Investigating the Relation between the Task Model and the Domain Model in a Task-Based Approach to User Interface Design

Costin Pribeanu
National Institute for Research and Development in Informatics
Bd. Mareșal Averescu Nr.8-10, București, Romania
pribeanu@acm.org

ABSTRACT
One limitation of existing model-based approaches for user interface design comes from exploiting models in isolation. The objective of this paper is to elaborate on a design framework aiming to integrate design knowledge by considering task and domain model simultaneously. In this respect task modeling at operational level is integrating requirements coming from several activities: task analysis, domain modeling and ergonomic analysis. A special attention is paid to relationships between domain objects. From which two types are discussed: aggregations and associations. Analyzing relationships makes it possible to include appropriate interaction structures into the task models. In turn, this enriched task model leads to the identification of interaction object groups and pre-defined dialog units in the user interface.

Keywords
task-based design, user interface design, operational task modeling, relationships in domain models

INTRODUCTION
The quality of user interfaces is a trade-off between utility and usability. Usable systems with a poor functionality create the feeling to the user that the user interface (UI) is leaving out of his reach many capabilities. For example, potential user tasks afforded by the domain model are not available in the interface although the user might need them.

On the other hand, many systems that provide the user with a reach but unstructured set of functions are perceived as being built regardless of the context of use. Many powerful facilities which are created by designers for their own pleasure lead to user interfaces providing with little user guidance and poor task compatibility.

During the last decade many approaches aiming to support automatic user interface generation have been developed. Some model-based approaches like UIDE [8], MECANO [7] and JANUS [1] were mainly based on the domain model. Other approaches like TRIDENT [9] or ADEPT [10] took also into consideration the system functions or the user’s task.

Both kind of approaches failed to develop tools with wide applicability because they put too much emphasis on only one model.

Using the domain model is useful to derive the basic constituencies of the user interface i.e. interaction objects, by which the user can interact with the computer. This is done by applying ergonomic criteria in order to choose the appropriate interaction technique for a given operation type (data entry or data display) and for given object characteristics like data type or data length.

Task models as resulted from early task analysis could only provide with useful information for improving user guidance and task compatibility. They help in structuring the interface following a task-based approach. Although the task of the user is to manipulate application data there is no clear relation between domain and task models as regard to their contribution to the user interface derivation. In this respect task models do not explicitly refer to devices giving access to domain objects and relationships.

The objective of this paper is to elaborate on a design framework aiming to integrate design knowledge by considering both domain and task models. In order to do this we will analyze the relation between three models: domain, tasks and presentation and we will take a closer look at the relationships between domain objects.

In a previous task-based approach [6] we developed some heuristics to derive a presentation model from a task and a domain model. This was done by grouping interaction objects around basic functions required by the user’s task. The resulting interaction object groups could be used as basic building blocks for user interface design. In order to enable the representation of constructs resulted from task, domain and interface models an operational task model was proposed.

Analyzing the relationships between objects makes it possible to more clearly express how the domain model is used when building a task model. This way we can identify typical interaction structures afforded by these relationships and to embody them into the task model. In turn, by exploiting this enriched task model make it possible the user interface derivation from at least two points of view. First, it provides the designer with bigger building blocks than unrelated interaction object groups. Second, these typical interaction structures provide with useful
The rest of this paper is structured as follows. In section 2 the design framework is briefly presented. A taxonomy of interaction objects is presented in section 3. Then task decomposition at operational level is detailed in section 4. In Section 5 two types of relationships between domain objects are analyzed. The results of this analysis are further used in section 6 in order to show how task and domain model could help to derive the presentation of the user interface. The paper ends up with conclusion in section 7.

THE DESIGN FRAMEWORK

In a previous work a framework for task-based design [6] has been proposed. This approach comprises five stages: choosing of appropriate interaction objects based on ergonomic criteria, task design completion, grouping of interaction objects according to semantic and functional criteria, identification of dialog units and adding of navigation control.

The framework needs three pre-requisites: an initial task model, an application domain model and a user interface development environment model (UIDE). In our approach we distinguish between two related activities: task analysis and task design.

Task analysis is an activity that is carried on early in the development cycle when technological options have not been decided. The result of this work is an initial task model that is reflecting basic requirements of the target domain. The lowest level tasks are unit tasks, associated with device-independent functional goals.

Task design means to rephrase the unit tasks (the leaves in the task hierarchy) in terms of available commands. When using a graphical UIDE this means to decide which are the appropriate interaction techniques that can support these tasks. The result of this work is the operational task model. The lowest level tasks are usually associated with interaction objects and they show how functional (or planning) goals are operationalized using a particular interaction object.

Therefore the designed task model acts as a cornerstone for the user interface design since makes it possible to represent the way by which the user is actually manipulating application data using a given technology.

The distinction between functional level and operational level in task decomposition is also important because it shows two stages in the development life cycle: task analysis and task design. Task analysis is usually done before choosing (or at least, regardless) the user interface development environment. It could also be the result of a different development team (including psychologists, sociologists, ethnographers) than the one which designs the user interface.

A similar distinction between task analysis and design is made in ADEPT [10] where task decomposition goes down up to the level of actions performed on interaction objects.

Although it provides the designer with more insight into the user’s task (including cognitive actions) the ADEPT framework could only support particular designs. The level of detail is to deep to provide with useful abstractions able to support user interface derivation in an effective manner.

Selection of interaction objects is done according to requirements coming mainly from the domain model by using ergonomic-based selection rules. Then grouping of interaction objects according to the task model is done in order to satisfy user guidance, cognitive work load and task compatibility requirements.

Interaction object groups are designed in a virtual space. Further structuring in dialog units takes into account an allocation strategy as well as information coming from the domain, task and platform models.

This approach is different from TRIDENT [9] where the first step is to identify logical widows, using a chaining graphs representation of system functions. This is a top-down process since abstract interaction objects are derived for each logical window.

The reason to take a different ordering is twofold. First, relying on system functions is not a task-based approach: user centred design means to embed system function into the user’s task and not vice versa. Second, we need some measure of the required screen space for each IO group in order to decide the structuring of the interface in dialog units. In this respect, working in a virtual space provides with more flexibility.

INTERACTION OBJECTS SELECTION

Graphical user interfaces are modelled in terms of interaction objects and dialog units. Abstract interaction objects (AIO) are supporting a generic interaction task like selecting an item from a list, choosing an option from a menu or pressing a button that triggers a transaction.

Interaction objects are dynamic if they accept input from or display information to the user and static if they are only used to present or to organise the information on the screen. However both categories could be termed as interaction objects since the user is actually interacting with them either perceptually or physically.

Dynamic interaction objects could be used for scrolling, navigation and control. Control AIOs are used to control interaction, functions (actions) or both. Information control AIOs are used to display and modify attribute values. The distinction between navigation and function control is not always clear. For example buttons from a main dialog box that only give access to some functions are rather navigation AIOs.

Function control AIOs are triggering processes over the application data. For example, a “Update” button will run the action of recording changes of some attributes in the database. Usually information control objects are grouped around a function control object. For example, the user can input the customer name using an information control
object and perform a search operation by pressing a command button.

Dialog interaction objects are a special type of AIOs that have standard interacting facilities and are mainly intended to host a dialog. In this respect they could be seen as containers of other AIOs. However, since they are provided with built-in interaction capabilities they could also be seen as interaction objects.

According to these definitions, taxonomy of AIOs is given in Table 1.

<table>
<thead>
<tr>
<th>Class name</th>
<th>Function</th>
<th>Typical IOs in the class</th>
</tr>
</thead>
<tbody>
<tr>
<td>static AIO</td>
<td>identification</td>
<td>icons, labels, group titles</td>
</tr>
<tr>
<td></td>
<td>grouping</td>
<td>group boxes, rectangles, lines, polylines</td>
</tr>
<tr>
<td></td>
<td>description</td>
<td>static images</td>
</tr>
<tr>
<td>dynamic AIO</td>
<td>scrolling</td>
<td>scroll arrows, scroll buttons, scroll bars</td>
</tr>
<tr>
<td></td>
<td>navigation</td>
<td>text links, image links, information control + link, menus, navigation buttons</td>
</tr>
<tr>
<td></td>
<td>information control</td>
<td>list boxes, combo boxes, radio buttons, check boxes, text boxes, text, status bars, progress indicators</td>
</tr>
<tr>
<td></td>
<td>function control</td>
<td>menus, command buttons</td>
</tr>
<tr>
<td>dialog AIO</td>
<td>context control</td>
<td>windows, dialog boxes, panels, forms, message boxes, web pages</td>
</tr>
</tbody>
</table>

Table 1. AIO classes

The reason of this taxonomy is twofold. First, it enables an abstract specification of the user interface thus making possible to re-use it in different physical environments.

Second, it provides the designer with a mapping between the user’s goals and the technological means that are available. This mapping is then used to further decompose unit tasks at operational level.

The selection of interaction objects could be automated by using selection trees that exploit the information provided by the domain model: data type, data length, type of data processing (input, output, input / output), type of data domain (known, unknown), number of input values to choose and number of possible values within the domain. Model-based approaches like MECANO (7), UIDE (8) or TRIDENT (9) use this kind of mapping for generating (part of) the presentation.

OPERATIONAL TASK MODELLING

The functional task decomposition stops at unit task level [2]. From the point of view of task analysis, unit tasks are showing “what-to-do” knowledge, regardless how this work will be actually carried on. Since the task will be accomplished using a given technology, the representation is said to be device independent.

Next step is the operationalisation of unit tasks up to the level of basic tasks. This task decomposition requires considering the technological means by which the user will accomplish his goals. As shown before, this needs to integrate into the task model basic tasks by which the user can interact with the computer.

A basic task is defined as the lowest level task that is using a single interaction object or a single external objects or serves a communicational goal [6]. In this respect, the stopping criteria for task decomposition are an interaction object (for interaction tasks, application tasks and cognitive user tasks), an external object (for manual user tasks), and a distinct step in communication (for communication user tasks). Decomposition of basic tasks up to action level is beyond our task-based approach.

The decomposition at operational level is device dependent since it shows how the unit task will be actually carried on, using interaction objects. In this respect, it provides with a mapping between presentation and task models. Since interaction objects are chosen according to the nature of domain objects it also integrates information coming from the domain model.

In [6] it was shown that the operational task model suggests the first level of aggregation of interaction objects into IO groups. Interaction object groups are the lowest level units of the interface that have associated both semantics (the task goal) and syntax (the ordering of basic tasks as defined by the temporal relations). However, if we consider also relationships in the domain model the mappings between domain, task and presentation models are more complicated and depends on the task context.

In order to illustrate how the relationships between domain objects affect these mappings we will take an example. The task is to build a guide using a tool for working with guidelines. A guideline could be hierarchically structured, from more general to more specific guidelines. A guide can have several bases; a base contains several sections an each section contains several guidelines. A guideline can be associated with other objects like references, examples and criteria. In Figure 1 task decomposition for the unit task “edit guideline” is presented.

The representation is using the CTT graphical notation [4] and highlights four types of tasks: abstraction (complex tasks that are further decomposed), interaction (data input), application (tasks performed by the system, including display of application data) and user tasks. The graphical notation is illustrated in Table 2.

<table>
<thead>
<tr>
<th>Abstraction</th>
<th>Interaction</th>
<th>Application</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Abstraction" /></td>
<td><img src="image" alt="Interaction" /></td>
<td><img src="image" alt="Application" /></td>
<td><img src="image" alt="User" /></td>
</tr>
</tbody>
</table>

Table 2 Graphical notation of tasks in CTT

In our task-based approach interaction, application and user tasks are basic tasks (as in CTT, although the definition and the stopping criteria are different). Interaction tasks are using an information control or a function control interaction object whilst application tasks are using information control objects (for data output only).

At first sight, the task decomposition in Figure 1 suggests
several interaction object groups, corresponding to semantic decomposition criteria. This representation shows the nested structure of unit tasks that are used in different task contexts. In this task context, a function control object ("new" button) enables the user to create a new guideline base if he does not find it in the list and could be useful when the guidelines are recorded in a source-oriented approach.

In the case of nested unit tasks there are two situations:

- both IO groups that implement the unit tasks are placed within the same dialog unit – this is the situation when there is a semantic relation between the IO groups and the user could be provided with feedback;
- a separate dialog unit is allocated for the inner task – this is the situation when either there is not enough screen space or when the task is invoked from different task contexts.

In the second case the inner task can be factored out in the task model. For example, "new base" could be invoked from base management tasks as well as from the "edit section" unit task. This situation is very frequent in database applications.

We took this example in order to show that interaction object groups derived from the domain model are covering only a part of the basic layer of the interface model. This example also highlights the limitations of existing approaches that are either mainly based on one model (task or domain) or do not take into consideration the relationships between domain objects. Performance of tasks like "new section" or "new base" depends on the task context: in a final organisation of guidelines they might not be allowed.

**EXPLORING THE RELATIONSHIPS**

In this paper we refine the framework in order to express more clearly the relation between the various models involved. The purpose is to take a closer look at the domain model, more precisely at the relationships between the domain objects. The objective of this preliminary step is to explore the potential tasks that could be afforded by these relationships.

The domain model defines the objects of the applications. There are three kinds of information, which are of particular interest for user interface design: domain objects, object attributes and relationships between objects. From our point of view the domain objects the user can manipulate through the interface are relevant.

The internal structure of an object and the attribute types are useful to select appropriate interaction objects. Most model-based approaches are featuring this type of mapping. Fewer approaches are exploiting the relationship between objects. In JANUS [1] associations and aggregations are transformed in lists. Items in the list are assigned a dialog window that allows the data entry for each attribute. Pisano, Shirota and Iizava [5] are exploiting relationships in order to generate master-detail forms for database applications.

In order to take full advantage from the information provided by the domain model but also to understand how domain and task model are related, a more detailed analysis is needed for each type of relationship.

In this paper we will discuss two general types of relationships between objects:

- aggregations, which correspond to the general relation "A is part of B" having the cardinality 1:N (one-to-many); for example, a guideline is part-of a section guide;
- associations, which correspond to the general relation "A is associated with B" having the cardinality N:M (many-to-many); for example a guideline respects one or more criteria and a criterion is respected by none or several guidelines.

The analysis will not start from the conceptual data model since the representation is too general and it illustrates early domain modelling results. Rather, we will use the logical data model, which is more complete in that it comprises additional constructs resulting from the logicalisation process. These constructs (i.e. foreign key attributes, tables) are useful for detailed dialog design and they are also...
typical for each kind of relationship.

**Aggregations**

The general case of an aggregation is illustrated in Figure 2, where both graphical and textual representations were included. In our example we will employ a relational model as logical data model.

There are two object types involved: section and item. The name and length of each attribute is not relevant for our purpose. However, there is an additional construct – the foreign key that is pointing to the primary table (section). The dialog model at interaction object level can manipulate this attribute in various ways, in order to satisfy user’s task requirements.

Another relationship type taking a similar form is classification. For example, guidelines are classified as regarding importance level (good practice, confirmed good practice, etc). The mechanism of manipulating the relationship is similar in that a foreign key attribute is used in the first table in order to have access to the data in the second table.

![Figure 2. Aggregation relationship](image)

In the case of aggregation we can distinguish between several tasks the user might want to perform:

a) selecting the higher level entity while editing the attributes of the part entity, for example, selecting the section while editing a guideline (moving a guideline from a section to another i.e. editing the relationship itself) – this could be accomplished using a selecting device like a drop down list;

b) selecting a lower level entity, for example, selecting the a section in order to perform some operations while editing a base and – this could be accomplished using a selecting device like a list and function control objects enabling the desired action;

c) showing higher and lower level entities, for example to display the general guideline and the more specific guidelines (recursive aggregation) – this could be accomplished using a text box and a list but also embedded dialog units showing a master-detail relationship;

d) selecting the higher level entity and showing its content, for example the section and its guidelines – this could be accomplished with two associated selection devices (selection of higher level object updates the content of the list) and additional function control objects, applying to the selected entity in the list.

The diagrams in Figure 3 show these typical situations for a usual data base application. For example, the aggregation relationship between section and base from Figure 1 corresponds to the situation in Figure 3a.

![Figure 3. Typical tasks afforded by an aggregation](image)

However, depending on the user’s task the relationships could be manipulated in several ways thus expanding to many other cases. In each situation, specific interaction objects for function control can be used. For example, in e-commerce applications selection is often combined with the display of related content in a separate window.

We could also consider that the user might want to examine in more detail an object from the list. In this case a “show” button could be added or just a double click on the item could invoke a dialog unit where all attributes of the given object are displayed. Also, more complex user interface constructs could be derived after selecting the first object, like for example master-detail dialog units.

Therefore an analysis of aggregation relationships between domain objects could only be done by taking into account the tasks the user may want to perform. The domain model itself does not provide with sufficient information to derive the user interface since there is a huge potential of possible tasks that could be done. This also demonstrates that the turn-table of the user interface design is the task model and highlights the importance of task decomposition at operational level.

**Associations**

The general case of an association is illustrated in Figure 4. In this case two constructs are resulted from logicalisation:

- foreign keys which are pointing to the data in the primary table
- a new relation (table Item-Association) which holds the explicit associations.

Regarding the associations, the following tasks could be relevant:

a) showing the associated objects, for example showing the more specific guidelines of a general guideline –
this goal could be achieved by using a list box with associated entities and one or more functional control objects for editing the relation (add new items or delete existing ones);

b) changing the current association by removing or adding an existing object to the list – this could be done by using an accumulator (two list boxes and function control IOs for moving objects from one box to another and for removing objects).

Figure 4. Association relationship

These two situations are illustrated in Figure 5.

Again, other situations could be imagined. For example, a shopping cart showing the items bought by a customer in an e-commerce store could be the case in Figure 5 a. In this situation, a “remove” button could be used to edit the association. On the other hand, in this kind of applications associating a new item could be done in an implicit way, by pressing the “buy” button in a different presentation unit.

Figure 5. Typical tasks afforded by an association

This demonstrates that the task context is crucial for the user interface design since it acts as a mediator between the domain model and the user interface model.

STRUCTURING THE INTERFACE

Interaction objects are embodying basic interaction techniques. They are covering both presentation and dialog model at a basic level, which we may term as lexical level. In this respect, interaction objects are the basic constituents of the user interface.

Interaction object groups which have one or more information control objects and one function control object (sometimes two, but the user could choose only one of them at a given time – for example OK vs. Cancel) provide with a first level of structuring the interface. Interaction object groups could be used as basic building blocks for the presentation model in a task-based approach.

More complicated interaction structures like those afforded by relationships in the domain model need a closer look in order to be embedded in useful task structures and thus aid the design of the user interface.

In this section we will analyse the two situations described in Figure 5, but in a more concrete task context. For example, the relation between guidelines and criteria could be edited in a separate dialog unit, where both unit tasks of adding a new item and deleting an existing one are possible. The diagram in Figure 6 shows the mapping between the task and the presentation models.

A different context for editing the association is presented in Figure 7. In this case the user is provided with a list of associated items. He can remove an association or he can add a new one by selecting it from a list. This is a better solution since the user is provided with some information about the existing state of association. On another hand, this requires more screen space in the first dialog unit.

The item collection could also be structured. In this case the user will first select the category and then the item. Actually this is a combination of association and aggregation relationships.

Interaction structures like those presented above could also be used as building blocks for user interface design. Like interaction objects (but despite interaction object groups) they provide with useful information for both presentation and dialog models. Based on a taxonomy of relationships is possible to create pre-defined (typified) dialog units that accept as parameters the data source extracted from the domain model.

This way the code needed to handle the dialog is re-used for similar interaction structures. The data model could be also exploited in order to derive appropriate interaction objects (for example, criterion instead of item) and to edit their properties (list width).

Figure 6. Editing an association in one dialog unit

Figure 7. Editing association in two dialog units
It seems difficult at this moment to generalize this approach for interaction object groups since there is a huge number of potential solutions as regard to the dialog model. However, interaction object groups are useful to derive the presentation in a task-based approach.

In Figure 8 an example of presentation derived from the task model described in Figure 1 is presented. The user can manipulate aggregation relationships as follows. Base name is only displayed. Section could be selected from a drop-down list. If not found, a new section could be created. The user could also select the general guideline and he is provided with a list of more specific guidelines. A classification relationship is used to select the importance level of the guideline from a drop-down list. In this example, the recursive aggregation of guidelines was implemented according to the case 3c in order to provide with maximum of feedback. Seeing both the ancestor and the descendants helps the user to better perceive the underlying domain model. However, the design decision is also dependent on the available screen space.

For each domain object associated with a guideline we have an interaction object group in the interface. Each group has a label, a list box and two buttons. Although not shown in Figure 1, the decomposition for the unit tasks “edit ref”, “edit ex” and “edit criteria” takes the form presented in Figure 7. This means it enables removing associations in the main dialog unit and adding new objects from another dialog unit.

Structuring the presentation in a task-based approach is a two-step process.

First, interaction object groups are derived from the operational task model. Except for the cases of pre-defined capabilities, like those discussed in this section, and built-in capabilities (like dialog units for error management or file/print management) this work takes place in a virtual space. The process is bottom-up and it produces interaction object groups as a basic structured layer.

Second, the task model is exploited in a top-down approach by taking into consideration screen space limitations and allocation strategies.

Task oriented strategies require to reconsider the task model. In this case, the virtual space is first transformed into a structured space in order to satisfy the allocation strategy. Then temporal relations could be analyzed in order to fit the available screen space.

Platform oriented strategy lead to minimal vs. maximal allocations when the variation is mainly in the available screen space. This case fits the previous one.

Content oriented strategy is typically used for web sites organizing their content in web pages. The structure of the interface takes precedence over the task model and requires to build a task model in a constraint presentation.

Structuring the interface in dialog units leads to add navigation control. This means navigational interaction objects. It is a good practice to review the task model at operational level and to add corresponding basic tasks.

Then is possible to check the task model structure for proof and to avoid design errors like unreachable dialog units or unreachable interaction objects.

A synthetic presentation of our task-based approach is done in Table 3 where three types of mappings are illustrated.

<table>
<thead>
<tr>
<th>DOMAIN MODEL</th>
<th>TASK MODEL</th>
<th>INTERFACE MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram" /></td>
<td><img src="image2.png" alt="Diagram" /></td>
<td><img src="image3.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Table 3. User interface derivation from domain and task model

First row shows how interaction object groups are derived from task and domain models. Domain objects are providing with attribute information from which interaction objects for information control are derived.

Grouping of interaction objects is done either according to the semantics of the domain model (rarely, for objects having many and / or compound attributes) or around function control IOs (often).

Static IOs denoting the group meaning are also added in order to increase user guidance. For example, IO groups...
could take the name of the unit task they support. Second row represents the derivation of more complex unit task structures by considering relationships between domain objects. In turn, these structures are further exploited in order to derive interaction object groups and dialog units. The last row is incomplete in that it illustrates allocation of dialog units only by considering the task model. In a task-based approach early task modeling is assumed to integrate systems functions. Therefore they are not further exploited during operational task modeling.

CONCLUSION
In this paper we proposed a framework for task modeling based on a clear distinction between functional goals and operational goals. Operational goals are formed in order to delegate the task to the computer. In this respect they are device dependent and show how the user is actually operating the interface.

On another hand, operational task structures describe how users are manipulating domain objects. Interaction object groups are formed by grouping information control objects around a function control object that is designed according to requirements coming from the task model. More complex task structures at operational level are afforded by relationships between domain objects. In this case not only interaction object groups could be derived but also dialog units. Thus bigger building blocks having both presentation and dialog parts could be derived.

The analysis of various tasks afforded by relationships shows some inherent limits of the automate generation of user interfaces. However, this does not undermine a computer-aided approach to user interface design. Rather, it suggests to create new tools able to handle with at least domain and task models at a time. Then a XML specification of the operational task model could be produced and further used as input for a user interface generator. Such tools could be integrated in the architecture of computer-aided design assistants like described in [3].

An issue with task editors is that they are mainly conceived to handle task modeling at functional level. Although CTT editor distinguishes between different types of basic tasks, like application, interaction and user, it does not provide with means for aggregated task structures. This is seriously reducing the capability of developing layered task models and effectively (i.e. in a computer-aided fashion) exploiting the task model for user interface design.

Another suggestion arising from this work is to look for complex interaction techniques (interaction patterns) that are suitable to accomplish the user’s goals and embed them into customizable interface (presentation and dialog) structures.

ACKNOWLEDGMENTS
This work has been supported by an advanced NATO fellowship at Université catholique de Louvain in Louvain-la-Neuve, Belgium.

REFERENCES
4. Paternò, F. Model-based design and evaluation of interactive applications. Springer Verlag, 1999