Reusable Structures in Task Models

I.M. Breedvelt-Schouten[+], F.D. Paternò[*], C.A. Severijns[+]

(+)Baan Research, Baan Company N.V.
P.O. Box 250
6710 BG, Ede, The Netherlands
{ibreedvelt, cseverijns}@baan.nl

(*)CNUCE - C.N.R.
Via Santa Maria 36
56126 Pisa, Italy
f.paterno@cnuce.cnr.it

Abstract

Task Analysis is a well-known approach which has been used to analyse the design of existing applications. Recently there is an increasing interest to apply this type of technique to the design and development of new applications, too. However if user interface designers want to apply task modelling on a larger scale, to industrial size case studies, the possibility of reuse is useful for saving time and effort. In this paper we present an approach for designing reusable structures in task models that allows designers to focus more clearly on the needs of the user and that speeds up the application design.

Keywords: Task Models, Reuse, Industrial Applications of Formal Methods

Introduction

In this paper we present the first results of a co-operation between the User-centred Design Group at CNUCE and Baan Company on new methods for the design of user interfaces. In this research we have considered ConcurTaskTrees [11], a diagrammatic notation to describe hierarchical task models, which allows designers to enrich the traditional functional specification by including the user's view of system functionality. And we have investigated its use for new applications of interest for Baan Company.

There are various reasons for choosing a task-based approach. This type of approach allows designers to focus on high-level semantic oriented aspects and developers to obtain system functionalities which reflect the user's view of these. Thus, while interacting with the user interface of an application designed with a task-based approach, the user will easily understand how to use the system. This is because: i) the user interface provides actions which can be immediately mapped to logical actions, ii) the temporal relationships between the actions in the user interface reflect those defined in the task models, and iii) all implementation aspects which are less comprehensible for the user are hidden. Another aspect is that task modifications can be more easily implemented: in Interactive Systems designed by task-driven approaches it is easy to locate which part of the system should be changed, when support of some tasks is re-

moved, added or modified, because it is possible to create a direct correspondence between tasks and the software components used to perform them.

Baan Company is one of the major vendors of solutions for Enterprise Resource Planning. These solutions are implemented by customising a large generic software package, that is developed by Baan Company itself and consists of approximately 5×10^3 applications ranging in functionality from simple editing of data to complicated planning tools. In order to manage this complexity it became soon important to achieve a relevant issue in the design and development of user interfaces: the possibility to reuse good design solutions of recurrent problems in dialogue specifications to save time and effort.

ConcurTaskTrees is a notation for specifying task models which has been developed to overcome limitations of notations previously used to design user interfaces, such as UAN [8]. Its main purpose is to be an easy-to-use notation that can support the design of real industrial applications, which usually means applications with medium-large dimensions. It can solve the problem of many notations, such as Interface Object Graph [5], which, while they are effective for simple limited examples, show low scalability for specifications of real case studies, thus soon becoming difficult to interpret.

The ConcurTaskTrees notation provides a graphical representation in a tree-like form of a hierarchical decomposition of tasks. A set of operators, mainly taken from the LOTOS [9] notation, is used to indicate the temporal relationships among tasks such as iteration, sequentiality, concurrency, disabling, and recursion. We used it to design various solutions for different problems. After performing several exercises we have realised that an interesting element is the possibility to identify specific task patterns in the tree-like structures describing task models which can be reused across different applications which, in some points, raise the same requirements.

One further possible advantage of task-driven user interface development, where a direct correspondence is created between software components and tasks to support, is that this correspondence can be exploited for software reuse purposes as well. This possibility has not yet been investigated in current task-driven proposals. In this paper we do not discuss this aspect since we want to focus on the possibility of reusable structures in task models for design purposes. We leave the topic of reusable task structures for supporting the implementation phase for a further paper because it requires different considerations.

In this paper, after a short discussion of related work, we introduce the ConcurTaskTrees notation which we use for building task models and we introduce the concepts of reusability relevant to the approach presented. Then we introduce examples of possible reusable task structures and one application designed with the support of a task template. We conclude with some remarks and indications for future work.

Related Works

A different task-oriented approach is proposed by Wilson et al. with Adept [17]. In their proposal they address the design of a task model, an abstract architectural model, and a related implementation. However, this is obtained mainly by the skill of the designer with limited support from predefined rules incorporated in an automatic tool.

In model-based approaches to user interface design and development there are some proposals (such as Trident [2] and Mastermind [15]) which consider task specification as an abstract model. However these proposals usually do not consider an explicit abstract architectural level moving directly from the task level to the implementation level. Similarly they do not consider reuse aspects in their approach.

UAN supports hierarchical specifications and has operators to express temporal relationships among tasks. However we found it has some limitations: first, it has a textual specification which makes it difficult to read and interpret (for example to find cross-references among tasks). It also provides limited support for deriving a software architecture, as its main purpose is to specify only the externally perceivable behaviour of the user interface by associating tables indicating above all user actions and system responses. As a consequence, it is oriented to provide low-level specifications with many details and is not very well suited for applying reuse.

In the field of formal methods for human-computer interaction there are approaches which consider task abstraction aspects [6, 13]. In these approaches the goal is usually oriented to analyse the dialogue of existing systems rather than giving the possibility to build an architectural specification which can be used for prototyping purposes as well. Other approaches to task-driven design are in [18, 19], but they mainly focus on presentation-related aspects rather than analysing the dialogue of Interactive Systems.

The analysis of current task-oriented approaches highlights the lack of proposals supporting design reuse. This is a relevant issue given the increasing complexity of Interactive Systems. Reuse has already been recognised as such in other areas of application design, for example, in object-orientation [7].

ConcurTaskTrees

It is becoming increasingly common for the various specialists (developers, designers, psychologists, application domain experts) involved in the design process to discuss the tasks that the system should support. To this end it is very important to have notations to develop task specifications so that:

- they are easy to understand and use, thus improving communication among people discussing the design;
- they are able to structure the large sized specifications which are developed in industrial applications;
- their semantics are precisely defined to avoid ambiguities in the communication.

When choosing the notation we found that some important features have to be supported:

- hierarchical logical structures which were introduced by GOMS have proved to be a useful way to represent task models because they allow designers to reason about the design at different abstraction levels and they support the refinement design process better;
- to be able to express a wide variety of temporal relationships since modern user interfaces are characterised by highly interactive behaviours in multimedia environments;

- to handle the complexity of task models for industrial applications, it is thus important to be able to express relevant relationships precisely and to have information on more detailed aspects in an interactive way.

A task defines how the user can reach a goal in a specific application domain. The goal is a desired modification of the state of a system or a query to it.

We can identify four types of tasks depending on their performance allocation:

- *user tasks* are completely performed by the user, they require cognitive or physical activities without interacting with the system. One example is when the user reads a list of flights satisfying some constraints and decides to select one of them for his/her journey. More generally, we say that user tasks are associated with some processing performed by the user on information received from the environment.
- *application tasks* are completely executed by the application. They receive information from the functional core and they can supply information to the user. They are activated by the application itself. For example, compiling a program and sending messages when some errors are detected, or receiving network messages and displaying them.
- *interaction tasks* are performed by user interactions with the application. These interactions are activated by the user. Examples are editing a diagram or formulating a query to a data base.
- *abstract tasks* are tasks which require complex actions, though how to allocate their performance has not yet been decided.

In the task specification the types of tasks are presented either by different icons or different geometric shapes as in Figure 1.

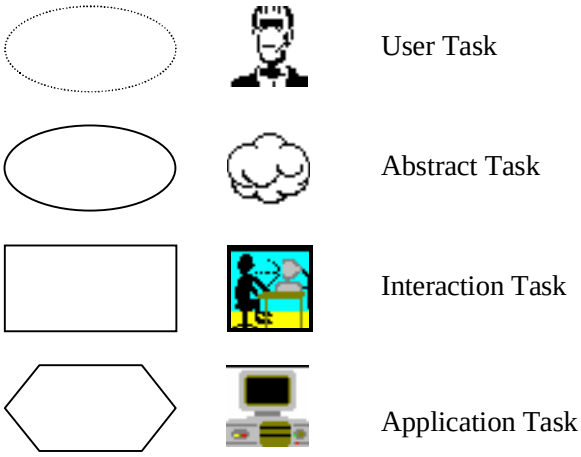

Figure 1: The two possible presentations of Task types.

The temporal relationships among tasks are expressed by extending the operators of LOTOS, which is a concurrent notation. This allows us to describe concurrent tasks, unlike the GOMS proposal which uses a hierarchical task decomposition only consisting of sequential tasks. It is possible to specify both synchronous and asynchronous communication among tasks. Tree-like structures combined with operators to indicate temporal relationships among tasks at the same level allow designers to specify more complex behaviours than those associated with basic LOTOS operators.

The operators that we use to describe the temporal relationships are:

T1 ||| T2 *interleaving*: the actions of the two tasks can be performed in any order;

T1 |[]| T2 *synchronisation*: the two tasks have to synchronise on some actions in order to exchange information;

T1 >> T2 *enabling*: when the first task is terminated then the second task is activated;

T1 []>> T2 *enabling with information passing*: when task T1 terminates it provides some value for task T2 besides activating it;

T1 [] T2 *choice*: when it is possible to choose between two tasks to perform;

T1 [> T2 *disabling/deactivation*: when one action from the second task occurs the first task is deactivated;

T1 [][> T2 *disabling/deactivation with information passing*: when one action from the second task occurs the first task is deactivated while passing some value;

T1* *iteration*: the task can be iterated many times;

T1(n) *finite iteration*: the task is performed n times.

[T1] *optional task*: the performance of this task is optional. It is not mandatory to perform this task

Recursion is obtained by allowing the use of a task within its own specification. This means that in the task subtree, which defines a given task, we can find again its occurence.

Reusability

Reusability is an important issue in every stage of application development. Object-orientation is a well-known approach for analysing, designing and implementing applications which simplifies the reuse in each of these three phases of the software development process, [1, 14]. Many reusable structures or *patterns* that occur in object-oriented systems, have been documented [7, 3]. By using these patterns developers can build more easily upon the work of others. If similar structures are available for task models, this would enhance the development of task models, too.

We distinguish two possible types of reuse: design reuse and software reuse. While software reuse indicates the possibility to use the same pieces of implementation in different contexts, design reuse means that we can identify pieces of task specifications which can be used in various applications. Thus, whenever designers realise that the problem they are considering is similar to one, which has already been found, and solved, then they can immediately reuse the solution previously developed. This can be done by applying the related task specification which is represented by a task pattern in a ConcurTaskTrees specification.

The hierarchical structure of this specification has two advantages:

i) it provides a large range of granularity allowing large and small task structures to be reused,

ii) it enables reusable task structures to be defined at both a low and a high semantic level.

Furthermore, it is possible to associate the tasks defining the template with a set of interactors, which define the architecture of the application [12]. This correspondence can be used to ease and speed-up the development process.

An additional advantage of using task patterns is that they help to make the task specification easier to read and interpret since it is possible to indicate their names if they occur in the lowest part of the task tree rather than specifying them completely. This makes the specification more compact and legible since some repetitive small specifications, which do not add new conceptual aspects, can be indicated very briefly. Examples are the modal dialogues controlling the printing of information or the closing of a session with the optional possibility of applying the modifications performed.

However, defining templates is not an easy task. There are at least two basic problems:

i) to identify the characteristic situations which can occur in different applications,

ii) to identify the information that should be used to define instances of these dialogues patterns.

Reusable Structures in Task Models

While analysing and (re-)designing parts of the Baan software, we found several templates in our models. In this section we will discuss four of the most common templates that we found.

i) A *Multi-values Input Task* appears in every application in which the user edits various values until (s)he decides to submit them. For this behaviour we found that the recurring structure is the one shown in Figure 2. We first have a distinction between the Edit values task and the Submit task. The second task must be able to disable the editing activities. Editing is performed by the interleaving of the editing of the various attributes. Each attribute editing is iterative because the user can decide to change the value before submitting it. As a result, all sub-tasks can be performed any number of times.

   An example of an instance of the *Multi-value Input* task is the description of an input to a database which allows the user to provide, for example, name, surname and telephone number of each element.

   We can note that this task template is independent of the number of attributes to be edited because this element does not change its global behaviour.

ii) A *Search Task* is a very common task template, because it allows a user to search

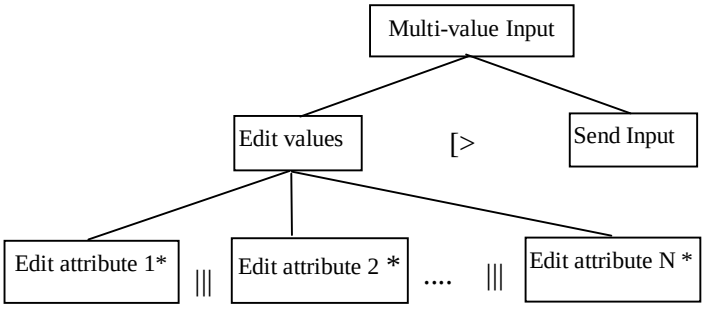

Figure 2: A Multi-value Input template

for specific information. Searching is not only «locating» data by using one query statement, it is also a navigational way of searching in which the next query is based upon the results of the previous one; this is called refinement. The only aspect that changes during refinement is the information used by the user to decide how to specify the next query.

The basic semantics of this task (see Figure 3) are: the user indicates what data to search for via a query statement entry, the query manager is activated via submitting the query to search for the matching data, the results are shown and finally the user is able to refine his/her query results. These results will be used as new input for the next search. More specifically, the user can decide, depending on the result of the previous query, to enter a new formulation of a query which probably will provide the desired information without additional disturbing elements. To describe this activity we introduced explicitly an additional user task (*Decide re-*

finement) which receives information from an application task (*Show data set ...*) and produces input for the next interaction task (*Refine query*) which requires the same user interactions as the *Define Query* task. As you can note the application tasks used are the same to provide both the result of an initial query and the query result deriving from a user refinement.

This search task can be applied in many applications where searching is needed, like search engines, database query applications and file managers.

iii) An *Evaluation Task* is a more complicated template (see Figure 4). It consists of two activities, selection of the data to be evaluated and the evaluation itself, which can be repeated until the user decides to stop. The evaluation consists of five sub-tasks: first the user selects the evaluation type. Next the required data structures for this type of evaluation are created by the application. Then the user can edit the parameters that are needed during the evaluation until (s)he decides to start the calculations required for the evaluation. Finally, the results of the evaluation are shown. Note, that the tasks Select Data and Evaluate Data are syn-

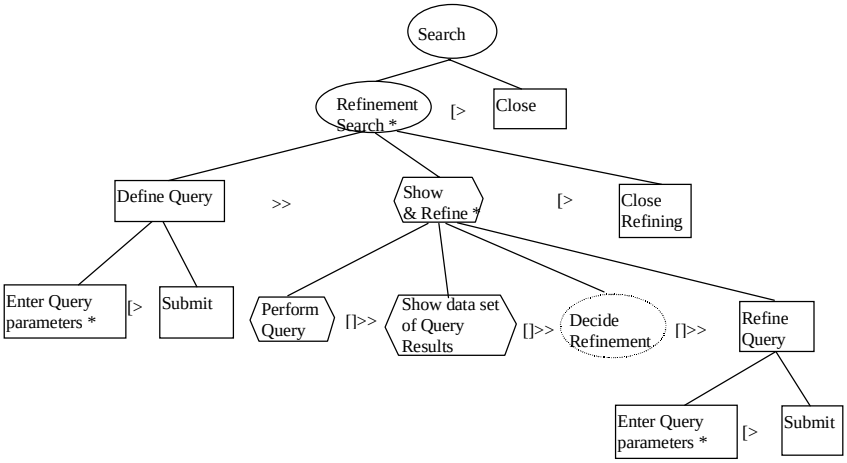

Figure 3: A Search Template

chronised, because the selection is directly influencing the types of evaluation that can be applied. An example is an information system which gives the user the possibility to get information about houses and evaluations about them (pricing, history, mortgage, sizes, etc.). The user can select a set of houses and concurrently the evaluation type. After selecting the evaluation type, it is possible to specify the parameters, like mortgage-rates, room-sizes, and taxes. Finally, results are shown. In some cases, depending on the type of evaluation requested, some houses cannot be selected, because the related information is not available.

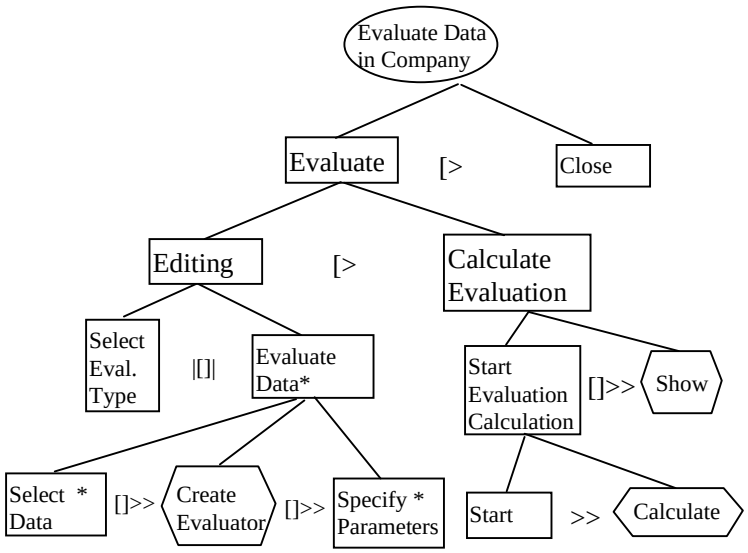

Figure 4: An Evaluation Task template

iv) A *Recursive Activation Task* template captures the recurrent situation in many dynamic modern user interfaces which makes available an initial task whose main purpose is to allow the user to activate new instances of another task. An example can be a word processor which, whenever a specific interaction technique is selected, allows the editing of a new file other than maintaining the possibility to edit files previously opened. A generic example of use of this template is shown in Figure 5, where the double occurrence of the abstract Handle set of Objects task indicates this type of recursion. In the example we have an application for handling a set of objects. If it is not closed, it allows the user to select and/or delete various objects until the Start Object task is performed, which means that the presentation for editing the selected Object is activated. The recursion ensures that the Handle Objects task is available again. As a result of this recursion, the user can create several instances of the Handle One Object task by performing the Handle Objects task.

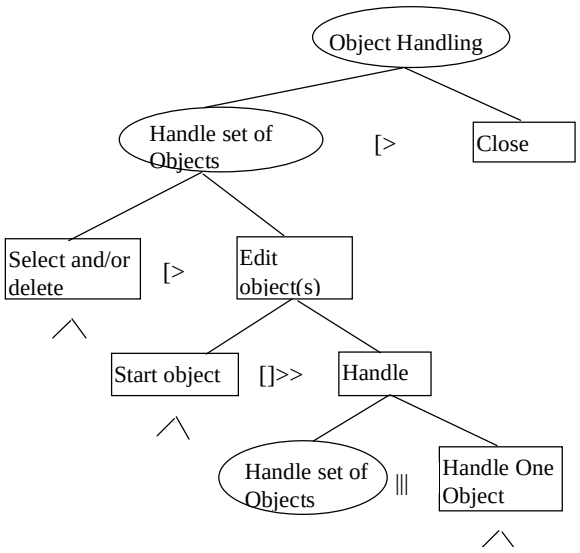

Figure 5: A Recursive Activation Task template

As illustrated by these examples, these templates are similar to patterns in object-oriented analysis and design [7]: they give a problem definition, describe a solution to this problem, and they can occur regularly in many applications.

Using Templates to Design an Application

The goal of most commercial companies is to sell as many products as possible. The activities of many departments within a company are aimed at achieving this goal. For example, the main activity of the sales department is to accept orders from customers and to provide other departments with the information required to deliver the products. Nowadays many companies support their activities, including those of their sales department, with an automated system for Enterprise Resource Planning (ERP). Amongst others, such an ERP system maintains information on the sales orders that have been and are being processed by the company. Sales employees have the task to start the processing of an order by entering new sales order information into the ERP system. They can also change existing sales order information when needed. In this section we will show how we can obtain an application for this task using templates, in this case the Recursive Activation Task and the Multi-Value Input templates.

The starting-point of the edit sales order task is the set of sales orders. The employee can add a new sales order, or delete or open an existing one. Once a sales order has been created (opened), the following data can be entered (modified): the order-specific data, the data related to the customer and the products, each product on its own order line. Regularly, a sales employee needs to access information on several sales orders simultaneously, e.g. while the sales employee is entering a sales order for customer A, customer B calls and wants a change in his/her sales order. This implies that a sales employee should be allowed to activate multiple instances of the sales order entry task. Therefore, we conclude that we can use the Recursive Activation Task Template as a starting point to construct our "Order Handling" task model. By replacing the word "Object" in the template by "Order" we obtain the part of the task model of Figure 6 shown in grey. The left part of the task model implies that the user can select orders until s/he decides to interrupt this selection by deleting or starting the selected orders or by starting a total new order.

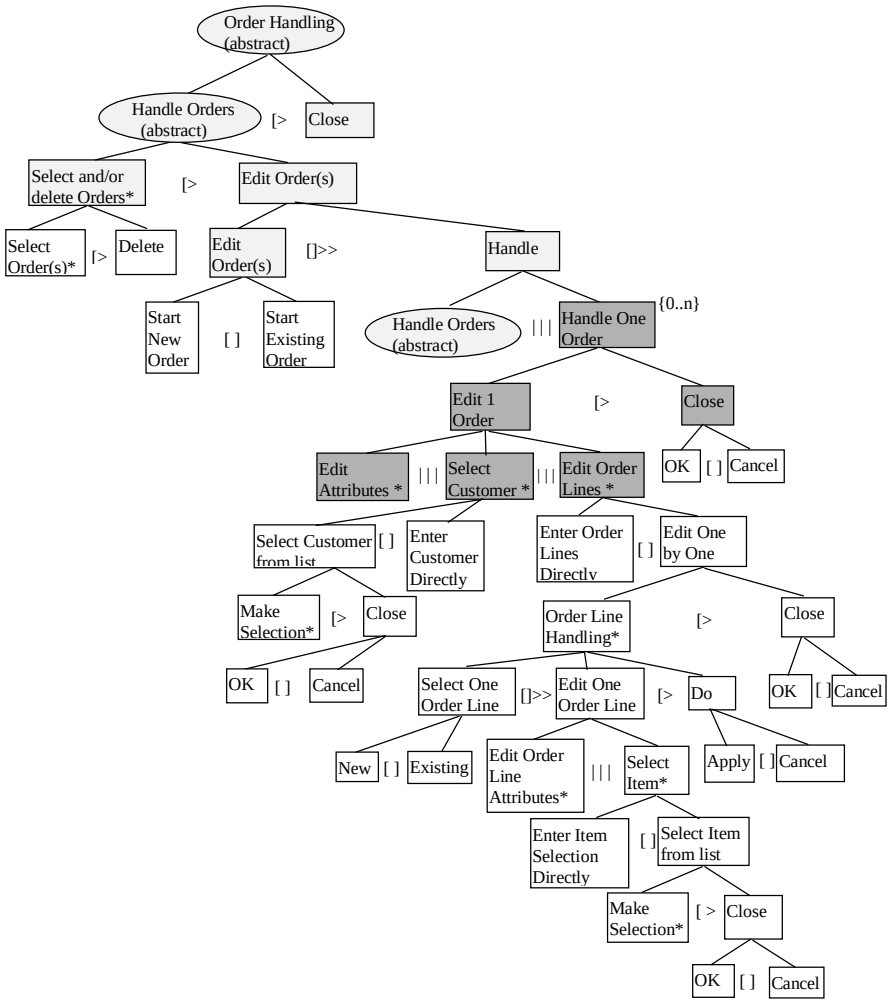

Figure 6: The task model that results from applying the Recursive Activation Task template to Orders and extending it with application specific tasks.

We still need to extend this model for specific actions on a sales order: entering order specific information (e.g. an order date), selecting the customer, entering the order lines and selecting the products for the order lines. For this purpose we can use the Multi-value Input task template.

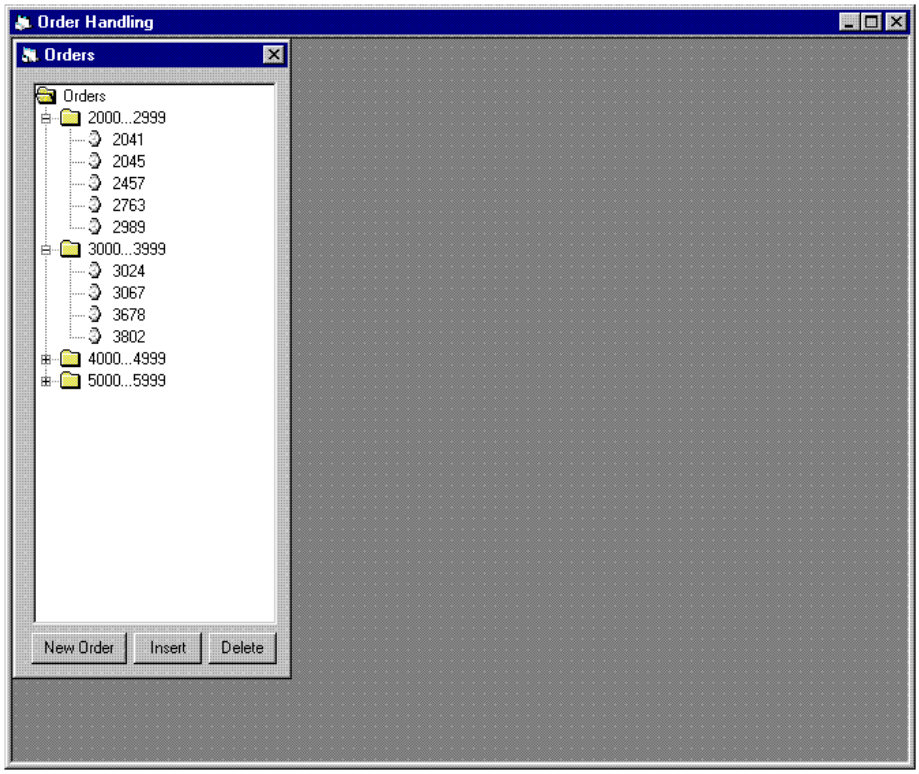

Figure 7: The set of orders, from which a user can select the orders.

The activation of the Edit One Order task is envisioned in the user interface by Figure 7. Note that the sales orders are indicated via numbers, which is a common practice in many companies. The order lines can be edited directly in the table of the Order window as shown in Figure 8, or the user can open another dialogue to edit the order lines in a separate dialogue for order lines (see Figure 9). The difference between 'enter selection directly' and 'select from list' is introduced to indicate that the user respectively knows the data to select by heart or needs extra support by selecting correct data from an existing list/set.

As a concluding remark we note that we can use the Recursive Activation Task Template in a similar manner to manage some other data types, such as editing production orders, customers, and inventories.

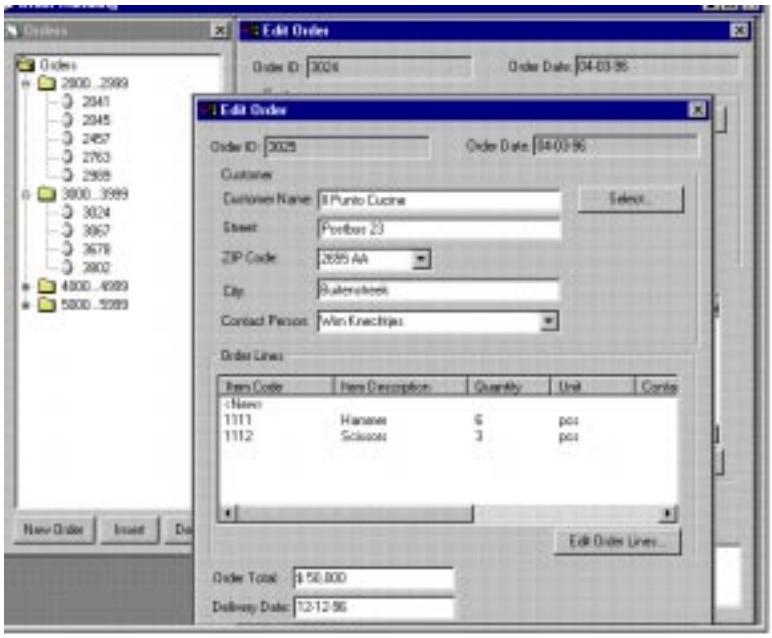

Figure 8: Two Edit Order tasks are open.

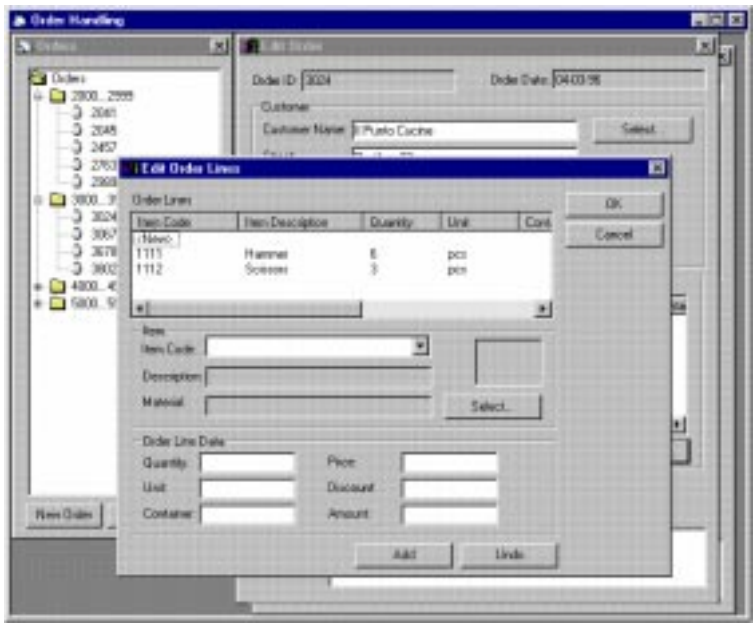

Figure 9: The Edit Order Lines dialogue in an Edit Order Task.

Conclusions

In this paper we have shown how it is possible to recognise and define reusable structures in task models. These reusable structures enable and encourage designers to focus more on the needs of users by providing high-level, semantics-oriented elements.

The need for them was raised by work on using task models to design industrial applications in the business area: these are usually large applications where similar problems occur often in different parts of the design. We found reusable task structures useful to speed-up the process of building task models for discussing design solutions because they incorporate immediate solutions for recurrent problems.

Currently we are working on extending the set of reusable structures in task models. We consider the set of templates presented here as a good starting-point for further investigation of reusability in task models. We plan to support the use of templates in a semi-automatic tool. In addition, we are investigating the possibility of obtaining reusable interactor networks: the software components, associated with the specification of tasks templates which can be used to describe similar situations in different contexts, in order to directly include them in different software applications. Thus, the software developer has the advantage of using a high-level, more immediate to understand, specification of software architectures which can be reused.

References

1. G.Booch, «Object-oriented Analysis and Design», Benjamin/Cummings Publ. Comp., 2nd ed., 1991, pg. 327.
2. F.Bodart, A.Hennerbert, J.Leheureux, J.Vanderdonckt, "A Model-based approach to Presentation: A Continuum from Task Analysis to Prototype", in Proceedings DSV-IS'94, Springer Verlag, pp.77-94.
3. F.Buschmann, R.Meunier, H.Rohnert, P.Sommerlad, M.Stal, «A System of Patterns: Pattern-oriented Software Architecture», Wiley, 1996.
4. S.Card, T.Moran, A.Newell. "The Psychology of Human-Computer Interaction", Lawrence Erlbaum, Hillsdale, N.J., 1983.
5. D.Carr, «Specification of Interface Interaction Objects», Proceedings ACM CHI'94, pp.372-377.
6. R.Fields, P.Wright, M.Harrison, «A Task-centred approach to analysing human error tolerance requirements». Proceedings Requirements Engineering'95, pp.18-26.
7. E.Gamma, R.Helm, R.Johnson, J.Vlissides, «Design Patterns: Elements of Reusable Object-Oriented Software», Addison-Wesley, 1995.
8. R.Hartson, P.Gray, «Temporal Aspects of Tasks in the User Action Notation», Human Computer Interaction, Vol.7, pp.1-45.
9. ISO - Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on temporal Ordering of Observational Behaviour. ISO/IS 8807, ISO Central Secretariat.
10. I.Jacobson, "Object-oriented Software Engineering - A use case driven approach", Addison Wesley, 1992.

11. F.Paterno', C.Mancini, S.Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models", Proceedings Interact'97, Chapmann&Hall, Sydney.
12. F.Paterno', C.Mancini, S.Meniconi, "Understanding Tasks and Software Architecture Relationships", CNUCE Internal Report, December 1996.
13. P.Palanque, R.Bastide, Verification of an Interactive Software by Analysis of its Formal Specification, Proceedings INTERACT'95, Lillehammer, June'95.
14. J.Rumbaugh, M.Blaha, W.Premerlani, F. Eddy, W. Lorensen, «Object-oriented Modeling and Design», Prentice-Hall, 1991, pg. 282.
15. P.Szekely, P.Sukaviriya, P.Castells, J.Muthukumarasamy, E.Salcher, "Declarative Interface Models for User Interface Construction Tools: the Mastermind Project", Proceedings EHCI'95, Chapmann&Hall, August '95.
16. D.Schmidt, M.Fayad, R.Johnson, «Software Patterns», Communications of ACM, October 1996, pp.36-40.
17. S.Wilson, P.Johnson, C.Kelly, J.Cunningham, P.Markopoulos, «Beyond Hacking: a Model-based Approach to User Interface Design, Proceedings HCI'93, Cambridge University Press.
18. A.Sutcliffe, P.Faraday, "Designing Presentation in Multimedia Interfaces", Proceedings CHI'94, pp.92-98.
19. A.Sears, AIDE: A step toward metric-based user interface development tools. Proceedings of UIST'95. ACM Press, pp.101-110.