*Journal of Universal Computer Science, vol. 14, no. 9 (2008), 1480-1495
submitted: 30/4/07, accepted: 19/2/08, appeared: 1/5/08 © J.UCS*

Capturing Interaction Requirements in a Model Transformation Technology Based on MDA

Jose Ignacio Panach, **Sergio España**, **Inés Pederiva**, **Óscar Pastor**
(Department of Information Systems and Computation,
Technical University of Valencia, Spain
{jpanach, sergio.espana, ipederiva, opastor}@dsic.upv.es)

Abstract: Currently, many models are used to capture functional software requirements. However, the Software Engineering community has faded interaction requirements into the background, dealing with interface mainly in design time. A sound MDA-compliant software development methodology, called OO-Method, is extended in this work to bridge this gap. The issue is to define a methodology for capturing interaction requirements. For this purpose, the formal notation ConcurTaskTrees (CTT) is used. This notation is a technique that is well-known in the Human Computer Interaction community. A set of interaction patterns has been defined to build CTT models. These patterns are defined with a very precise syntax and semantics. Moreover, transformation rules are defined to transform the Task Model into the OO-Method Presentation Model, which specifies the user interface in an abstract and platform-independent way. However, since editing the CTT models is hard work, this paper proposes superimposing a layer to the CTT diagram in order to capture interaction requirements using sketches. CTT models will be synchronously generated from these sketches. Because this transformation is 'transparent' to the analyst, he only needs to draw the sketches during the interaction requirements elicitation. The approach presented in this paper is instantiated for the environment of the OLIVA**NOVA** technology. This environment makes it possible to obtain a final software product from its corresponding Conceptual Model through a Model Compilation process, where interaction modeling is properly embedded with the most conventional data and process modeling.

Keywords: Model transformation, automatic code generation, sketches, interaction requirements, usability, Model Compiler, automatic code generation
Categories: H.5.2, I.6.5

1 Introduction

The Software Engineering (SE) community has historically relegated interaction aspects to the design step. Therefore, there are no extensively known models to capture interaction requirements and to model them in the analysis step. In contrast, the Human-Computer Interaction (HCI) community uses several techniques to capture interaction requirements with the user. This paper proposes a method to bring the two communities closer together, because interaction is currently a critical aspect in developed systems.

The goal of this paper is to include techniques from the HCI community in a MDA [MDA] environment to enrich it with the capture of interaction requirements. The MDA paradigm argues that a system can be viewed at different abstraction levels: A high level of abstraction, which corresponds to the problem space

(Conceptual Model), and a lower level of abstraction, which corresponds to the solution space (system code). Therefore, following the MDA paradigm, interaction requirements can become generated code after several automatic transformations.

This paper is focused on the MDA environment called OO-Method [Pastor 01]. OO-Method includes a Conceptual Model to represent the problem space, which is equivalent to the Platform-Independent Model in MDA. In short, this Conceptual Model is composed by the following model views: the Object Model, which specifies the object structure and its static interactions; the Dynamic Model, which represents the control, the valid sequences of events, and the interaction between the objects; the Functional Model, which specifies how events change the object states; the Presentation Model, which is a model for the abstract specification of user interfaces.

OO-Method has an equivalent to the Platform-Specific Model of MDA called Model Compiler. By applying the Model Compiler, the code that implements the system can be automatically generated from the Conceptual Model, building the softwapre product at the solution space (called Code Model in MDA). The OO-Method methodology is supported by an industrial tool called OLIVANOVA [CARE].

Figure 1 shows the OO-Method methodology from the requirements step to the source code generated using the Model Compiler. Currently, OO-Method has a set of models to capture functional requirements: Mission Statement, Functional Refinement tree, and Use Cases. As [Insfran 02] explains, these models have a mapping with some models of the Conceptual Model: Object Model, Dynamic Model, and Functional Model. Therefore, parts of these models can be obtained by transformation rules from the capture requirements step. However, OO-Method lacks a model to capture the set of interaction requirements that can be used to derive the Presentation Model. In order to solve this problem, this paper proposes a methodology in order to capture interaction requirements to complement the functional requirements specification step.

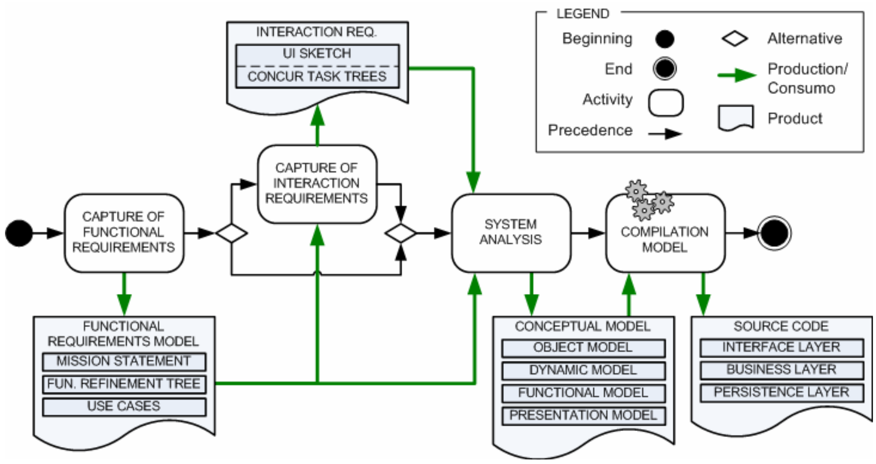

Figure 1: OO-Method methodology

To represent the interaction requirements we have chosen the ConcurTaskTree notation (CTT) [Paternò 97]. This is a notation based on tasks that is well-known in the HCI community. The main reason for using this notation is because it is a formal language, which provides a formal semantics, makes the model verifiable, and avoids ambiguity in the specification.

This paper proposes a set of well-defined rules to use the CTTs to build structural task patterns. These structural task patterns represent a primitive of the OO-Method Presentation Model. Each Presentation Model primitive is represented by only one structural task pattern. Once all the structural task patterns that represent the interaction have been built, the Presentation Model can be obtained automatically using transformation rules. Therefore, all the analyst's efforts devoted to interaction are focused on building the structural task patterns.

However, the construction of the structural task patterns is hard. The CTTs are unreadable for small systems and they are hard to build even though the analyst uses a tool to draw them. For this reason, the proposed interaction requirements model adds a more abstract level. This new level is based on another technique used in the HCI community: the sketch.

Sketches are another instrument to capture interaction requirements in the initial steps of the software development process. They provide some advantages over the structural task patterns. They are easier to build (by the analyst) and easier to interpret (by the user). Therefore, a set of primitives should be defined to provide syntax and semantics to the sketches. Since each sketch primitive represents one of the structural task patterns, there is a correspondence between the sketch primitives and the structural task pattern. The structural task patterns are built as the sketches are drawn. This transformation is hidden for the analyst. This mechanism has the benefit of both notations. On the one hand, sketches are easy to use; and on the other hand, the structural task patterns provide a formal notation. Once the structural task patterns have been built from the sketches, the Presentation Model can be obtained from the structural task patterns by applying the corresponding transformation rules.

The paper is structured as follows. Section 2 presents a review of works related to the capture of interaction requirements. Section 3 explains the details of the OO-Method Presentation Model. Section 4 details how to obtain the Presentation Model from structural task patterns. Section 5 describes the layer that is superimposed on the structural task pattern based on sketches. Section 6 details some aspects related to the task-tree notation used in this paper. Section 7 presents a case study with the proposed method. Finally section 8 presents the conclusions.

2 State of the Art

From a software engineering point of view, interaction modeling is not a key issue when requirements and conceptual modeling is dealt with a software production process. Nevertheless some approaches from the HCI world have been presented in recent years. Many of them, such as DiaTask [Reichart 04], UI Pilot [Puerta 05], UsiXML [Vanderdonckt 04], and KBS [Liborio 05], propose the task model as an abstract interaction model to derive the abstract interface model.

Diatask uses a dialog graph, derived from a task model, to generate an initial abstract interface prototype. This prototype mainly reflects the navigation structure of the user interface. A GUI editor was designed in such a way it allows specifying interface elements in the prototype built. UI Pilot is similar to Diatask. UI Pilot uses Wireframes. They are a simple annotated description of the elements that must be implemented for an interface. Thanks to Wireframes, the user should not have to know the representation of the interface model, since he/she only uses Wireframes. The other method based on the task model is UsiXML. UsiXML defines an abstract interaction model from task models. This abstract model is refined in a concrete interaction model, where the analyst specifies the way in which widgets will be shown. UsiXML has been used in other methods of interaction modeling as KBS. In this method, the UsiXML language is used to generate abstract interface objects. From these objetcs, interfaces are generated. Each element of interfaces generated with KBS are accompanied by the explanation of why and how it was inserted onto the interface.

The CTT notation is used in some of these methods, as UsiXML or KBS [Liborio 05], and in other methods as Wisdom [Nunes 00] and SUIDT [Baron 02] among others. These three methods use CTT to validate the interaction with the user. The last two methods define extensions for the notation. UsiXML also uses CTT as a tool for detecting interaction patterns and relates the Abstract Interface Model with the Domain Model (manually or automatically). However, all these proposals lack a full software development process. These tools only generate interfaces, not functional systems. Other methods use a notation more formal than CTT. In this group is Thimbleby's work [Thimbleby 04], which is based on lineal algebra. In this method, matrix operations model actions that occur when user and system synchronize in what they are doing.

All methods mentioned above are not easy understanding by the user. They use a too complex notation, as Thimbleby's work, or the abstraction level is not enough for the user, as KBS or UI Pilot. However, sketches are a notation closer to the end user. With regard to sketching, many works and tools have been presented: SILK [Landay, 01]; JavaSketchIt [Caetano 02]; and SketchiXML [Coyette 05], which allows the user to electronically sketch interfaces that generate end user interfaces. SILK uses four primitives (line, rectangle, straight line and ellipse), which can be combined to prototype the interface that is later transformed into Visual Basic or Common Lisp. JavaSketchIt generates the Java code of the designed interface. SketchiXML is the only method that allows prototyping with sketches in a multiplatform because it transforms the sketches into UsiXML specifications. Although there are many approaches in the literature, not all of them are focused on the early capture of interaction requirements in a software development process. Those that do attempt this do not support a software production process or the automatic generation of the complete application.

In general these methods do not support interface building from defined interaction patterns. Therefore a restrictive task model is not needed. This leads to a lack of closed transformation, which makes it impossible to generate a complete user interface with a fully functional system.

3 OO-Method Presentation Model

This section presents a detailed explanation of the model that represents the interaction in the OO-Method Conceptual Model. As Figure 1 shows, the Presentation Model (PM) is one of three models related to the OO-Method Conceptual Model. Presentation Model specifies the configuration of a set of sixteen patterns called Just-UI [Molina 03]. These patterns are structured in three layers of abstraction

1. **Level 1:** Hierarchical Action Tree. The Hierarchical Action Tree, or HAT, is a pattern that helps the designer abstractly define how the end user can access the system's functionality.
2. **Level 2:** Interaction Unit. An Interaction Unit (IU) models the way in which the end user will interact with the system. The IU is closely related to domain objects, and how they are visualized and manipulated. Four Interaction Units can be defined:

- **Instance Interaction Unit** (IIU): This abstractly models the presentation of a particular instance from an object class.
- **Population Interaction Unit** (PIU): This abstractly models the presentation of a set of different instances of the same class.
- **Service Interaction Unit** (SIU): This abstractly models a presentation dialog in which the end user can launch a service. The user can insert parameters for the service in this IU.
- **Master / Detail Interaction Unit** (MDIU): this is a model that is a combination of IIU and PIU related with each other.

3. **Level 3:** Elementary Patterns. Level 3 defines those patterns that make up and restrict the Interaction Units. There are 11 Elementary patterns, but this paper only uses those related to Population IU:

- **Display Set Pattern**: this pattern specifies which attributes of an object can be shown. It is associated to a PIU or to a IIU.
- **Action Pattern**: the pattern specifies which services can be launched when an instance of an object is selected.
- **Navigation Pattern**: the pattern specifies which related objects can be accessed when an instance of an object is selected.
- **Filter Pattern**: by specifying this pattern, different values can be entered in order to list a group of objects with some common criteria.
- **Order Criteria Pattern**: whenever this pattern applies to a list of objects, the user can list them using different criteria and order.

4 Correspondence between CTT and the Presentation Model

As Figure 1 shows, interaction requirements are captured by means of sketching methodology. The results obtained using this methodology are supported by a Task Model with CTT [Paternò 97] notation in a way that is transparent to the analyst.

The grammar used in the structural task pattern has the following components:

- Lexical: it is provided by the CTT notation (interaction tasks, system tasks, and abstract tasks).

- Syntactic: it is made up of structural task patterns that are structures of tasks related with each other by means of temporal operators.
- Semantic: it is provided by the correspondence between task patterns and Model Presentation patterns of OO-Method.

Structural task patterns have been defined in a generic way. Therefore, they offer arguments that are instantiated when patterns are used to model a specific interface. In the following figures, these arguments are shown in cursive format, indicating that their names should be instantiated. Arguments with variable cardinality are represented with ellipses (i.e., 1...N). This paper presents the structural task patterns corresponding to the following patterns of the Presentation Model: The Population IU and the Elementary Patterns related to it.

- **Filter and Order criteria:**

Figure 2a shows the structural task pattern for the Filter Elementary Pattern. The proposed CTT pattern presents several arguments: the filter name and the fields that define the filter (Figure 2a). In Figure 2b, each interaction task, which is a leaf represents each of the order criteria.

Figure 2: Filter with CTT notation (a), Order criteria with CTT notation (b)

- **Actions and display set:**

As Figure 3a shows, the interaction task in the CTT models the selection of the instance in which the actions are applied. These actions are business rules that depend on each unusual case: that is the reason why they are arguments. The system tasks shown in Figure 3b represent the fields that will be displayed. These arguments are instantiated depending on the requirements of the system queries.

Figure 3:Action with CTT notation (a), Display set with CTT notation (b)

- **Navigation**:

The interaction tasks in Figure 4 are used to represent the selection of the instance where the navigation is applied. The information related to the selected instance will be shown by means of this navigation. The set of possible destinations builds the arguments of this pattern. Finally, the system task does the navigation.

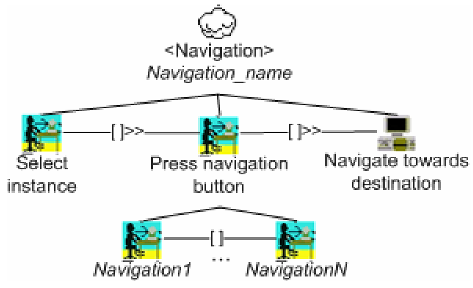

Figure 4: Navigation with CTT notation

Once the correspondences between the structural task patterns and the third level of the Presentation Model patterns are defined, the next step is to do the same with the second level of the Presentation Model patterns.

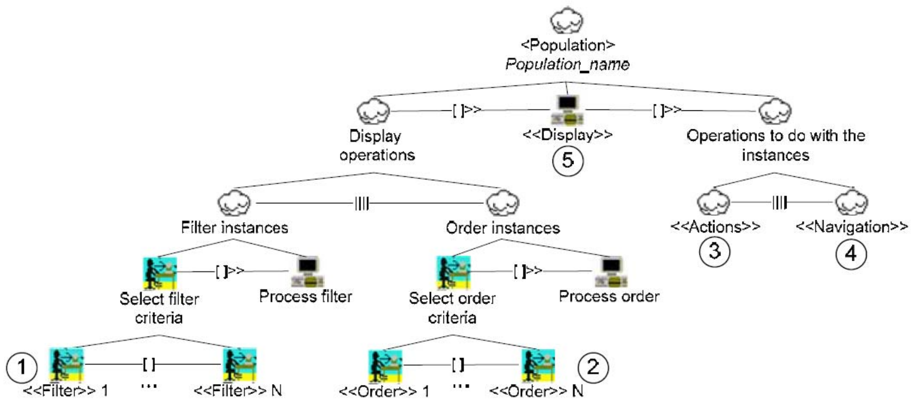

Figure 5: Population with CTT notation

As Figure 5 shows, the CTT that represents the population pattern includes interaction tasks that are in charge of filtering (Filter) and arranging (Order) the instances. The brackets represent grammatical-composition rules. In other words, they are points to hook the leaves to other structural task patterns. A system task is then in charge of showing the instances of the objects (Display) that are filtered and ordered by the selected criteria on the screen. Finally, the user can carry out Action and Navigation operations with these instances, which are represented in the diagram by means of abstract tasks. The numbering system indicates the different structural task patterns that represent elements of the OO-Method Presentation Model in the third level. All of these structural task patterns make up a single structural task pattern that represents the primitive of the second level called Population IU.

Once the CTT has been built using the defined structural task patterns, transformation rules are applied to obtain the OO-Method Presentation Model. By adding the rest of the OO-Method models (Object Model, Dynamic Model and Functional Model) and applying the Model Compilation, the final interface and the system functionality are automatically generated (Figure 1).

5 Construction of Structural Task Patterns from Sketches

The preceding section has shown how to represent primitives of the OO-Method Presentation Model by means of structural task patterns in the requirements-capture step. However, the manual construction of these structural task patterns is very hard, even though there is a tool to support their drawing. Moreover, for small applications, the structural task patterns become illegible due to the huge number of CTTs that are created. This paper proposes a higher abstraction level to represent the interface by means of sketches. The analyst with the help of the user draws sketches that represent the final interfaces. As the skecthes are drawn, the structural task patterns are built automatically in a way that is hidden for the analyst.

In order to define a new model based on sketches, the first step is to establish a set of basic builders. In other words, the primitives for building a sketch to represent the interface should be defined. Due to the graphical characteristics of the sketch, the shape of these primitives is very similar to their visual representation.

The proposal methodology builds sketches and CTT simultaneously. Therefore a biunivocal relationship between sketch primitives and structural task patterns should be defined. In other words, for each structural task pattern, a sketch primitive should be defined. For reasons of space, this work is based on the structural task patterns related to the list of instances.

- **Filter**:

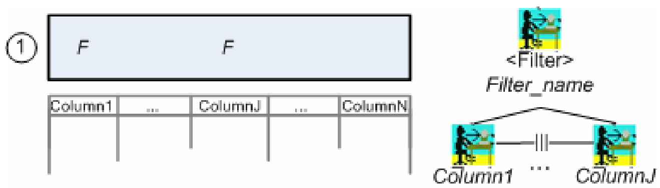

Figure 6: Graphical primitive and structural task pattern for Filter

Filter primitives for sketches specify a filter criterion for the listed instances in the population. The analyst must place a symbol above the columns that the user wants to use in the filter. These marked columns instantiate the arguments of the corresponding structural task pattern.

- **Order criteria**:

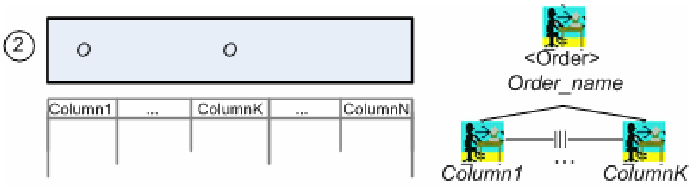

Figure 7: Graphical primitive and structural task pattern for Order criteria

- **Actions:**

The Actions represent operations that can be made with the selected instance in the population list. Some actions are very common (i.e., create a new instance, modify it, or delete it); others are specific to the system that is being sketched. Actions that are drawn instantiate the arguments of the structural task pattern.

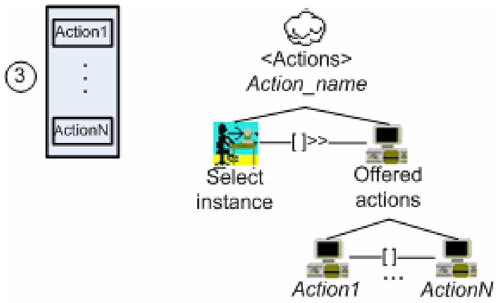

Figure 8: Graphical primitive and structural task pattern for Actions

- **Navigations**:

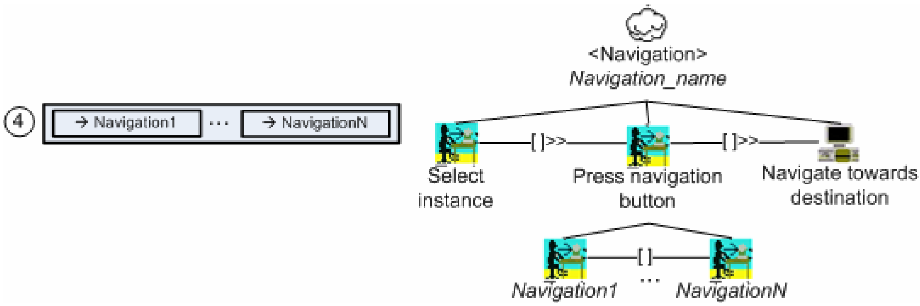

Figure 9: Graphical primitive and structural task pattern for Navigation

The user can access other interfaces in the system throughout the navigation. In other words, the navigation permits access to system interfaces that implement queries or editions on objects related to the object source. For example, starting from an invoice line list, the user can navigate to the client data. Navigations that are inserted in the sketch instantiate the arguments in the structural task pattern.

- **Display set**:

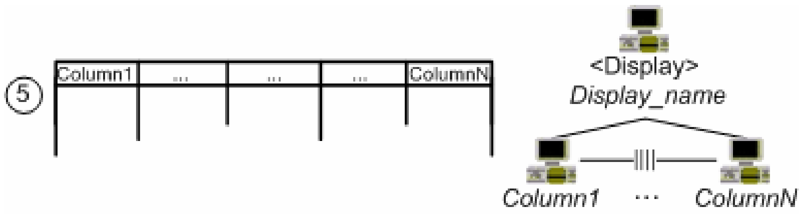

Figure 10: Graphical primitive and structural task pattern for Display set

Display set primitive specifies the columns of the population instances that will be shown graphically. This primitive is a set of columns that the analyst can assign a name to. In the analysis step, (when the Object Model is derived) the columns defined in the sketch are linked with class attributes. The names of the columns inserted in the sketch provide the values of the structural task pattern arguments.

- **Population:**

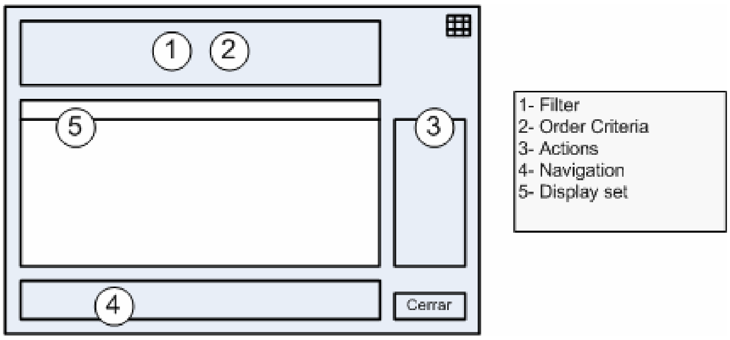

Figure 11: Graphical primitive for Population

This primitive presents a list of instances from a business object class to the user (i.e., a list of customers). As Figure 11 shows, it may include all the primitives that represent elements of the third level in the OO-Method Presentation Model through structural task patterns. Figure 5, presents the structural task pattern that the primitive shown in Figure 11 represents. The numbering system of Figure 5 represents the different primitives drawn in Figure 11.

6 Further Comments on using Task-Tree Notation

Task analysis is a widely accepted practice in Human-Computer Interaction and is at the core of many of the software development methods proposed by this community. Limbourg and Vanderdonckt [Limbourg 03] identify three related poles involved in task analysis: task models, task analysis methods, and support tools. In the following, we will discuss how our proposal deals with these three poles.

In the first pole of task analysis, models are descriptions of the world that capture some facets of a problem. ConcurTaskTrees (CTT) is a modeling language developed by Paternò [Paternò 97]. CTT models are used to describe how people perform the tasks they undertake. The tasks are hierarchically decomposed to the level of basic tasks, which are defined as tasks that should not be further decomposed. The second pole appears in this stage because a stepwise approach is required. Several questions arise at this point of the method.

What is the correct granularity for tasks? Since the criterion for starting and stopping task decomposition is sometimes fuzzy, we have taken some arbitrary but useful decisions. A task tree is built for each use case that appears in the functional requirements elicitation; therefore, the granularity of the root task of our task models is a use case. The use cases are built before the interaction requirements are captured.

Is a hierarchical decomposition appropriate for dealing with tasks? As Diaper acknowledges in [Diaper 03], it is often argued that much of the natural world is not truly hierarchical and would be better modeled as a heterarchy, allowing a more flexible mereology. This concern is better understood if one asks the following question: Do task trees model the interface or the interaction? These are not the same; the first one identifies interface components in order to make a description of the interface, and the second one identifies a path through the interface so as to describe a particular interaction with that interface.

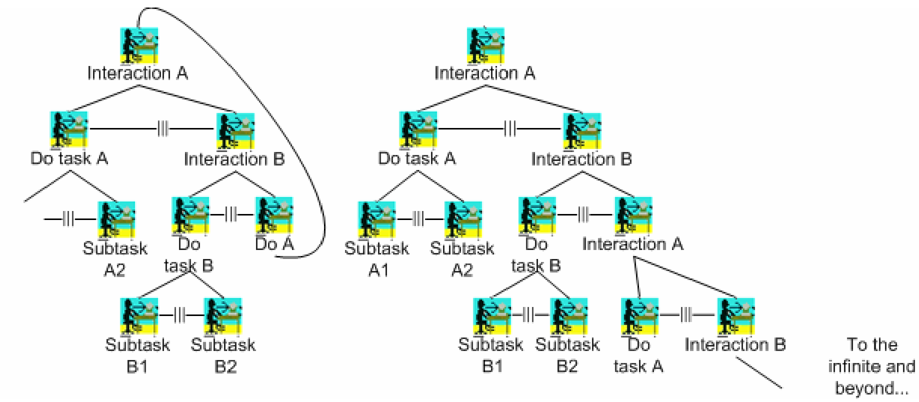

Figure 12: Two inappropriate ways of dealing with navigation in CTT models

Interfaces can be effectively described in terms of a component tree whereas navigation needs cyclic graphs. Task trees are closer to interaction modeling, although temporal operators offer the possibility of simultaneously describing many of the possible paths through an interface (i.e., using the [ ] operator). A problem appears when trying to specify navigation in CTT models. Suppose two tasks (Interaction A and Interaction B) that have access to each other. Figure 12 shows two solutions that describe this requirement. Figure 12.a includes Interaction B as a subtask of Interaction A and then closes a loop by drawing an explicit link between Interaction B and Interaction A; this diagram violates the notation because it is not a tree anymore. Figure 12.b recursively includes each task inside the other one; although this solution complies with the notation, it produces trees of infinite depth.

To overcome this problem, we have defined an implicit semantics for those tasks dealing with navigation. Figure 13 shows a solution to the above mentioned requirement of navigation between Interaction A and Interaction B by creating two task trees and using implicit navigational semantics (the dashed arrows which show the destiny of the navigation are merely informative and would not be part of the model).

Even if navigation is properly addressed, two more questions arise: Are the resulting task trees manageable in big projects? Can task trees be reused? The structural pattern approach is a stepwise approach to task modeling and offers an additional advantage: it allows the reuse of the structural task patterns by defining connection points. Note that each structural task pattern defines a male-plug connection point of the form <pattern type> (see the figures in Section 4). Some of the structural task patterns include socket-like connection points of the form <<pattern type>> (see Figure 5); structural task patterns of the appropriate type can be attached to these connection points (those that define the corresponding male-plug connection point). For example, an Actions structural task pattern (see Figure 3a) would be attached to the <<Actions>> abstract task of a Population structural task pattern (see Figure 5). This composition mechanism is our extension to the CTT syntactic layer. It allows structural task patterns to be reused.

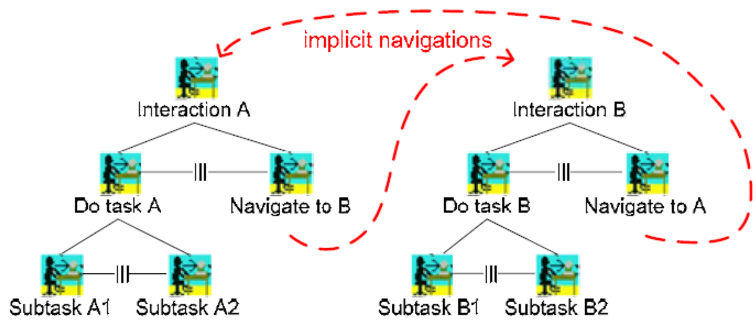

Figure 13: Navigation modeled by implicit semantics

In the third pole of task analysis there should be a task model repository that stores and reuses structural task patterns. This repository should contain the primitives described in section 5 related to interaction modeling based on sketches. If task models are transparent to the analyst, they are definitely manageable.

7 Case Study

An application for managing the water supply has been selected to apply the proposed method. To simplify the case study, the example is focused on the List meters task.

This task obtains a list with all the meters of the system.

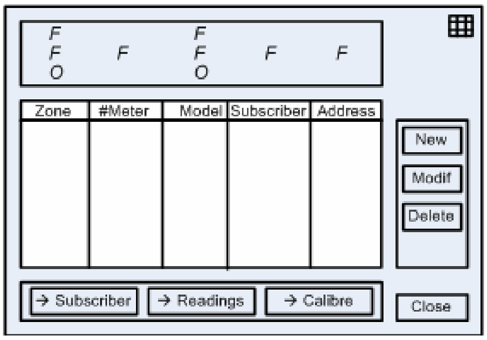

Figure 14: Sketch for List meters

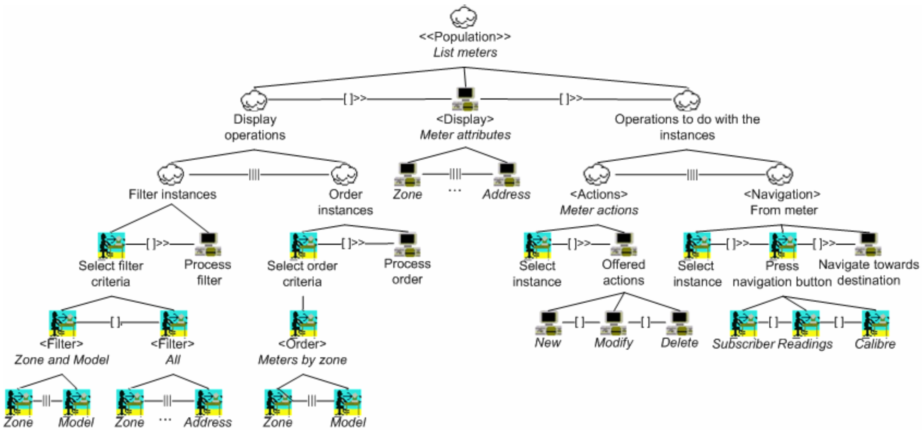

Figure 15: CTT for List meters

The sketch shown in Figure 14 represents a Population IU for the List meters task. The list of meters has two filters and one order criterion. The Filter (F) and Order (O) primitives have been used in this example. Each column that has the letter F above it represents a filter argument. The same criterion is used for the Order primitive, but in this case the letter O is placed above the column. The sketch also contains a set of buttons for Actions (the right side of the sketch) and Navigations (the lower portion of the sketch). The column names represent the Display set primitive. The CTT model is built automatically when the analyst draws the sketch (Figure 15).

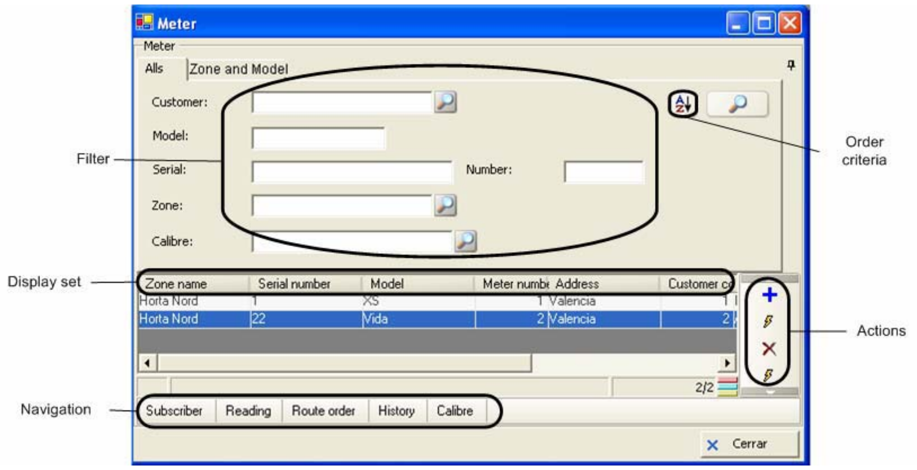

Figure 16: Interface for List meters

The transformation rules explained in section 4 can be applied to the CTT shown in Figure 15 to derive the Presentation Model. The system is automatically generated by applying the Model Compiler to the Presentation Model together with the rest of models that make up the Conceptual Model. Figure 16 shows this system.

8 Conclusions

In this work, an interaction requirement specification methodology has been presented. This methodology is based on the construction of a task model. An interesting contribution of this paper is the definition of structural task patterns that aid in the specification of the interaction by offering a systematic way of building the task model. The structural task patterns are patterns of tasks that represent common interactions between the user and the information system. These patterns are formally defined as an extension of the syntactical and semantical layers of the CTT notation.

Furthermore, we have demonstrated that an abstract interface model can be automatically derived from the task model using our approach. We have instantiated this transformation for the OO-Method by defining a set of transformation rules between structural task patterns and the primitives of the view of the OO-Method Conceptual Model that is devoted to interface modeling; that is, the Presentation Model. A Model Compiler can then take the Conceptual Model (which includes the Presentation Model) and generate the source code of the application.

However, experience has shown that CTT diagrams grow to be almost unmanageable. To overcome this difficulty, we have proposed leaving the task models at the background of interaction modeling. A layer of sketches has been superimposed on top of the task model. The task model is now transparent to the analyst. The sketches and the task models are synchronously built. This is achieved by defining a set of correspondences between the sketching primitives and the structural task patterns. The task-model construction rules influence the drawing of the sketches. However, even though the tight coupling between sketches and task models reduces the degree of freedom that the analyst has to create the sketches, it allows the process to benefit from the formal properties of CTT-based models.

Future work includes the implementation of a tool that supports the proposed methodology: OO-Sketch. The tool should have a shape recognition engine to identify the interface components drawn with an electronic pen. Moreover, the tool should have defined a set of patterns with all the sketches primitives, so that the analyst can drag them instead of drawing. At the same time, the corresponding task trees should be built by applying the correspondence rules defined in this paper. This functionality, together with the transformation of the task model to the Presentation Model, will offer technological support for automatically generating application interfaces efficiently and will allow early feedback from the user.

Another advantage of using a formal language like CTT as an underlying stratus is the possibility of validating the sketches to which the task trees are bound. Therefore, we intend to implement a syntactic validator for task trees so that only the construction of well-formed CTT trees (and sketches) is permitted.

Last, but not least, we plan to carry out an empirical study of our proposal. A series of experiments will be conducted on the OO-Sketch tool to verify the methodological improvements offered by our technique.

Acknowledgements

This work has been developed with the support of MEC under the project DESTINO TIN2004-03534 and cofinanced by FEDER.

References

[Baron 02] Baron M., G. P. "SUIDT: A task model based GUI-Builder." Task MO-dels and DIAgrams for user interface design (TAMODIA) (Romania 2002).: 64-71.

[Caetano 02] Caetano, A., Goulart, N., Fonseca, M., Jorge, J. JavaSketchIt: Issues in Sketching the Look of User Interfaces. AAAI Spring Symposium - Sketch understanding.AAAI Press. (2002). pp. 9-14.

[CARE] Care Technologies: http://www.care-t.com Last visit: April-2007.

[Coyette 05] Coyette, A., Vanderdonckt, J. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. INTERACT 2005, LNCS 3585. (2005). pp. 550-564.

[Diaper 03] Diaper, D. The Handbook of Task Analysis for Human-Computer Interaction, Lawrence Erlbaum Associates. (2003).

[Insfran 02] Insfrán E., Pastor O. and Wieringa R. Requirements Engineering-Based Conceptual Modelling. Requirements Engineering 7 (2) (2002): 61-72.

[Landay 01] Landay, J., Myers, B.A. Sketching Interfaces: Toward More Human Interface Design. IEEE Computer 34. (2001). pp. 56-64.

[Liborio 05] Libório, A., Furtado, E., Rocha, I., Furtado, V. Interface design through knowledge-based systems: an approach centered on explanations from problem-solving models. Workshop on Task models and diagrams, ACM Press. (2005). pp. 127-134.

[Limbourg 03] Limbourg, Q., Vanderdonckt J. Comparing Task Models for User Interface Design. The Handbook of Task Analysis for Human-Computer Interaction. Mahwah, Lawrence Erlbaum Associates. (2003).

[MDA] MDA "Model Driven Architecture" http://www.omg.org/mda Las Visit: April-2007.

[Molina 03] Molina, P., User interface specification: from requirements to automatic generation, PhD Thesis, DSIC, Universidad Politécnica de Valencia, March 2003 (in Spanish).

[Nunes 00] Nunes, N. J. y J. F. e. Cunha "Wisdom: a software engineering method for small software development companies." Software, IEEE 17(5) (2000): 113-119.

[Pastor 01] Pastor, O., Gómez, J., Insfrán, E. Pelechano, V. The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems, 26(7) (2001): 507-534.

[Paterno 97] 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.

[Puerta 05] Puerta, A., Micheletti, M., Mak, A. The UI pilot: a model-based tool to guide early interface design, San Diego, California, USA, ACM Press. (2005). pp. 215-222.

[Reichart 04] Reichart, D., Forbrig, P., Dittmar, A. Task models as basis for requirements engineering and software execution. Conference on Task models and diagrams, Prague, Czech Republic, ACM Press. (2004): pp. 51-58.

[Thimbleby 04] Thimbleby, H. "User interface design with matrix algebra." ACM Transactions on Computer-Human Interaction (TOCHI 11(2) (2004): 181-236.

[Vanderdonckt 04] Vanderdonckt, J., Q. Limbourg, et al. USIXML: a User Interface Description Language for Specifying Multimodal User Interfaces. Proceedings of W3C Workshop on Multimodal Interaction WMI'2004, Sophia Antipolis, Greece. (2004).