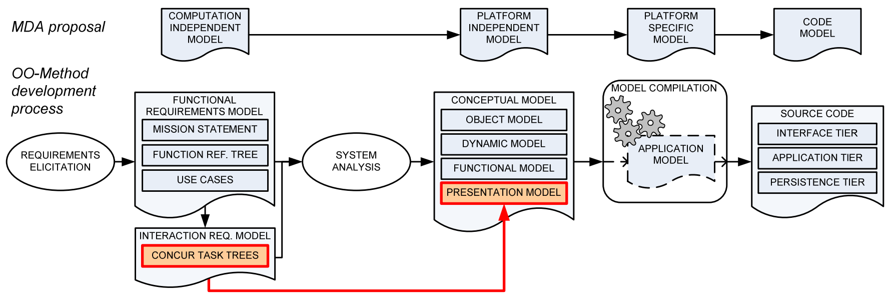


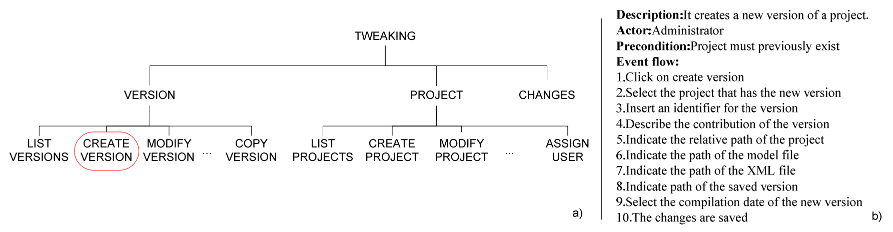
Figure 1-1. A MDA-based development framework for UI development

2.1 Obtaining Functional Requirements

The first step in the construction of a software system is the requirements elicitation. Its purpose is to specify what the customer needs. In our process, this step is accomplished through the definition of a *Functional Requirements Model* [5]. The requirements model is composed by: a mission statement, a function refinement tree, and a use-case model.

The *Mission Statement* describes the purpose of the system in one or two sentences. The external interactions are partitioned into functions, which are hierarchically structured in a *Functions Refinement Tree (FRT)* where the root is the mission statement, the internal nodes are business activities, and the leaves are use cases. Finally, the *Use Case Model* includes the interaction (decomposed in steps) between the system and an external actor.

In order to explain our proposal, we have chosen an application generated using ONME. The system to be built is *OlivaNova Automatic Tweaking*, part of the ONME suite. It is intended to automate subtle manual changes requested by clients after the code generation. We have simplified the case study, selecting one task of this system: ***Create a version***. In this task, the user must select an existing project and add a new version for it. In Figure 1-2 (a), we show part of the FRT in which this task is included.



Description: It creates a new version of a project.
Actor: Administrator
Precondition: Project must previously exist
Event flow:
1.Click on create version
2.Select the project that has the new version
3.Insert an identifier for the version
4.Describe the contribution of the version
5.Indicate the relative path of the project
6.Indicate the path of the model file
7.Indicate the path of the XML file
8.Indicate path of the saved version
9.Select the compilation date of the new version
10.The changes are saved

Figure 1-2. Functional requirements for the Tweaking system

Once we have the FRT, the Use-Case Model can be obtained from the leaves of the tree. Our case study will be centered on the *Create version* use case that corresponds to the leaf that is marked in the FRT. Figure 1-2 (b) presents the specification of this use case. It consists of a use case description, the actors who can invoke it, the conditions needed to execute it and the list of events which compose it.

2.2 Eliciting User Interaction

In order to document *interaction requirements*, we propose the use of the *Concur Task Trees* (CTT) notation [13]. The interaction between the user and the system is specified by means of a task model, resulting in a hierarchical task tree in which the tasks have different granularity and are related by temporal operators.

A CTT tree is built for each use case. It specifies how the user interacts with the system in order to accomplish the task required. The use case constitutes a high-level task that is decomposed into lower granularity tasks; it is important to define the criterion of decomposition: we propose to reach basic tasks concerning data elements. There is a deep mapping between elements of the use-case specification and elements of its corresponding task model: the steps of the use case involving elemental data manipulation appear as basic tasks of the task tree.

Sometimes this interaction modelling process involves several use cases, which results in their restructuring; that is, the interaction requirements reorganize the functional requirements. These interaction requirements complete the CIM level of our development approach. We have built the task trees with certain structures of tasks that we have observed to be frequent.

Following with the example, Figure 1-3 shows the CTT model for the *Create version* use case.

The root of the task tree is the use case whose interaction is being modelled. Since this use case has the selection of a project as a precondition, we reuse the *List versions from project* CTT, and we include the *Demand the creation of a new version* interactive task.

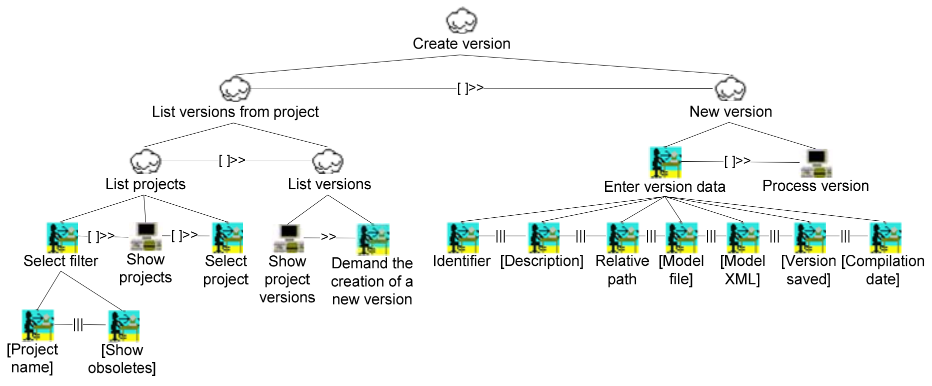


Figure 1-3. CTT model for the *Create version* use case

We can now identify the mapping between the steps of the use-case specification and the lowest level tasks; i.e., step 5 “Indicate the relative path of the project” has its correspondent interaction task: “Relative path”. It is also noticeable that several use cases have been reorganized: the main functionality of *Create version* (the *New version* abstract task) is accessed through the interface of the *List projects* and *List versions* use cases.

2.3 Modelling Data, Behaviour, and Interaction

After the Functional Requirements Model and the CTTs are specified, the data, behaviour, and presentation issues should be modeled. OO-Method defines a PIM called *Conceptual Model* [12] consisting of four models. The *Object Model*, the *Dynamic Model*, and the *Functional Model* can be built using the Functional Requirements Model as input.

Interaction between the system and the user is specified in the *Presentation Model* [8]. It is based on a pattern language which defines three levels of interaction patterns: (1) **Hierarchy of Actions Tree (HAT)**: it organizes the functionality that will be presented to the different users who access the system; (2) **Interaction Units (IUs)**: it represents abstract interface units that the user will interact with to carry out his/her tasks. There are four types of UIs: Service IU, Instance IU, Population IU, and Master/Detail IU; (3) **Elementary patterns (EPs)**: these patterns constitute the primitive building blocks of the UI and allow the restriction of the behaviour of the different interaction units.

As suggested in Figure 1-1, there is a direct mapping between parts of the CTT and parts of the Presentation Model. To do that, we have to define a design pattern for CTT and concrete names for their components. We show some of them in Table 1-1.

Table 1-1. Some matching between CTT and Presentation Model

CTT	Presentation Model
Two abstract tasks related by an *Enabling with information passing* temporal operator	Master / Detail
An abstract task whose name starts with *New* or *Modify*	Service IU
An abstract task whose name starts with *List*	Population IU
An abstract task whose name starts with *Detail*	Instance IU
An interaction task named *Select filter*	Filter
An interaction task named *Select sort criteria*	Order criteria
An interaction task, leaf of a subtree, in a Service IU	Introduction
An interaction task whose name starts with *Demand*	Action

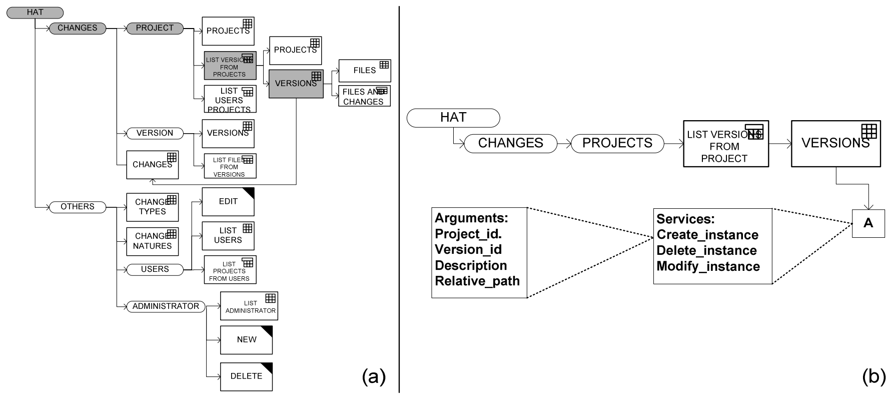


Figure 1-4. Presentation Model for the Tweaking system

Figure 1-4 (a) shows the system’s first two levels of presentation patterns according to the case study. The patterns that appear highlighted in gray are detailed in (b). Figure 1-4 (b), shows the services and the arguments of *Create_Instance*.

To design this model, we consider the CTT generated in the previous step. In particular, Figure 1-4 (b) was generated by the definition provided in Figure 1-3. The left subtree of the CTT is mapped to a Master/Detail pattern. And, on the other hand, the right side of the tree model is equivalent to a Service IU; its execution will call the *Create_Instance* service. As this service corresponds to a service defined in the Object Model, it turns towards an early model validation, based on the traceability among the Presentation Model and the Use Cases.

2.4 Generating the System

Once we have completed the PIM, the next step is to take advantage of ONME automatic production of the *source code* (Code Model). Nowadays, ONME implements a *Model Compilation* process to automatically transform the information captured in the *Conceptual Model* into a full software system over the following platforms: Visual Basic, C #, ASP. NET, Cold Fusion and Java; using as repository SQL server 2000, ORACLE or DB2. The resulting application is a three-layer application that includes the interface tier, the application tier, and the persistence tier. Some correspondences between PIM elements and their final, concrete widgets for the Visual Basic platform are defined in [8].

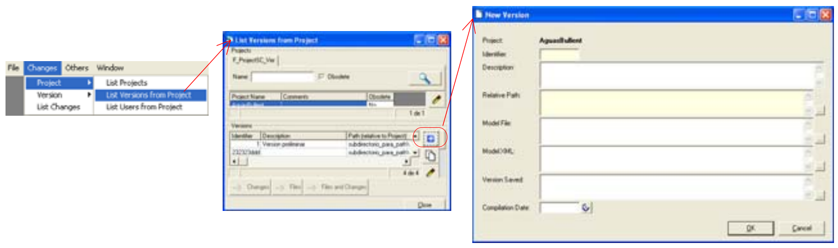


Figure 1-5. New version window

The result of applying the translation patterns [8] is shown in Figure 1-5. The HAT has become the application menu and the *List Version from Project* and *Versions* patterns have been turned into windows. In the *List Versions from Project* window, we can invoke the tasks related to the versions listed by a project. It contains the task to create a version (marked button). Once the button is clicked, the *New Version* window appears to allow the introduction of the information to create a version.

3. SUMMARY AND FUTURE WORK

Software production methods need a complete software production process that properly integrates system functionality, behaviour, and user interaction in the early stages of the system lifecycle. This process should also allow the sketching, modelling, and prototyping of UIs. In accordance with these ideas, we have presented a software production process that starts from requirements elicitation and uses CTT notation based on tasks to build a full software system, not just its user interface. To do this, we embedded the CTT notation in a model-based development approach by respecting its original semantics.

The proposed process will be empirically validated in the near future to prove its effectiveness. As a future work, an application that will implement CTT drawing using these proposed patterns will be integrated into the ONME suite. This will allow CTT nodes to be reused easily.

REFERENCES

[1] Care Technologies: http://www.care-t.com Last visited: Mar-2006.
[2] da Silva, P. P. (2001) ”User interface declarative models and development environments: A survey”. Interactive Systems. Design, Specification, and Verification, 8th International Workshop, DSV-IS 2001, Glasgow, Scotland, Springer-Verlag Berlin.
[3] da Silva, P. P. d. and N. W. Paton (2003). "User Interface Modeling in UMLi " IEEE Softw. 20 (4); pp. 62-69.
[4] Hayes, P., Szekely, P. and Lerner, R. (1985) Design Alternatives for User Interface Management Systems Based on Experience with COUSIN. Proceedings of SIGCHI'85, pp. 169-175. Addison-Wesley.
[5] Insfrán, E., Pastor, O., Wieringa, R. (2002). Requirements Engineering-Based Conceptual Modelling. Requirements Engineering, Vol. 7, Issue 2, p. 61-72. Springer-Verlag.
[6] Limbourg, Q. and J. Vanderdonckt (2004). Addressing the mapping problem in user interface design with UsiXML Proceedings of the 3rd annual conference on Task models and diagrams Prague, Czech Republic ACM Press; pp. 155-163.
[7] MDA: http://www.omg.org/mda Last visited: Jan-2006.
[8] Molina P., User interface specification: from requirements to automatic generation, PhD Thesis, DSIC, Universidad Politécnica de Valencia, March 2003 (in Spanish).
[9] Montero, F., V. López-Jaquero, et al. (2005). Solving the mapping problem in user interface design by seamless integration in IdealXML. Proc. of DSV-IS'05, Newcastle upon Tyne, United Kingdom, Springer-Verlag.
[10] Mori G., Paternò F., Santoro C. (2004) "Design and Development of Multidevice User In-terfaces through Multiple LogicalDescriptions" IEEE Transactions on Software Engineer-ing; pp.507-520.
[11] Nunes, N. J. y J. F. e. Cunha (2000). "Wisdom: a software engineering method for small software development companies." Software, IEEE 17(5); pp. 113-119.
[12] Pastor, O., J. Gómez, et al. (2001). "The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming." Information Systems 26(7): 507-534.
[13] Paternò, F., C. Mancini, et al. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, Chapman & Hall, Ltd.; pp. 362-369.
[14] Szekely, P. (1990) Template-Based Mapping of Application Data to Interactive Displays. Proceedings of UIST'90, pp. 1-9. ACM Press.
[15] Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., and Salcher, E. (1996) Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach. In Engineering for HCI, pp. 120-150, London, UK, Chapman & Hall.
[16] Vanderdonckt, J., Q. Limbourg, et al. (2004). USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, Greece.