Challenges for a task modeling tool supporting a task-based approach to user interface design

Costin Pribeanu
National institute for Research and Development in Informatics – ICI Bucharest
Bd. Maresal Averescu Nr. 8-10, Bucharest, Romania
+40 (0)213160736 - pribeanu@ici.ro

ABSTRACT
In a task-based approach to user interface development, the task model is given a leading role among other models. Task modeling is a key activity in the design of a usable user interface. Task patterns that are based on both task and domain models are typical interaction structures that are capturing operations performed onto domain objects. This paper is discussing some key requirements for a task modeling tool aiming to support model transformations in UsiXML: full computer-aided task decomposition, model validation, model simulation, operations with task patterns, and simultaneous access at task and domain model elements based on mapping rules.

Author keywords
UsiXML, model transformation, task patterns.

General Terms
Design, Experimentation, Human Factors, Verification.

ACM Classification Keywords
D2.2 [Software Engineering]: Design Tools and Techniques
– Modules and interfaces; user interfaces. D2.2
[Software Engineering]: Miscellaneous – Rapid Prototyping; reusable software. H5.2 [Information interfaces and presentation]: User Interfaces – Prototyping.

1. INTRODUCTION
Models are used to capture design knowledge needed for the construction of the future user interface (UI). Main concepts abstracted into these models refer to users, tasks, application domain, presentation and dialog. In a transformational approach to UI design the task and domain models are the main source models for the derivation of presentation and dialog models at various levels of abstraction [2], [3], [6].

Model-based approaches which are giving the task model a leading role among the other models are also referred to as task-based approaches. There are several model-based approaches that are using the information from both the task and domain models in order to exploit task patterns associated with typical operations performed onto domain objects [10], [14]. A limitation of these approaches is the fact that they are exploiting only the last layer in the task models.

A well known task modeling notation is CTT (Concur Task Tree) which was widely used in various interactive environments such as CTTE (CTT Environment) and Teresa [8]. As pointed out by Luyten et al., CTT served as inspiration for several tools aiming at supporting task modeling in UsiXML [7]. CTTE enables the designer to create task trees and to specify task properties such as task type, frequency, and estimated execution time. There are two important CTTE features that represent a first key requirement for a task modeling tool: the model validation and the model simulation.

The goal of the UsiXML project is to develop an innovative model driven language supporting the “µ7” concept: multi-device, multi-user multi-cultural / language, multi-organization, multi-context, multi-modality and multi-platform. UsiXML will define a flexible methodological framework that accommodates various development paths as found in organizations and that can be tailored to their specific needs [5].

This paper is aiming to discuss some key requirements for a task modeling tool aiming to support the model transformations in UsiXML: full computer-aided task decomposition, operations with task patterns, and simultaneous access at domain and task model based on mapping rules.

In order to explain our approach we will use as example a software assistant for formative usability evaluation that was previously developed in a task-based approach. We will focus on the relationships between objects and analyze the categories of tasks which are afforded by each type of relationship. Then we will look for mappings between the domain model and the task model with the aim of covering the whole task hierarchy.

The rest of this paper is organized as follows. In section 2, we will briefly describe our task modelling framework and some general mappings between domain and task models. We will also present our example. In the next section we will discuss the difficulties associated with a transformational task-based approach and some limitations of the existing task modeling tools. The paper ends with conclusion in section 4.

2. APPROACH
2.1 The layered task model
In our approach we distinguish between three decomposition layers, which are relevant in the task modeling for user interface design [11]:

- A functional task layer that results from mapping application functions onto user tasks. Each function corresponds to a business goal, which is accomplished by carrying on one or several user tasks.
- A planning task layer that results from the decomposition of functional tasks up to the level of unit tasks, having a clear relevance for the user [4]. This layer shows how users are planning task performance by decomposing a task in sub-tasks and giving an ordering preference for each of them.
- An operational task layer that results from the decomposition of unit tasks up to the level of basic
tasks, showing how a unit task will be actually carried on by using various interaction techniques [13].

In terms of a pattern language philosophy [1], domain and task models are the main forces driving the model-based design of a UI. In a previous work, we identified several task-domain mappings that are useful for the model-based derivation of the user interface [10]:

- **Unit tasks – domain objects**: operations performed onto domain objects, such as create new, edit, display or delete, are modeled as unit tasks in the task model. Unit tasks could have a simple operational structure or could be nested in a goal hierarchy.

- **Basic tasks – domain object attributes / available commands**: operations performed onto domain object attributes are mapped onto information control basic tasks while available commands on the target platform are mapped onto function control basic tasks.

In the same work, we presented a software tool exploiting these mappings between task and domain models in order to produce the operational task layer. A second key requirement for a task modeling tool is to provide support for computer-aided task decomposition in a transformational approach. This means to cover also the decomposition at functional and planning levels by exploiting the mappings between the task and domain models, as shown in Table 1.

**Table 1. The task-domain mappings**

<table>
<thead>
<tr>
<th>Task model</th>
<th>Task goals</th>
<th>Domain Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional layer</td>
<td>Class and object management</td>
<td>Classes and objects</td>
</tr>
<tr>
<td>Planning layer</td>
<td>Add, edit, display, delete objects and relationships</td>
<td>Objects and relationships</td>
</tr>
<tr>
<td>Unit tasks</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational layer</td>
<td>Basic tasks – info control</td>
<td>Object attributes</td>
</tr>
<tr>
<td></td>
<td>Edit or display object attributes</td>
<td></td>
</tr>
<tr>
<td>Operational layer</td>
<td>Basic tasks – function control</td>
<td>Objects, relationships and attributes</td>
</tr>
<tr>
<td></td>
<td>Commands (transactions, navigation)</td>
<td></td>
</tr>
</tbody>
</table>

Basic tasks for function control have as operational goals to start, confirm or cancel a transaction (e.g. add, edit, ok, cancel) over domain model elements or to navigate between interaction units.

### 2.2 Example

We will use as example a software assistant for formative usability evaluation that was developed in a task-based approach [12]. The purpose of formative usability evaluation is to identify and fix usability problems as early as possible in the development life cycle. There are two main categories of methods frequently used: inspection-based evaluation (heuristic evaluation, guideline-based evaluation) and user testing. In each case, a heuristic and / or usability guideline is used to document the usability problem. Usability problems are recorded for each evaluation task.

The main goal of the application is the management of usability problems (UP).

The functional layer shows the business goals that should be accomplished with the target application. Each function is mapped onto a user task. In our example, the tasks are accomplishing the management of four classes of objects: evaluation tasks, usability problems, heuristics and guidelines. The task decomposition at functional level is presented in Figure 1 using the CTT notation.

![Figure 1. The task ManUP – functional layer](image)

Usability problems are the central class in this application. Heuristics are used to document a usability problem. Guidelines are used to detail a heuristic. The heuristics are displayed in a list box. When a heuristic is selected, its definition is displayed in a text box below. Several heuristics could be associated to a usability problem.

The user could consult the guidelines detailing a heuristic in a separate window (by pressing the ShowGuidelines button). In Figure 1, the interaction unit for editing a usability problem is presented.

Ideally, it should be possible to create the user interface illustrated in Figure 1 by applying transformation rules to the task and domain models.

![Figure 2. The user interface for editing a usability problem](image)

Tasks on the first decomposition level in Figure 1 are interactively specified in CTTE since there is a simple one-to-one mapping from application functions to user tasks. The abstract (complex) tasks are iterative (infinitely) and linked with the “|=|” temporal operator (could be performed in any order).
The next decomposition level could be obtained automatically, by performing a task to task transformation. This task pattern has four types of operations for each class: display, add, edit and delete. The four tasks are also iterative (infinitely) and linked with a `|=|` temporal operator (could be performed in any order) like their parent tasks.

When simulating the task model, we experienced difficulties since CTTE supports both task id and task name, but is using the task name in simulation. So we had to rename all similar tasks in order to get unique names (e.g. AddT, AddUP, AddH, AddGIdln), which resulted in an extra work and reduced the specification readability. Therefore, a third key requirement for a task modeling tool is to automatically assign a task id, to use it for internal validation and model simulation and to display the task name (which should not be unique).

This requirement is important since it makes possible to define task patterns and to perform operations with them easier.

3. TRANSFORMATION ISSUES

3.1 The development path

The flexibility principle of UsiXML methodology makes it possible to support different development paths. In our approach, the concrete interface model is produced by applying transformation rules from task and domain models. The rationale for this approach is explained below.

According to Norman, changing the artifact is changing the nature of the task [9]. The change of platform is not only affecting the operational task layer but also the functional and operational layers. The task requirements are different for desktop computers and mobile devices. Although it is possible to migrate from a desktop computer to a mobile device, this works well for some tasks that are likely to be performed in a particular context of use. For example, consulting the heuristics – guidelines hierarchy could migrate to a mobile device. Otherwise, each platform is supporting specific interaction metaphors which are shaping the overall user interface design.

Another problem is the change of modality. Some application functions could support the change of graphic modality with voice. However, for a relatively large application is difficult to ensure an acceptable level of usability without radical changes in the task model, at least at planning level. For example, respecting accessibility guidelines only ensures that a disabled user could access the content. In order to make it usable, the interaction spaces should be tailored to the limitations of human working memory. In other words, the user interface should be structured differently for a blind user, in smaller interaction units.

The choice of what to show into an interaction unit is an important design decision that sometimes is taken beforehand. Many designers put forward an interaction concept or a metaphor that provide with a better user experience. This means that the choice of structuring the user interface in interaction units could be a subjective human decision. In many cases, criteria like familiarity, consistency (with other applications or real life artifacts) prevail.

The goal hierarchy in the task model, the relationships between objects and the mappings between task and domain models could positively influence the design decision on structuring the user interface in a usable way but we consider that these criteria should be used at the concrete interface level.

In all these cases (change of platform, change of modality, user interface structure) the task model is also changing because of the navigation tasks. When the user interface is split into several interaction units, the number of navigation tasks increases. Simulating the task model helps in structuring the interface since it makes obvious the navigation tasks that are not needed.

For example, the list of heuristics could be available in the same window or in a separate window opened when the user wants to associate a heuristic to a usability problem. Since each usability problem is documented by at least one heuristic, the best is to have the list of heuristics permanently displayed in the editing window, like in Figure 1. From a methodological point of view, we advocate for a complete task specification at operational level (including navigational tasks) based on task and domain mappings as a prerequisite for the user interface derivation.

3.2 Task & domain to task transformations

We consider that transformation rules should be embodied in a set of tools that are producing UsiXML specifications.

In order to make a UsiXML specification manageable with a reasonable effort, several functions are required for such a tool: opening a model, finding a model element, inserting the specification of a model element into the model, and combining small models into larger models.

Transformations from task and domain to task are producing the planning and operational layers in the task model. Actually, these transformation rules are producing task patterns that could be further exploited in similar applications.

The transformations producing the planning layer are exploiting the semantic relationships between objects. In our example, editing a usability problem is a complex task with several sub-goals: editing the attributes of the usability problem, displaying info about a heuristic, editing the associated heuristics, and displaying the guidelines associated with a heuristic.

Depending on the use of semantic relationships, the tasks on the second decomposition level in Figure 2 could have a specific degree of complexity. For example, the task “EditUP” has a hierarchical structure and is further decomposed in several sub-tasks corresponding to the aforementioned sub-goals.

In Figure 3 the planning layer for the task EditUP is presented.

There are three relationships between objects used in this task. The first is the relationship between the evaluation
tasks and usability problems (aggregation) which is visible (the task id is displayed). The second is the relationship between a usability problem and the associated heuristics (1...*) which is visible. The second is the relationship between a heuristic and the associated guidelines which could be made visible by pressing the “Show guidelines” button.

When editing a usability problem, the user can change the UP attributes and the associated heuristics but can only consult the heuristics and associated guidelines. As it could be seen in Figure 3, the use of domain objects and their relationship are structuring the task model at planning layer.

![Figure 3. The task EditUP – planning layer](image)

A similar task pattern applies for adding a usability problem. The only difference is the task for selecting the usability problem which is no longer needed. In this respect, a fourth key requirement for the task modeling tool is to provide with operations over a task patterns collection.

The transformations rules are using attributes of the task in the source task model (such as canonical task type and task nature) and the information available in the domain model. In our approach, the source task is the parent task (which is decomposed).

A fifth key requirement for a task modeling tool is to provide a way to specify a link between each task and the corresponding domain model element.

In this respect, the corresponding element for each task could be an object class, a relationship, an object attribute or none (for navigational tasks). This mapping will be further exploited for task decomposition, concrete user interface derivation, and writing the code for event processing. Ideally, the task modeling tool should provide a model validation that extends the CTTE feature (based on temporal operators) with checking the consistency of mappings between task and domain elements.

If no relationship is involved, a simpler task pattern is needed for editing a domain object. The task decomposition for editing the evaluation tasks (EditT) is presented in Figure 4.

![Figure 4. The task EditT – planning layer](image)

We mentioned several key requirements for a task modeling tool supporting a transformational approach to user interface design. The purpose of such a tool is to provide with a pattern language for task modeling including pattern definition based on task and domain mappings, and operations with task patterns over the UsiXML task model.

We discussed several aspects influencing a transformational approach to user interface design. The relationships between domain objects and their centrality for the user tasks give a sort of directness of the user interface structure. There are also other aspects that are shaping a user interface, like the platform and modality for which is primarily developed as well as the context specific requirements.
Using a task modeling tool supporting the aforementioned requirements makes possible to produce more than a task model. In this respect we consider useful to include all the relevant information from the domain model in order to get an extended task+domain model. This extended task+domain model is the source model for the transformations from task and domain to the concrete user interface. As such, it could serve as a sort of abstract user interface that support model checking (including task – domain mappings), model simulation and reasoning about the user interface usability.

Acknowledgement
This work was carried on in the framework of the Eureka Cluster ITEA2 European project USIXML (08026) funded under the PNCDI II Innovation Project 294E.

References