Linking requirements specification with interaction design and implementation

Sergio España, Inés Pederiva, José Ignacio Panach, Silvia Abrahão, Óscar Pastor

Department of Information Systems and Computation
Valencia University of Technology
Camino de Vera s/n, 46071 Valencia, España
{sergio.espana,ipederiva, jpanach,sabrahao,opastor}@dsic.upv.es
Phone: +34 96 387 7000, Fax: +34 96 3877359

Abstract: One challenging goal in the context of Software Engineering (SE) and Human-Computer Interaction (HCI) is to provide appropriate bridges between the most well-known software production methods and techniques. SE is supposed to be strong in specifying functional requirements, while HCI is centred on defining user interaction at the appropriate level of abstraction. In any case, general-perspective software production methods that combine most functional-oriented, conventional requirements specification with the most interaction-oriented, user interface modelling are strongly required. In this paper, we present a specific approach in this context, intended to properly combine a sound functional requirements specification with an abstract model of the user interface represented by a CTT model. When the functional specification is enriched with such an interaction model, it is easier to derive the final software implementation that will represent both the structure and behaviour of the system and the user interaction. The presented approach has been successfully implemented in a MDA-based approach called Oliva Nova Model Execution, demonstrating that Conceptual Modeling-based strategies are more powerful when user interaction and system behaviour are modelled within a unified view.

1. Introduction

Conventional software production methods focus on a precise specification of system structure and behaviour. Normally, a class diagram-like model is used to model the system structure, and a process model fixes the functionality that the system is supposed to provide. Many CASE tools [11] [13] [14] have been proposed during the last few years with the objective of providing some kind of automation to manage these data and process models in the context of a well-defined software process.

Generally speaking, when defining such a software process, there is a Requirements Modelling step where requirements elicitation and specification are the issues which are normally based on some kind of use case-based strategy where functional requirements are dealt with. These functional requirements are the source model for creating a Conceptual Schema, where the requirements are reified in the corresponding set of classes and relationships between classes. Finally, it is common to have a of

model transformation approach to guide the final step, where the software product that properly represents the initial requirements is provided.

However, from a HCI point of view, these conventional Software Engineering approaches for software production have a mayor problem: to determine how to correctly embed user interaction modelling in such a software production process. It is curious that interaction modelling is not a key issue when requirements and conceptual modelling is faced in a software production process. Together with data and process models, it is very rare to have an Interaction Model playing a basic role at the same abstraction level. It is clear that system structure and behaviour are considered to be basic parts of such a description.

The way in which users interact with the system should be part of the system model in order to be a basic part of the world description and to model the way in which user interaction is going to make feasible the putting into practice the static and dynamic system parts (class architecture and functional requirements respectively) To do this, we claim that functional requirements specification and user interaction design and implementation must be linked in a precise way. Precise means with a formal background, providing a clear set of concepts and a notation to specify user interaction from the very beginning of a software production process; this includes linking the different models involved at the different levels of abstraction starting from functional requirements to a final software product, through the corresponding conceptual schema.

To do this, in this paper, we present a specific software production process, where

1. The first step is a conventional requirements modelling approach, that is based on use-cases, but that has a precise proposal for structuring the relevant functional requirements.
2. When the functional requirements are specified, the corresponding user interaction model associated to the requirements model is specified. To do this with a set of precise concepts and notation and with the required formal support, we use the ConcurTaskTrees (CTT) model [10]. Tasks from the Requirements Model are enriched with the specification of their associated interactions, and a well-defined and contextualized use of the CTT model is proposed to avoid the lack of methodological rules that the use of CTT often suffers from in practice.
3. When both the functional requirements and their associated interaction model have been specified, a Conceptual Schema can be obtained. As an interaction model has a precise semantics provided by the CTT, it is possible to design and implement not only classes and services, but also user interfaces, providing a full software production process from requirements to the final software product where the user interface corresponding to the modelled interaction is properly incorporated.

It is important to note that the Interaction Model that we introduce with the use of CTT complements the functional system specification with the kind of logical interaction design description that the CTT model provides. The functional requirements specification provides the necessary analysis of human life and work context and takes the source Information System as input. The CTT model associates a user interaction to each system task. This will allow the model to execute correctly from the user interaction point of view at the solution space. Within a well-defined software production process, these Interaction Models can be converted into the corresponding

sets of user interfaces of the final software product. In this way, system structure, behaviour, and interaction properties are properly dealt with.

Furthermore, this strategy fulfils the current MDA-based approaches that defend the metaphor of model transformation and its automation as the key for defining modern and efficient software processes [4]. In fact, the ideas presented in this paper are already being successfully applied to Oliva Nova Model Execution [1]. This is a model-based code generation tool where our interaction model has been implemented to put into practice all the ideas discussed here.

This paper has the following structure. After the introduction, Section 2 explains how to model functional requirements. To determine how to model user interaction to enrich the Requirements Model, Section 3 describes how the Requirements Model is enriched to incorporate user interaction specification using the CTT model in a precise context for very specific objectives. Once both functional requirements and user interaction are properly specified, Section 4 explains show how to go to the solution space by generating the user interface that implements the user interactions modelled in the previous section. We present an example taken from Oliva Nova Model Execution which is the software generation tat we have used to implement our ideas.

2. Obtaining functional requirements

In the SE field, the first step in building a software system is to capture the functional properties that the system requires. In our software production process, this is done through the definition of a Requirements Model [2] [3]. This model contains a description of the objectives and external behaviour of the system, that is, *what* the system must do without describing *how* to do it.

The Requirements Model incorporates a set of complementary techniques: the mission statement, the functions refinement tree, and the Use-Case Model.

The *Mission Statement* is a high-level description of the nature and purpose of the system. Through this definition, it is possible to accurately determine what the system will and will not do.

The *Functions Refinement Tree (FRT)* represents the hierarchical decomposition of the business functions of a system independently of the current system structure. The resultant tree is merely an organization of external functions and does not say anything about the internal decomposition of the system. The leaves of this tree represent the functions of the desired system, which are the use cases. This gives the entry point for building the Use-Case Model instead of starting from scratch, and it avoids the potential problem of mixing the abstraction levels of use cases.

Once we have defined the *FRT*, the next step is to create the *Use-Case Model*. A use case is an interaction between the system and an external entity. This interaction can usually be decomposed into an activity set (Case Use specification) defined at this level as atomic functions. The leaf nodes of the FRT (elemental functions) are considered to be primary use cases; they represent the most important functions of the system. It is also possible to have secondary use cases. These use cases are scenarios that have no direct correspondence to the FRT; however, they are important for organizing

and managing complexity through relationships among use cases that are stereotyped as EXTEND and INCLUDE.

In order to explain our proposal, we use an example taken from a real system from the Oliva Nova Model Execution portfolio: the Bullent's Water application. This system is used in a company that delivers water to homes. The main functions of the system are: read client's meter, emit an invoice, register the use of some material in a repair, and maintain the stock in the warehouses. For the sake of simplicity, we have centred our attention only in one task of this system: the task to *create a new subscriber* in the company. This task is composed by several subtasks: create a client, create a transfer, create a meter, create a new meter address, create a destination and verify the new subscriber data. Figure 1 shows the functions refinement tree for the task being studied. It contains all the functions needed to create a new subscriber.

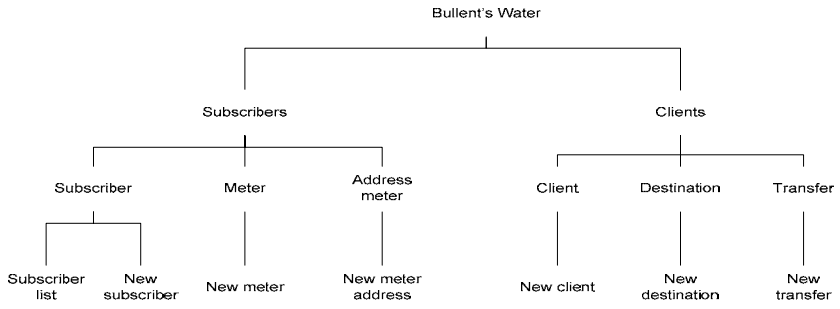
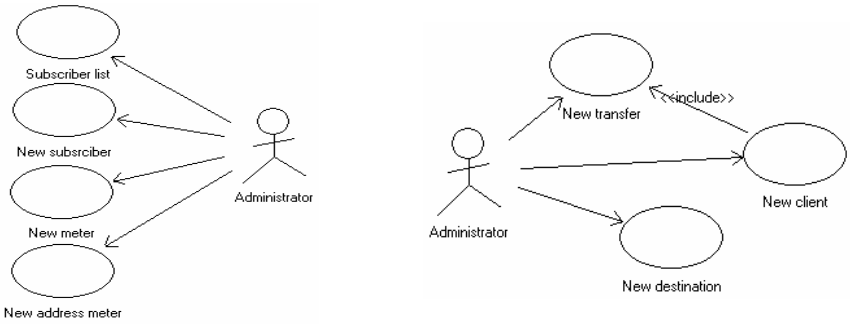


Fig. 1. Function refinement tree

The next step is the definition of a Use-Case Model. To draw this diagram, we have used RETO[1] (Requirements Engineering TOol).



Subscriber functional group (a) Clients functional group (b)

Fig. 2. Use Case Diagrams

These diagrams show all the functions that make up the task "create a new subscriber". There is an *include* relationship between the "New subscriber" use case and

[1] http://reto.dsic.upv.es/reto/

all the other use cases, with the exception of the "New transfer" use case. These lines have not been drawn to simplify the diagram reading.

Finally, all the steps of the use-case behaviour must be specified. As an example, the "New subscriber" use case is detailed in the following template:

- **Description:** It creates a subscriber in the company. It records information about the person, his/her meter, his/her destination and his/her transfer.
- **Steps:**

1. Create a new client INCLUDE USE CASE: New client
   1.1 (Inside New client) Create a transfer for the new client INCLUDE USE CASE: New transfer
2. Create a meter INCLUDE USE CASE: New meter
3. Create a meter address INCLUDE USE CASE: New meter address
4. Select a zone
5. Select a category
6. Select a house type
7. Introduce dispenser diameter
8. Introduce branch number
9. Select discharge date
10. Create a destination INCLUDE USE CASE: New destination
11. Introduce connection length
12. Introduce connection diameter
13. Select invoice option
14. Select invoice type
15. Introduce concept
16. Select currency
17. Save the new subscriber

As a consequence, all system functional requirements are captured in the Requirements Model. The leaves of the functions refinement tree are tasks functionally described using the use case templates. However, the next step is to properly capture the user interaction.

3. Modelling interaction with CTT task model

After using the RETO tool to elicit and document system requirements, we have a complete decomposition of the system behaviour. As the leaves of the FRT are well-defined use cases, the functional requirements of the Information System are properly stated; however, the interaction between the user and the system is still not sufficiently documented. To be able to use this functionality, this section presents a way to deal with interaction requirements. It not only shows how to model an interaction but also to reason about it. To do this, we use the ConcurTaskTree notation (CTT) in the framework of our development process [9] [10].

CTT, as originally proposed by Paternò, constitutes a formal notation to express human-computer interaction as a graphical task decomposition. The main constructors are tasks and relationships between them. Tasks can be of four types (abstract, inter-

action, application and user task) and several relationships are available to link sibling tasks. These basic blocks for building the task model were well defined, but little criteria has been provided for its application.. Thus, as the HCI and SE communities started using this notation, different approaches were used, which has resulted in creating its own criteria for task decomposition and granularity. We define the use of the CTT notation according to our specific context.

One of the first criterion needed is the starting point for the interaction specification: we are not building a single tree that describes all the interactions between the user and the system, but rather a forest of task trees with the root of each tree being a leaf of the FRT, that is, a use case of our functional Requirements Model. Given the correspondence between use cases and the upper abstract task node, we decompose these into a *data introduction* task followed by an *application* task that processes the data entered by the user (see Figure 3). For some complex use cases, the interaction is fragmented into several subtasks which are embedded in a similar structure.

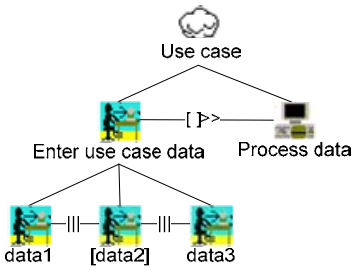


Fig.3. Example of the interaction related to a generic use case

The decomposition is continued until the individual data elements that make up the message that is being communicated to the software system are reached. These data elements are commonly derived from (and consequently mapped to) the fields of business forms; they are documented in the use case description (see example in Section 2) and will later be mapped to data entry interface fields. This way, we can model the edition of the message that the user wants to communicate to the system.

We also include application tasks, which correspond to the processing of the information supplied by the user and the feedback resulting from this process. These tasks are triggered by the user with an implicit end-of-edition signal (e.g. after the last piece of data is entered) or an explicit end-of-edition signal (e.g. the user presses a button). Application tasks are later mapped to services of objects or global transactions; in the latter case, we could decompose the application task into sub-tasks that would model the structure of services involved in the transaction. However in order to keep the task tree as simple as possible, this refinement would be done in a separate tree in a packet-like approach.

Figure 4 shows the CTT for the *Create subscriber* use case that was described in the previous section.

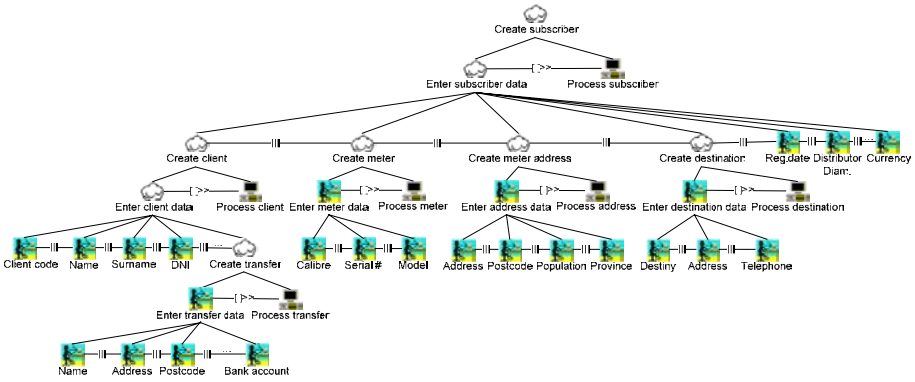


Fig. 4. CTT for the *Create subscriber* use case

Since a use case that includes other use cases a complex one, the resulting CTT has nested abstract tasks that also consist of data entry and process; for example, the need for creating a new meter to be related to the subscriber.

It can be seen how the abstract task in the root maps to a use case of our functional Requirements Model and also how the decomposition stops at the level of individual data elements. For the sake of simplicity, we have limited the number of data elements in Figure 4 by informally using ellipsis points.

4. Generating the user interface

The previous sections define ways to specify the requirements of a software system in terms of its functionality and user interaction. These requirements describe what the system needs to provide and does not describe how to implement the solution. In order to complete the software development process, the software should be implemented. For this phase, there are tools that provide support for designing conceptual models derived from the requirements and that generate the source code of the designed application.

Although there are many popular tools that support this process [11] [13] [14], none of them consider the design of the user interface. Therefore, none of these tools provides automatic derivations for the final user interface. The above mentioned OlivaNova™ Model Execution (ONME) implements the method proposed in OO-Method [8] which provides both the automatic transformation desired and also takes into account the abstract design of user interfaces.

In accordance with the directives of the Model-Driven Architecture (MDA) framework [4], ONME has a direct correspondence with the different models that MDA proposes. OlivaNova is a MDA-based technology that implements model transformations in an industrial context [6]; specifically, it allows the automatic generation of complete applications from the Platform-Independent Model, which describes the

information system at the analysis level and comes right after the Computation-Independent Model.

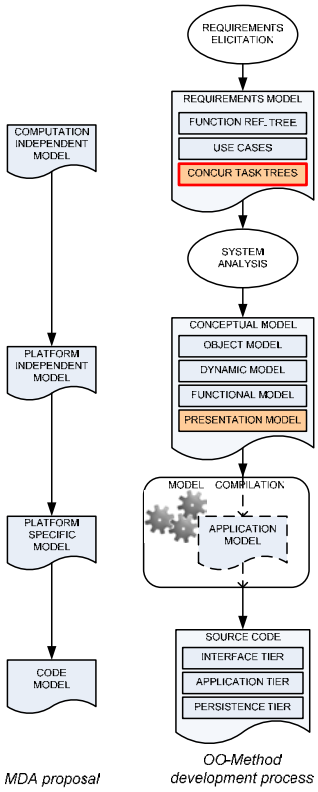


Fig. 5. Usage of ConcurTaskTrees in a MDA-based development framework

In the Platform-Independent Model level, OO-Method includes the *Presentation Model* as a view of the *Conceptual Model*, which is a model for the abstract specification of user interfaces. Based on a pattern language called Just UI [5] [7], this model allows the analyst or the designer to manipulate, configure and link those patterns in order to build the interface abstract model.

With the complete design, the source code of the final application can be obtained with OMNE, resulting in a three-layer application which includes the application logic, the database, and the user interface.

The design and development process can be resolved based on the requirements obtained and the interaction designed, but there is no explicit link between them. Although there is no direct binding, we have found a relationship between the CTT model and the Presentation Model and its final developed interfaces.

As experimentation, we designed the Presentation Model following the design of the CTT task model using the patterns provided by Just UI [5] [7] and we obtained the resulting interfaces. Figure 6 shows the final window resulting from the application of this technique with the CTT previously described in Figure 4.

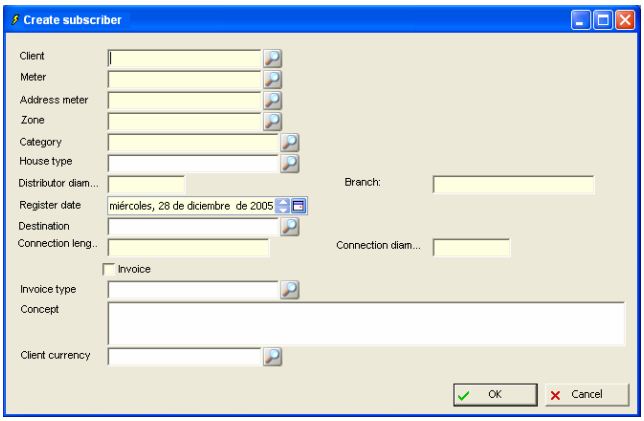


Fig. 6. Generated window

In these experiments, we found that there was a service behind each task. Therefore we defined a window for each task. to the comments of each window corresponded to the definition of the task tree and the following rules were applied in the transformation:

1. Each data introduction had a corresponding widget, corresponding to the data type that the system required.
2. Each *application* task had its corresponding Ok button in order to apply the changes
3. Each *abstract* task called another window to fulfil the entry (obviously, the window called was generated by the decomposition of the abstract task).

Figure 7, presents a graphical explanation of these correspondences.

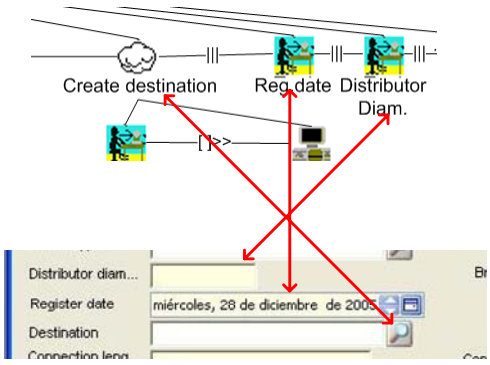


Fig. 7. Correspondences between CTT and the final widgets

The developed application is currently in production phase in the company. The final users have indicated that they are happy with the user interface. The results of this work suggest that we can continue working in this direction in order to link the speci-

fication of the CTT task model and the Presentation Model. The mapping of CTT primitives and the conceptual patterns proposed by Just UI could open up a new avenue for automatically generating user interfaces based on the CTT task model with a guaranteed valid semantic granted by CTT task models.

5. Conclusions

5.1 Current research

Current software production methods are characterized by not having the design of user interaction as part of the system lifecycle. As a result, most of the problems in software development are related to user-interface design. Similarly, current UI tools in the HCI field tend to focus on issues such as colours, fonts, and alignment, which are more appropriate in the later design stages. In addition, most of these tools do not support a process that links UI design with system behaviour. What is needed is 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. It is our position that SE and HCI software production methods and techniques can be integrated to provide such an approach.

In accordance with these ideas, in this paper we have presented a software production process that integrates model-based and task-based approaches to user interface design. This process properly combines a functional requirements specification with a hierarchical model of the user interface represented by a CTT model. In fact, the system functionality and behaviour as well as the user interface are modelled at an early stage of the system lifecycle. This is done thanks to the Oliva Nova Model Execution approach which allows the modelling and automatic generation of software applications. This is a pure model-transformation process where conceptual primitives of the Conceptual Model level are converted into their associated software component counterparts, within what we might call a Conceptual Model Compilation Process. This approach can help to bridge the gap between SE and HCI since it addresses the issue of modelling user interaction, which has often been ignored by SE researchers.

5.2 Pending Issues

Although the use of CTT task models in the context of the OlivaNova Model Execution approach seems to be useful, we must still verify this empirically. We are planning to run an empirical study to assess the effectiveness and perception of the analysts in the use of our software production process with and without CTT task models. The goal is to verify the quality of the user interfaces obtained following a well-defined semantic provided by the CTT task models.

In addition; we want to make an in-depth analysis of the use of MDA-based techniques to provide a full software production process. In this process, the unified modelling of system structure, system behaviour, and user interaction should guide the

different model transformations at the different levels of abstractions from requirements to the final software product through the corresponding conceptual schema. The final goal is to have a General Model Compiler as the main software production tool, which will make it easier to guarantee that the final software product is the correct representation of the initial user requirements. All of this should be based on a common, sound, and rigorous engineering approach for dealing with the software production process as a whole, including static, dynamics and interaction.

References

[1] Care Technologies: http://www.care-t.com Last visited: Dic-2005

[2] Insfrán E., Pastor O. and Wieringa R., Requirements Engineering-Based Conceptual Modelling. Requirements Engineering 7 (2): 61-72 (2002).

[3] Insfrán E., Molina P., Martí S., Pelechado V., Requirements Engineering applied to the Conceptual Modeling of User Interfaces, IV Iberoamerican Workshop on Requirements Engineering and Software Environments (IDEAS'2001), Santo Domingo, Heredia, Costa Rica, pp. 181-192, April 2001 (in Spanish)

[4] MDA: http://www.omg.org/mda Last visit: Dec-2005

[5] Molina P., Specification of User Interfaces: from requirements to automatic generation, PhD Thesis, Dept. of Information Systems and Computation. Valencia University of Technology , March 2003 (in Spanish).

[6] Molina, J.C., and O. Pastor. "MDA, OO-Method and the OlivaNova Model Execution technology". I Workshop on model driven development, MDA and applications. Málaga, Spain, 2004.

[7] Molina, P. J. Meliá, S. and Pastor, O. (2002), Just-UI: A User Interface Specification Model. In Ch. Kolski and J. Vanderdonckt (Eds.), Computer-Aided Design of User Interfaces III, Kluwer Academics Publisher, 63–74.

[8] Pastor, O., Gómez, J., Insfrán, 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.

[9] Paternò, F. (2000). Model-Based Design and Evaluation of Interactive Applications, Springer-Verlag, Berlin, Alemania.

[10] 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.: 362-369.

[11] Posiedon http://www.gentleware.com Last visited: Dic-2005

[12] Pribeanu, C. and J. Vanderdonckt (2002). "A methodological approach to task-based design of user interfaces." Studies in Informatics and Control 11(2): 145-158.

[13] Rational http://www-306.ibm.com/software/rational/ Last visited: Dic-2005

[14] Together http://www.borland.com/together Last visited: Dic-2005