24

ConcurTaskTrees: An Engineered Notation for Task Models

Fabio Paternò
ISTI-C.N.R.

Task models represent the intersection between user interface design and more engineering approaches by providing designers with a means of representing and manipulating a formal abstraction of activities that should be performed to reach user goals. For this purpose they need to be represented with notations able to capture all the relevant aspects. In this Chapter, the ConcurTaskTrees notation is described and discussed with particular attention to recent evolutions stimulated by experiences with its use and the need to address new challenges.

24.1 INTRODUCTION

One universally agreed-on design principle for obtaining usable interactive systems is this: *Focus on the users and their tasks*. Indeed, of the relevant models in the human-computer interaction (HCI) field, task models play an important role because they represent the logical activities that should support users in reaching their goals (Paternò, 1999). Thus, knowing the tasks necessary to goal attainment is fundamental to the design process.

Although task models have long been considered in HCI, only recently have user interface developers and designers realized their importance and the need for engineering approaches to task models to better obtain effective and consistent solutions. The need for modeling is most acutely felt when the design aims to support system implementation as well. If we gave developers only informal representations such as scenarios (chap. 5) or paper mock-ups (chap. 1), they would have to make many design decisions on their own, likely without the necessary background, to obtain a complete interactive system.

In order to be meaningful, the task model of a new application should be developed through an interdisciplinary collaborative effort involving the various relevant viewpoints. If experts from the various fields contribute appropriately, the end result will be a user interface that can effectively support the desired activities. This means that the user task model (how users think that the activities should be performed) shows a close correspondence to the system task model (how the application assumes that activities are performed).

There are many reasons for developing task models. In some cases, the task model of an existing system is created in order to better understand the underlying design and analyze its potential limitations and how to overcome them. In other cases, designers create the task model of a new application yet to be developed. In this case, the purpose is to indicate how activities should be performed in order to obtain a new, usable system that is supported by some new technology.

Task models can be represented at various levels of abstraction (chap. 1). When designers want to specify only requirements regarding how activities should be performed, they consider only the main high-level tasks. On the other hand, when designers aim to provide precise design indications, then the activities are represented at a smaller granularity, including aspects related to the dialogue model of a user interface (which defines how system and user actions can be sequenced).

The subject of a task model can be either an entire application or one of its parts. The application can be either a complete, running interactive system or a prototype under development. The larger the set of functionalities considered, the more difficult the modeling work. Tools such as CTTE (publicly available at http://giove.cnuce.cnr.it/ctte.html) open up the possibility of modeling entire applications, but in the majority of cases what designers wish to do is to model some subsets in order to analyze them and identify potential design options and better solutions.

Although task models have long been considered in human-computer interaction, only recently have user interface developers and designers realized their importance and the need for engineering approaches to task models to better obtain effective and consistent solutions. An engineering approach should address at least four main issues (these are also discussed in chap. 22):

- The availability of flexible and expressive notations able to describe clearly the possible activities. It is important that these notations are sufficiently powerful to describe interactive and dynamic behaviors. Such notations should be readable so that they can also be interpreted by people with little formal background.
- The need for systematic methods to support the specification, analysis, and use of task models in order to facilitate their development and support designers in using them to design and evaluate user interfaces. Often even designers who do task analysis and develop a model do not use it for the detailed design of the user interface because of the lack of structured methods that provide rules and suggestions for applying the information in the task model to the concrete design. Structured mothods can also be incorporated in tools aiming to support interactive designers.
- Support for the reuse of good design solutions to problems that occur across many applications. This is especially relevant in an industrial context, where developers often have to design applications that address similar problems and thus could benefit from design solutions structured and documented in such a way as to support easy reuse and tailoring in different applications.
- The availability of automatic tools to support the various phases of the design cycle. The developers of these tools should pay attention to user interface aspects so that the tools have intuitive representations and provide information useful for the logical activities of designers.

Task models can be useful both in the construction and the use of an interactive system. They can aid designers and developers by supporting high-level, structured approaches that allow integration of both functional and interactional aspects. They can aid end users by supporting the generation of more understandable systems.

This chapter first presents the basic concepts on which ConcurTaskTrees (Paternò, 1999) has been developed. The chapter not only provides an introduction of the notation but also discusses some issues that have been revealed to be important during its development and application: the relationships between task models and more informal representations, how to use task models to support the design of applications accessible through multiple devices, how to integrate them with standard software engineering methods such as UML (see chaps. 7, 11, 12, 19, 22, 23 and 27), how to represent task models of multiuser applications, and what tool support they need in order to ease and broaden their analysis by designers. The chapter also touches upon the experience of using ConcurTaskTrees (CTT) in a number of projects.

24.2 BASIC CONCEPTS

Of the relevant models, task models play a particularly important role because they indicate the logical activities that an application should support to reach user goals.

A goal is either a desired modification of a state or an inquiry to obtain information on the current state. Each task can be associated with a goal, which is the state change caused by its performance. Each goal can be associated with one or multiple tasks. That is, there can be different tasks that achieve the same goal.

Task descriptions can range from a very high level of abstraction (e.g., deciding a strategy for solving a problem) to a concrete, action-oriented level (e.g., selecting a printer). Basic tasks are elementary tasks that cannot be further decomposed because they do not contain any control element.

For example, making a flight reservation is a task that requires a state modification (adding a new reservation in the flight database) whereas querying the available flights from Pisa to London is a task that just requires an inquiry of the current state of the application. Making a reservation is a task that can be decomposed into lower level tasks such as specifying departure and arrival airports, departure and arrival times, seat preferences, and so on.

It is better to distinguish between task analysis and task modeling. Whereas the purpose of task analysis is to understand what tasks should be supported and what their related attributes are, the aim of task modeling is to identify more precisely the relationships among such tasks. The need for modeling is most acutely felt when the design aims to support system implementation as well. If developers were given only a set of scenarios, they would have to make many design decisions on their own in order to obtain a complete system. This is why UML has widely been adopted in software engineering: Its purpose is to provide a set of related representations (ranging from use cases to class diagrams) to support designers and developers across the various phases of the design cycle.

There are various types of task models that can be considered:

- *The task model of an existing system.* The purpose of this model is to describe how activities should be performed in an existing, concrete system in order to understand its limitations, problems, features, and so on.
- *The task model of an envisioned system.* In this case the goal is to define how a new system, often supported by some new technology, should aid in the achievement of users' goals or improvement of some aspect of existing systems.
- *The user task model.* This model indicates how the user actually thinks that tasks should be performed. Many usability problems derive from a lack of direct correspondence between the user and the system task model (Norman, 1986; chap. 18 and 19).

The design cycle in Fig. 24.1 shows one way of viewing the various task models. Often people start with the task model of the existing system and have requirements for designing a

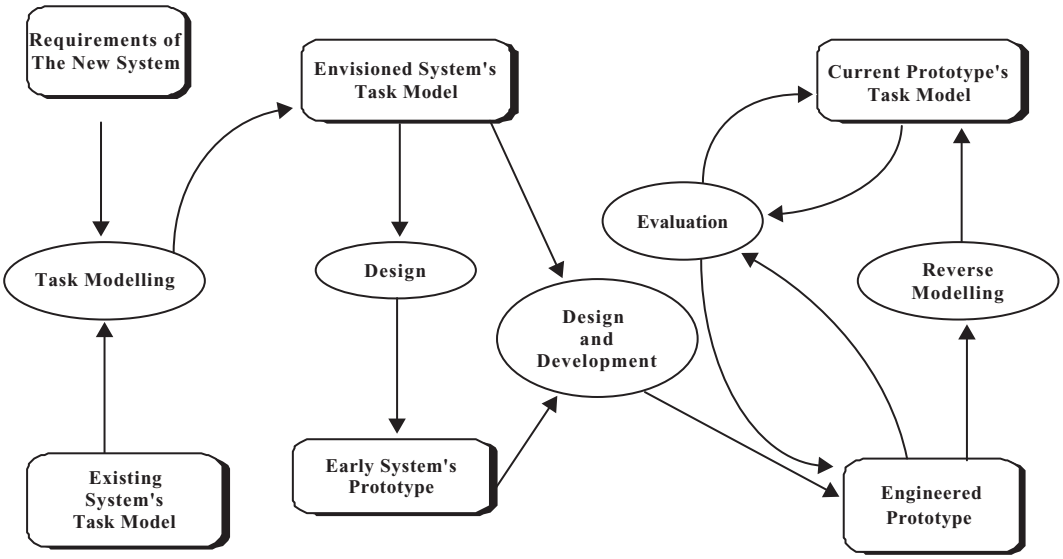


FIG. 24.1. Use of task models in the design cycle.

new system (e.g., the use of some new technology). The modeling activity is used to create a task model of the envisioned system—a model that can act as the starting point for the design of the concrete user interface. In order to have a first discussion regarding the preliminary design ideas, designers often use early, rapid prototypes produced using paper, mock-ups, and similar things. The task model and the early prototypes are the input for the actual design and development of the engineered prototype that will then become the delivered system. Constraints in the development of the system can result in a system whose task model is different from the initial model. So, if task models are used to support the evaluation of the system, then a kind of reverse engineering has to be done in order to identify the actual system task model that needs to be evaluated.

More generally, task models can be used for many purposes:

- *To improve understanding of the application domain.* The modeling exercise requires designers to clarify what tasks should be supported, their mutual relationships, their constraints, and so on. In an air traffic control project, we developed task models together with designers who had been working for a long time in this domain without this technique (Paternò, Santoro, & Sabbatino, 2000). The designers admitted that this exercise raised a number of issues that they never considered before.
- *To record the results of interdisciplinary discussion.* Because task models are logical descriptions of the activities to support, they should take into account the viewpoint of all the relevant stakeholders (designers, developers, end users, domain experts, the client, etc.).
- *To support effective design.* If a direct correspondence is created between the tasks and the user interface, users can accomplish their goals effectively and satisfactorily.
- *To support usability evaluation.* Task models can be used in evaluation in various ways. For example, they can be used to predict time performance, as in the GOMS approach (John & Kieras, 1996; chap. 4) and to analyse the user's actual behavior, as in WebRemUSINE (Paganelli & Paternò, 2002).

- *To support the user during a session.* They can be used at run time to automatically generate task-oriented help, including explanations of how to enable disabled actions and perform them.
- *To provide documentation on how to use a system to accomplish the desired tasks.*

24.3 INFORMAL DESCRIPTIONS VERSUS TASK MODELING

One issue often discussed is whether to use structured abstractions or more informal descriptions during the design cycle. This section explains similarities and differences between these approaches.

Scenarios (Carroll, 2000; chap. 5) are the best known of the informal types of description. The importance of scenarios is generally recognized, and their adoption in current practice is impressive. Often scenarios are used even by people who have no knowledge of the HCI literature. When people want to discuss design decisions or requirements for a new system, they find proposing scenarios (i.e., examples of possible use) to be very intuitive. A lively debate on the relationship between task analysis and scenarios can be found in the journal *Interacting with Computers* (2002, issues 4 and 5).

One question is, What is a scenario? Sometimes it seems that any type of description of system use can be a scenario. This brings to mind a discussion on use cases (Cockburn, 1997) in which 18 definitions of use cases were classified based on four issues (purpose, contents, plurality, and structure). Likewise, distinguishing between various types of informal descriptions—stories, scenarios, use cases, and task analyses—can be useful. Stories include some emotional aspects (Imaz & Benyon, 1999); scenarios provide a detailed description of a single, concrete use of a system; use cases are a bit more general and describe groups of use; and task analyses aim to be even more general.

In particular, scenarios, on the one hand, and task analysis and modeling, on the other, should not be confused. Actually, each can be applied without the other, even though it is advisable to use both of them. It is telling that in software engineering scenarios are mainly applied in the requirements phase. In fact, their strong point is their ability to highlight issues and stimulate discussion while requiring limited effort to develop, at least as compared with more formal techniques. However, because a scenario describes a single system use, there is a need for other techniques that may provide less detail but broader scope. Here is where task analysis comes in.

It is often difficult to create a model from scratch, and various approaches have been explored to make it easier. CRITIQUE (Hudson et al., 1999) is a tool that is designed to create KLM/GOMS models (Card, Moran et al., 1983) from the analysis of use session logs. The model is created following two types of rules: The types of KLM operators are identified according to the type of event, and new levels in the hierarchical structure are built when users begin working with a new object or when they change the input to the current object. One limitation of this approach is that the task model only reflects the past use of the system and not other potential uses. U-Tel (Tam, Maulsby, & Puerta, 1998) analyzes textual descriptions of the activities to support and then automatically associates tasks with verbs and objects with nouns. This approach can be useful, but it is too simple to obtain general results. The developers of Isolde (Paris, Tarby, Vander Linden, 2001; chap. 23) have considered the success of UML and provide some support to import use cases created by Rational Rose in their tool for task modeling.

The ConcurTaskTrees Environment (CTTE; Mori, Paternò et al., 2002) supports the initial modeling work by offering the possibility of loading an informal textual description of a

scenario or use case and interactively selecting the information of interest for the modeling. In this way, the designer can first identify tasks, then create a logical hierarchical structure, and finally complete the task model. The use of these features is optional: Designers can start to create the model directly using the task model editor, but such features can ease the work of modeling.

To develop a task model from an informal textual description, designers first have to identify the different roles. Then they can start to analyze the description of the scenario, trying to identify the main tasks that occur in the description and refer each task to a particular role. It is possible to specify the category of the task in terms of performance allocation. In addition, a description of the task can be specified, along with the logical objects used and handled. Reviewing the scenario description, the designer can identify the different tasks and add them to the task list. This must be performed for each role in the application considered.

When each role's main tasks in the scenario have been identified, it might be necessary to make some slight modifications to the newly defined task list. The designers can thus avoid task repetition, refine the task names so as to make them more meaningful, and so on. Once the designers have their list of activities to consider, they can start to create the hierarchical structure that describes the various levels of abstraction among tasks. The final hierarchical structure obtained will be the input for the main editor, allowing specification of the temporal relationships and the tasks' attributes and objects.

24.4 HOW TO REPRESENT THE TASK MODEL

Many proposals have been put forward for representing task models. Hierarchical Task Analysis (Shepherd, 1989; chap. 3) has a long history and is still sometimes used. More generally, such notations can vary according to various dimensions:

Syntax (textual vs. graphical). Some notations are mainly textual, such as UAN (Hartson & Gray, 1992; chaps. 22, 23, and 27), in which there is a textual composition of tasks enriched with tables associated with the basic tasks. GOMS is also mainly textual, even if CPM-GOMS has a more graphical structure because it has been enhanced with PERT charts that highlight parallel activities. ConcurTaskTrees and GTA (van der Veer, Lenting et al., 1996; chaps. 6 and 7) are mainly graphical representations aimed at better highlighting the hierarchical structure. In ConcurTaskTrees the hierarchical structure is represented from the top down, whereas in GTA the representation is from left to right.
Set of operators for task composition. There are substantial differences in regard to these operators among the proposed notations. As Table 24.1 shows, UAN and CTT provide the richest set of temporal relationships and thus allow designers to describe more flexible ways of performing tasks.
Level of formality. Occasionally notations have been proposed without sufficient attention paid to the definition of the operators. When task models are created using such a notation, it is often unclear what is being described because the meaning of many instances of the composition operators is unclear.

The main features of ConcurTaskTrees are as follows:

Focus on activities. ConcurTaskTrees allows designers to concentrate on the activities that users aim to perform. These are what designers should focus on when designing interactive applications that encompass both user- and system-related aspects, and they should avoid

TABLE 24.1
Comparison of Task Model Notation Operators

	GOMS	*UAN*	*CTT*	*MAD*	*GTA*
Sequence	X	X	X	X	X
Order independence		X	X		X
Interruption		X	X	X	
Concurrency	Only CPM-GOMS	X	X	X	X
Optionality			X	X	
Iteration		X	X		X
			X		X
Objects			X		X
Performance	X		X		X
Pre-post condition	X	X	X	X	X

low-level implementation details that, at the design stage, would only obscure the decisions they need to make.

Hierarchical structure. A hierarchical structure is very intuitive (chap. 1). In fact, when people have to solve a problem, they tend to decompose it into smaller problems, still maintaining the relationships among the various parts. The hierarchical structure of this specification has two advantages: It provides a wide range of granularity, allowing large and small task structures to be reused, and it enables reusable task structures to be defined at both low and high semantic levels.

Graphical syntax. A graphical syntax often, though not always, is easier to interpret, because the syntax of ConcurTaskTrees reflects the logical structure of tasks, it has a treelike form.

Rich set of temporal operators. A rich set of possible temporal relationships between the tasks can be defined. This set provides more possibilities than those offered by concurrent notations such as LOTOS. The range of possibilities is usually implicit, expressed informally in the output of task analysis. Making the analyst use these operators is a substantial change from normal practice. The reason for this innovation is to get the designers, after doing an informal task analysis, to express clearly the logical temporal relationships. Such an ordering should be taken into account in the user interface implementation to allow the user to perform at any time the tasks that should be enabled from a semantic point of view.

Task allocation. How the performance of the task is allocated is indicated by the related category, and each category has a unique icon to represent it. There are four categories: tasks allocated to the system, tasks allocated to the user, tasks allocated to an interaction between the system and user, and abstract tasks. Abstract tasks, represented by a cloud icon, include tasks whose subtasks are allocated differently (e.g., one subtask is allocated to the user and one to the system) and tasks that the designer has not yet decided how to allocate (see also chaps. 6 and 22).

Objects and task attributes. Once the tasks are identified, it is important to indicate the objects that have to be manipulated to support their performance. Two broad types of objects can be considered: user interface objects and application domain objects. Multiple user interface objects can be associated with a domain object (e.g., temperature can be represented by a bar- chart or a textual value).

For each single task, it is possible to directly specify a number of attributes and related information. They are classified into three sections:

General information. General information includes the identifier and extended name of the task; its category and type; its frequency of use; informal annotations that the designer may want to store; indication of possible preconditions; and indication of whether it is an iterative, optional, or connection task. Although the category of a task indicates the allocation of its performance, task types allow designers to group tasks based on their semantics. Each category has its own types of task. In the interaction category, the task types include selection (the task allows the user to select some information), control (the task allows the user to trigger a control event that can activate a functionality; chap. 19), editing (the task allows the user to enter a value), monitoring, responding to alerts, and so forth. This classification can help in choosing the most suitable interaction or presentation techniques for supporting the task performance. Frequency of use is another useful type of information because the interaction techniques associated with more frequent tasks need to be better highlighted to obtain an efficient user interface. The platform attribute (e.g., desktop, PDA, cellular, etc.; see also chaps. 22 and 26) allows the designer to indicate for what type of devices the task is suitable (Paternò & Santoro, 2002). This information is particularly useful in the design of nomadic applications (applications that can be accessed through multiple types of platforms). It is also possible to specify if there is any specific precondition that should be satisfied before performing the task.

Objects. For each task, it is possible to indicate the objects (name and class) that have to be manipulated to perform it. Objects can be either user interface or domain application objects. It is also possible to indicate the right access of the user to manipulate the objects when performing the task. Since the performance of the same task in different platforms can require the manipulation of different sets of objects, it is possible to indicate for each platform what objects should be considered. In multi-user applications, different users may have different right accesses.

Time performance. It is also possible to indicate the estimated performance time (including minimal, maximal, and average performance times).

Figure 24.2 presents the temporal operators provided by CTT in greater detail.

It is important to remember that in CTT hierarchy does *not* represent sequence. In Fig. 24.3, the goal of the designer is to specify that "in order to book a flight, I have to select a route, then select a flight, and lastly perform the payment." However the specification is wrong because the sequential temporal evolution should be represented linearly from left to right instead of from the top down.

When there is a need to get back to some point in the specification, it is possible to use the structure of the model and the iterative operator. For example, in Fig. 24.4, once the task *CloseCurrentNavigation* is performed, the task *SelectMuseum* is enabled again because the parent task is iterative. This means that it can be performed multiple times, and so its first subtask is enabled once the last one has been completed.

Optional tasks have a subtle semantics in CTT. They can be used only with concurrent and sequential operators. Their names are enclosed in square brackets. For example, in Fig. 24.5, *Specify type of seat* and *Specify smoking seat* are optional tasks. This means that once the mandatory sibling tasks (*Specify departure* and *Specify arrival*) are performed, then both the optional tasks and the task after the enabling operator (*Send request*) are enabled. If the task after the enabling operator is performed, then the optional tasks are no longer available.

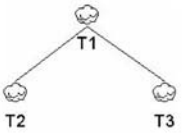
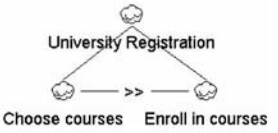
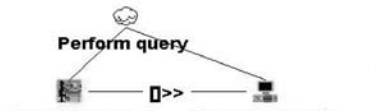
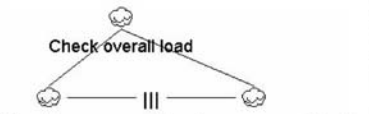
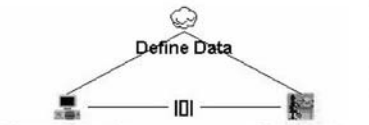
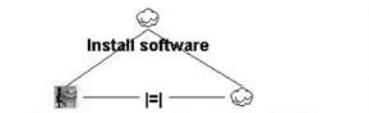
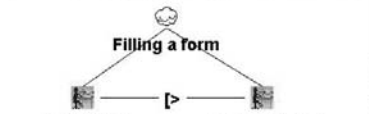
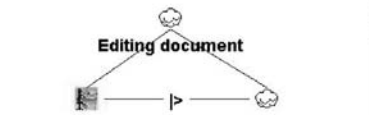
	**Hierarchy** Tasks at same level represent different options or different tasks at the same abstraction level that have to be performed. Read levels as "In order to do T1, I need to do T2 and T3", or "In order to do T1, I need to do T2 or T3"
	**Enabling** Specifies second task cannot begin until first task performed. Example: I cannot enroll at university before I have chosen which courses to take.
	**Choice** Specifies two tasks enabled, then once one has started the other one is no longer enabled. Example: When accessing a web site it is possible either to browse it or to access some detailed information.
	**Enabling with information passing** Specifies second task cannot be performed until first task is performed, and that information produced in first task is used as input for the second one. Example: The system generates results only after that the user specifies a query and the results will depend on the query specified.
	**Concurrent tasks** Tasks can be performed in any order, or at same time, including the possibility of starting a task before the other one has been completed. Example: In order to check the load of a set of courses, I need to consider what terms they fall in and to consider how much work each course represents
	**Concurrent Communicating Tasks** Tasks that can exchange information while performed concurrently Example: An application where the system displays a calendar where it is highlighted the data that is entered in the meantime by the user.
	**Task independence** Tasks can be performed in any order, but when one starts then it has to finish before the other one can start. Example: When people install new software they can start by either registering or implementing the installation but if they start one task they have to finish it before moving to the other one.
	**Disabling** The first task (usually an iterative task) is completely interrupted by the second task. Example: A user can iteratively input data in a form until the form is sent.
	**Suspend-Resume** First task can be interrupted by the second one. When the second terminates then the first one can be reactivated from the state reached before Example: Editing some data and then enabling the possibility of printing them in an environment where when printing is performed then it is no possible to edit.

FIG. 24.2. Temporal operators in ConcurTaskTrees.

FIG. 24.3. Example of wrong specification of sequential activities.

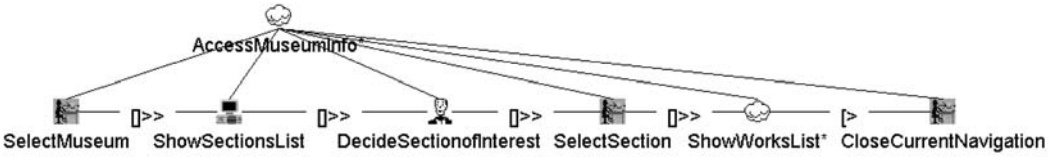


FIG. 24.4. Example of parent iterative task.

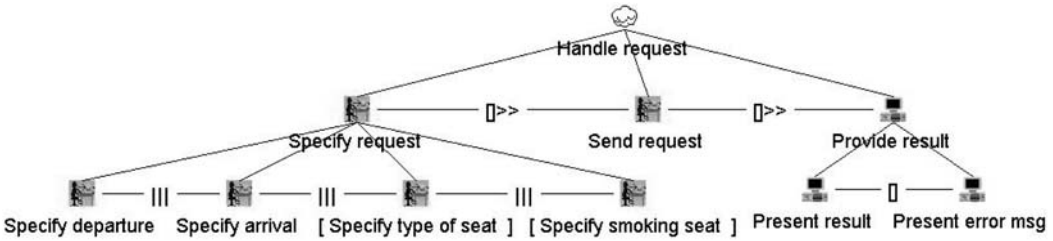


FIG. 24.5. Example of optional tasks.

In the task model, tasks inherit the temporal constraints of the ancestors. So, for example, in Fig. 24.6, *ShowAvailability* is a subtask of *MakeReservation*, and since *Make Reservation* should be performed after *SelectRoomType*, this constraint will apply also to *ShowAvailability*.

With this approach to representing task models, it is possible to represent slightly different behaviors even with small modifications in the specification. The two examples in Fig. 24.7 have the same structure, but the iteration in one case is in the parent task, in the other it is in the subtasks. In the first example, therefore, the parent a task (*Booking flights*) can be performed multiple times, and each time the two subtasks should each be performed once (i.e., overall they will be performed the same number of times). In the other example, the parent task has two iterative subtasks, and thus each of them can be performed an undefined number of times (i.e., the number of times each is performed is independent of the number of times the other is performed).

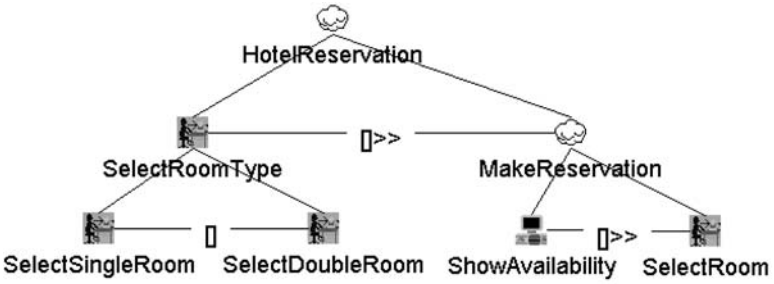


FIG. 24.6. Example of inheritance of temporal constraints.

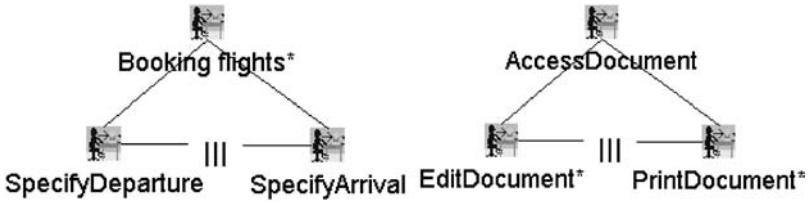


FIG. 24.7. Example of different structures in task model specifications.

Some of the features described were not included in the original version of CTT. The order independence operator has been added to provide an immediate description of a specific type of constraint. In the case of order independence, when a task is started, it has to be completed before another task is begun. With concurrent tasks, it is possible to start the performance of another task before the completion of the first task started.

Also recently added is the platform attribute. It allows designers to specify what platforms the task is suitable for. In addition, it is possible to specify, for each object manipulated by the task, the platforms suitable for supporting the task. This type of information allows designers to better address the design of nomadic applications—those that can be accessed through a wide variety of platforms. The basic idea is that designers start by specifying the task model for the entire nomadic application and can then derive the system task model for each platform by filtering tasks through the platform attribute.

24.4.1 An Example of Specification

Figure 24.8 represents the task model of a mobile phone. The task model is presented in an incomplete form for the sake of brevity. It is important to note that the task model depends on the features of one specific type of cellular phone. It shows nonetheless that task models can be useful for analyzing the design of more recent mobile systems.

The model is structured into two sequential parts: One dedicated to connecting the phone and the other one to handling the communication. At any time, it is possible to switch off the phone. The first part is a sequence of interactive and system tasks. The second part is dedicated to the actual use of the phone, which is iterative because multiple uses are possible in a session. First of all, the user has to decide what he or she wants to do. Next, there are two main choices: either make a call (the most frequent function) or use other functions. In case of a call, the user can either select the number from a list or recall and enter it. The other possible uses are not detailed for the sake of brevity.

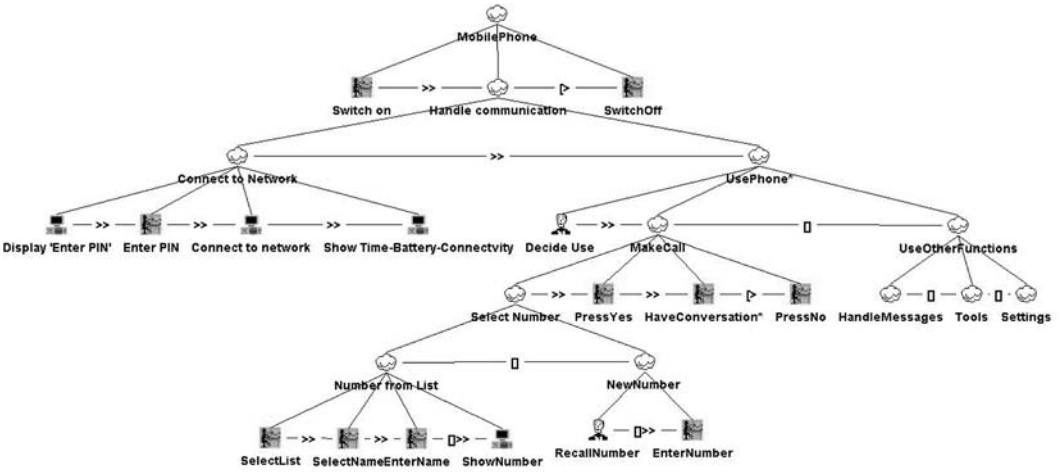


FIG. 24.8. Example of task model specification with ConcurTaskTrees.

24.5 UML AND TASK MODELS

Model-based approaches have often been used in software engineering. If we consider UML, one of the most successful model-based approach for the design of software systems, we notice a considerable effort is usually made to provide models and representations to support the various phases and parts of the design and the development of software applications. However, of the nine representations provided by UML, none is particularly oriented to supporting the design of user interfaces. Of course, it is possible to use some of them to represent aspects related to the user interface, but it is clear that this is not their main purpose. To integrate the two approaches (task models represented in ConcurTaskTrees and UML), various basic strategies can be used (see below). These can exploit, to different extents, the extensibility mechanisms built into UML itself (constraints, stereotypes, and tagged values), which can extend UML without requiring changes in the basic UML metamodel. The basic strategies are as follows:

Representing elements and operators of a task model by an existing UML notation. For example, if you consider a ConcurTaskTrees model as a forest of task trees, where ConcurTaskTrees operands are nodes and operators are horizontally directed arcs between sibling nodes, you can represent the model as UML class diagrams. Specific UML class and association stereotypes, tagged values, and constraints can be defined to factor out and represent properties of and constraints on CTT elements.

Developing automatic converters from UML to task models. For example, it is possible to use the information contained in system-behavior models supported by UML (i.e., use cases, use case diagrams, and interaction diagrams) to develop task models.

Building a new UML for interactive systems. A new UML can be obtained by explicitly inserting ConcurTaskTrees in the set of available notations while still creating semantic mapping of ConcurTaskTrees concepts into a UML metamodel. This encompasses identifying correspondences, at both the conceptual and structural levels, between ConcurTaskTrees elements and concepts and UML ones and exploiting UML extensibility mechanisms to support this solution.

Of course, there are advantages and disadvantages in each approach. In the first, it would be possible to have a solution compliant with a standard that is already the result of many

long discussions involving many people. This solution is surely feasible. An example is given in Nunes and Falcao (2000): CTT diagrams are represented as stereotyped class diagrams. Furthermore, constraints associated with UML class and association stereotypes can be defined so as to enforce the structural correctness of ConcurTaskTrees models. However, two key issues arise: whether the notation has enough expressive power and whether the representations are effective and support designers in their work rather than complicate it. The usability aspect is important not only for the final application but also for the representations used in the design process. For example, activity diagrams are general and provide sufficient expressive power to describe activities. However, they tend to provide lower level descriptions than those in task models, and they require rather complicated expressions to represent task models describing flexible behaviors.

Thorny problems also emerge for the second approach. In particular, it is difficult to first model a system in terms of object behaviors and then derive a meaningful task model from such models. The reason is that object-oriented approaches are usually effective for modeling internal system aspects but less adequate for capturing users' activities and their interactions with the system.

The third approach seems to offer more promise as a way to capture the requirements for an environment supporting the design of interactive systems. However, care should be taken to ensure that software engineers who are familiar with traditional UML can make the transition to this new method easily and that the degree of extension from the current UML standard remains limited. More specifically, use cases could be useful in identifying tasks to perform and related requirements, but then there is no notation suitable for representing task models, although there are various ways to represent the objects of the system under design. This means that there is a wide gap that needs to be filled in order to support models able to assist in the design of user interfaces.

In defining a UML for interactive systems (Paternò, 2001), the designers can explicitly introduce the use of task models represented in CTT. However, not all UML notations are equally relevant to the design of interactive systems; the most important in this respect appear to be use cases, class diagrams, and sequence diagrams.

In the initial part of the design process, during the requirement elicitation phase, use cases supported by related diagrams should be used. Use cases are defined as coherent units of externally visible functionality provided by a system unit. Their purpose is to define a piece of coherent behavior without revealing the internal structure of the system. They have shown to be successful in industrial practice.

Next, there is the task-modeling phase, which allows designers to obtain an integrated view of functional and interactional aspects. In particular, interactional aspects (aspects related to the ways of accessing system functionality) cannot be captured well in use cases. In order to overcome this limitation, use cases can be enriched with scenarios (i.e., informal descriptions of specific uses of the system). More user-related aspects can emerge during task modeling. In this phase, tasks should be refined, along with their temporal relationships and attributes. The support of graphically represented hierarchical structures, enriched by a powerful set of temporal operators, is particularly important. It reflects the logical approach of most designers, allows the description of a rich set of possibilities, is declarative, and generates compact descriptions.

In parallel with the task-modeling work, the domain modelling is also refined. The goal is to achieve a complete identification of the objects belonging to the domain considered and the relationships among them. At some point there is a need for integrating the information between the two models. Designers need to associate tasks with objects in order to indicate what objects should be manipulated to perform each task. This information can be directly introduced in the task model. In CTT it is possible to specify the relationships between tasks and objects. For each task, it is possible to indicate the related objects, including their classes

and identifiers. However, in the domain model more elaborate relationships among the objects are identified (e.g., association, dependency, flow, generalization, etc.), and they can be easily supported by UML class diagrams.

There are two general kinds of objects that should be considered: presentation objects, those composing the user interface, and application objects, which are derived from the domain analysis and responsible for the representation of persistent information, typically within a database or repository. These two kinds of objects interact with each other: Presentation objects are responsible for creating, modifying, and rendering application objects. The refinement of tasks and objects can be performed in parallel so that first the more abstract tasks and objects are identified and then the more concrete tasks and objects. At some point, the task and domain models should be integrated in order to clearly specify the tasks that access each object and, vice versa, the objects that are manipulated by each task.

24.6 TASK MODELS FOR MULTIPLATFORM APPLICATIONS

The availability of an increasing number of device types ranging from small cellular phones to large screens is changing the structure of many applications (see chap. 26). It is often the case that within the same application users can access its functionality through different devices. In a nomadic application, users can access the application from different locations through different devices. This means that context of use is becoming increasingly important, and it is essential to understand its relationship with tasks.

In general, it is important to understand what type of tasks can actually be performed in each platform. In a multiplatform application, there are various possibilities:

1. Same task on multiple platforms in the same manner (there could be only a change of attributes of the user interface objects).
2. Same task on multiple platforms with performance differences:
   - Different domain objects. For example, presenting information on works of arts can show different objects depending on the capability of the current device.
   - Different user interface objects. For example, in a desktop application it is possible to support a choice among elements graphically represented, whereas the same choice in a Wap phone is supported through a textual choice.
   - Different task decomposition. For example, accessing a work of art through a desktop can support the possibility of accessing related information and further details not supported through a Wap phone.
   - Different temporal relationships. For example, a desktop system can support concurrent access to different pieces of information that could be accessed only sequentially through a platform with limited capabilities.
3. Tasks meaningful only on a single platform. For example, browsing detailed information makes sense only with a desktop system.
4. Dependencies among tasks performed on different platforms. For example, during a physical visit to a museum users can annotate works of art that should be shown first when they access the museum Web site.

All these cases can be specified in ConcurTaskTrees through the use of the platform attribute that can be associated with both tasks and objects. Such an attribute indicates what platforms are suitable to support the various tasks and objects, thus allowing designers to represent the task model even in the case of nomadic applications.

24.6.1 Task Models for Cooperative Applications

Providing support for cooperative applications is important because the increasing spread and improvement of Internet connections makes it possible to use many types of cooperative applications. Consequently, tools supporting the design of applications where multiple users can interactively cooperate are more and more required.

In our approach, when there are multi-user applications, the task model is composed of various parts. A role is identified by a specific set of tasks and relationships among them. Thus, there is one task model for each role involved. In addition, there is a cooperative part whose purpose is to indicate the relationships among tasks performed by different users.

The cooperative part is described in a manner similar to the single user parts: It is a hierarchical structure with indications of the temporal operators. The main difference is that it includes cooperative tasks—those tasks that imply actions by two or more users in order to be performed. For example, negotiating a price is a cooperative task because it requires actions from both a customer and a salesperson. Cooperative tasks are represented by a specific icon with two persons interacting with each other.

In the cooperative part, cooperative tasks are decomposed until tasks performed by a single user are reached (these are represented with the icons used in the single-user parts). These single-user tasks will also appear in the task model of the associated role. They are defined as *connection tasks* between the single-user parts and the cooperative part. In the task specification of a role (see, e.g., the top part of Fig. 24.9), connection tasks are identified by a double arrow under their names.

With this structure of representation, in order to understand whether a task performance is enabled, it is necessary to check both the constraints in the relative single-user part and the constraints in the cooperative part. It may happen that a task without constraints in the

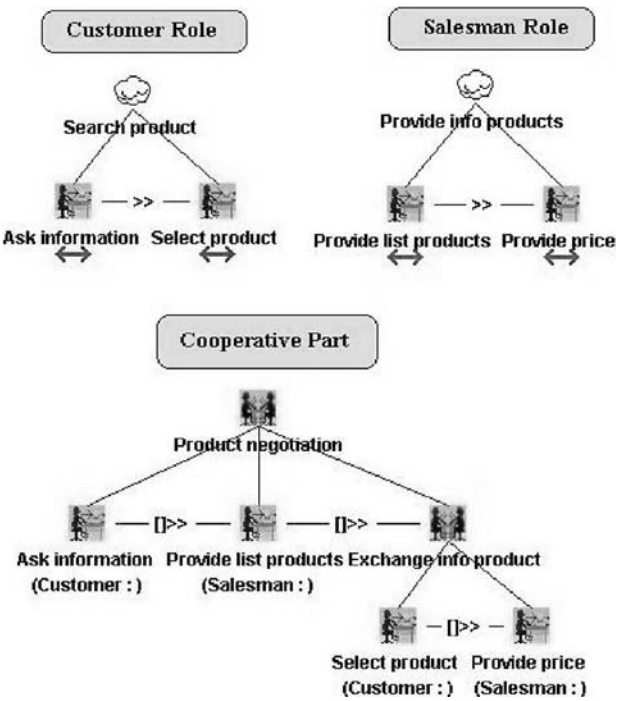


FIG. 24.9. Simplified example of cooperative task model.

single-user parts is not enabled because there is a constraint in the cooperative part, indicating that another user must first perform another task. Figure 24.8 shows a *Search Product* task performed by a customer and a *Provide Info Product* task performed by a salesperson. If we consider each part of the task model in isolation, these two tasks can be started immediately. However, if we consider the additional constraint indicated in the part below, we can see that the performance of the *Provide list products* task (by the salesperson) needs to wait for the performance of the *Ask information* task (by the customer) in order to be enabled.

We are aware that in cooperative applications, users interact not only while performing their routine tasks but also when some external event blocks the work flow. Moreover, such events can create situations where previously defined tasks need to be changed and/or repeated. The task model associated with a certain role enables the distinguishing of individual tasks and of tasks that involve cooperation with other users. Then, moving on to the cooperative part, it is possible to see what the common goals are and what cooperation among multiple users is required in order to achieve them. The set of temporal operators also allows the description of flexible situations where, when something occurs, activities need to be performed in a different manner. Of course, specifying flexible behaviors requires an increase in the complexity of the specification. Trying to anticipate everything that could go wrong in complex operations would render the specification exceedingly complex and large. The extent to which designers want to increase the complexity of the specification to address these issues depends on many factors (e.g., the importance of the specification for the current project, the resources available, and their experience in task modeling).

The CTTE tool allows different ways to browse the task model of cooperative applications. The designers can interactively select the pane associated with the part of the task model of interest. (In Fig. 24.9 the task model is composed of a cooperative part and two roles, customer and sales representative.) In addition, when connection tasks are selected in the task model of a role (they are highlighted by a double arrow below their name), it is possible to automatically visualize where they appear in the cooperative part and vice versa.

24.6.2 Tool Support

Table 24.2 summarizes some features deemed important for analysis tools and shows whether task models are supported by some of the currently available tools. The features provided by CTTE (see Fig. 24.10) highlight its contribution to understanding the possibilities in terms of the analyses that can be performed with such tools. The first row indicates whether the tools are able to consider concurrent tasks, (that is, their analysis is not limited to sequential

TABLE 24.2
Comparison of Task-Modeling Tools

	CTTE	*EUTERPE*	*VTMB*	*QDGOMS*	*IMAD**	*Adept*
Concurrent task	X	X	X		X	X
Cooperative tasks	X	X			X	
Metrics	X	X				
Reachability	X					
Performance evaluation	X			X		
Simulator	X		X		X	

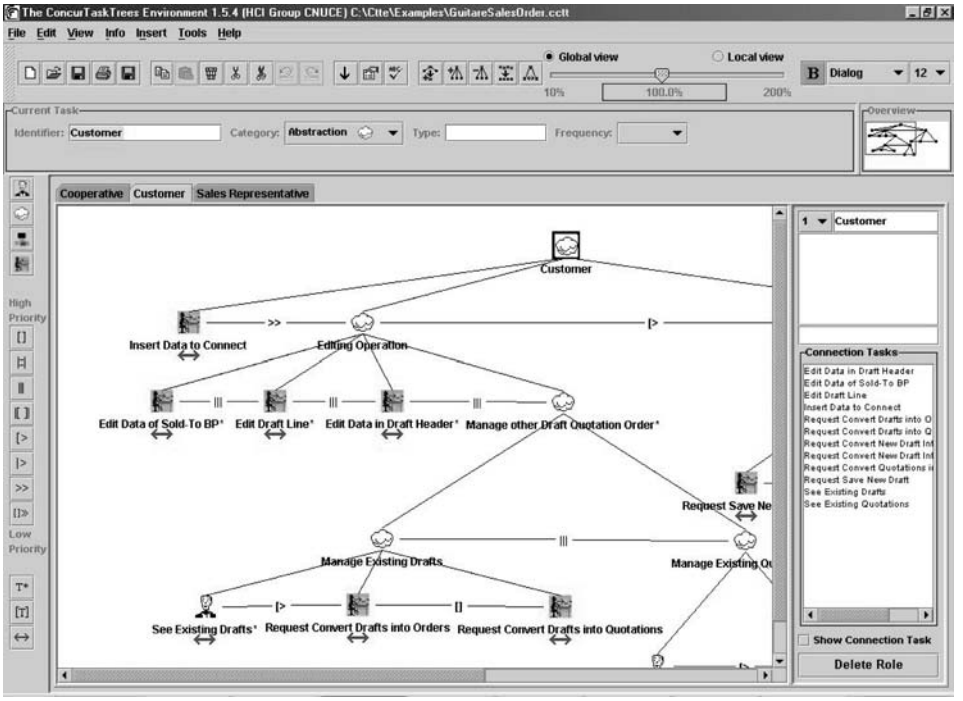


FIG. 24.10. The layout of CTTE.

tasks). The next row indicates their ability to analyze cooperative tasks involving multiple users. The third row indicates whether they are able to calculate some metrics. CTTE is able to compare two models with respect to a set of metrics, and EUTERPE (van Welie, van der Veer, & Eliëns, 1998; chaps. 6 and 7) also supports the calculation of some metrics to analyze a specification and find inconsistencies or problems. The remaining features considered concern the ability to predict task performance. This is usually supported by tools for GOMS such as QDGOMS (Beard, Smith, et al., 1996) that interactively simulate the task model's dynamic behavior. Interactive simulation is also another important feature supported by a few tools, such as VTMB (Biere, Bomsdorf, & Szwillus, 1999).

24.7 CONCLUSION

CTT and its related environment represent a contribution to engineering the use of task models in the design, development, and evaluation of interactive systems. In this chapter, the main features of the notation have been discussed, and indications how to address important related issues (e.g., integration with UML, design of multiplatform applications, and creation of task models for cooperative applications) have also been provided.

The approach has also stimulated a good deal of research work: Nunes and Falcao (2000) proposed a transformation from UML to CTT, Sage and Johnson (1997) proposed integrating CTT and a functional language to obtain a user interface prototyping environment, and Pribeanu, Limbourg et al., (2001) explored ways to extend CTT to include context-of-use information. Furthermore, the reference framework for designing user interfaces able to adapt to multiple platforms while preserving usability (Calvary, Coutaz, & Thevenin, 2001) considers task models (in particular, those represented by CTT) as a starting point for the design

work, and the MEFISTO project, which aimed to identify methods supporting the design of air traffic control applications, provided an integration between task models represented by CTT and system models represented by ICO Petri Nets (Navaree, Palanque, Paternò, Santoro, & Bastide, 2001). In addition, the notation and tool have been used for teaching purposes and in various projects all over the world.

ACKNOWLEDGMENT

Thanks to Carmen Santoro for useful discussions on the topics addressed in this chapter.

REFERENCES

Beard, D., Smith, D., & Denelsbeck, K. (1996). Quick and dirty GOMS: A case study of computed tomography. *Human-Computer Interaction, 11*, 157–180.

Biere, M., Bomsdorf, B., & Szwillus G. (1999). Specification and simulation of task models with VTMB. In *Proceedings CHI'99* (Extended abstracts; pp. 1–2). New York: ACM Press.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *Unified Modeling Language reference manual.* Reading, MA: Addison-Wesley.

Calvary, G., Coutaz, J., & Thèvenin, D. (2001). A Unifying reference framework for the development of plastic user interfaces. In *Proceedings Engineering HCI '01* (pp. 218–238). Springer-Verlag.

Card, S., Moran, T., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Carroll, J. (2000). *Making use: Scenario-based design of human-computer interactions.* Cambridge, MA: MIT Press.

Cockburn, A. (1997). *Structuring use cases with goals.* Available: http://members.aol.com/acockburn

Gamboa-Rodriguez, F., & Scapin, D. (1997). Editing MAD* task description for specifying user interfaces at both semantic and presentation Levels. In *Proceedings DSV-IS'97* (pp. 193–208). Springer-Verlag.

Gamma, E., Helm, R., Johnson, R., & Vlissides J. (1995). *Design patterns: Elements of reusable object-oriented software.* Reading, MA: Addison-Wesley.

Hartson, R., & Gray, P. (1992). Temporal aspects of tasks in the user action notation. *Human-Computer Interaction, 7*, 1–45.

Hudson, S., John, B., Knudsen, K., & Byrne, M. (1999). A tool for creating predictive performance models from user interface demonstrations. In *Proceedings UIST'99* (pp. xxx). New York: ACM Press.

Imaz, M., & Benyon, D. (1999). How stories capture interactions. In *Proceedings Interact'99* (pp. 321–328).

John, B., & Kieras, D. (1996). The GOMS family of analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction, 3*, 320–351.

Limbourg, Q., Ait El Hadj, B., Vanderdonckt, J., Keymolen, G., & Mbaki, E. (2000). Towards derivation of presentation and dialogue from models: Preliminary results. In *Proceedings DSV-IS 2000* (Lecture Notes in Computer Science 1946; pp. 227–248). Springer-Verlag.

Mori G., Paternò F., & Santoro, C. (2002). CTTE: Support for Developing and analyzing task models for interactive systems design. *IEEE Transactions on Software Engineering,*

Navarre, D., Palanque, P., Paternò, F., Santoro, C., & Bastide, R. (2001). A tool suite for integrating task and system models through scenarios. (pp. 88–113). Springer-Verlag.

Norman, D. (1986). Cognitive engineering. In D. Norman & S. Draper (Eds.), *User centered system design: New perspectives on human-computer interaction* (pp. 31–61). Hillsdale, NJ: Lawrence Erlbaum Associates.

Numes, N., & Falcao, J. (2000). Towards a UML profile for user interface development: The Wisdom approach. In *Proceedings UML'2000* (Lecture Notes in Computer Science; (pp. 50–58). Springer-Verlag.

Paganelli, L., & Paternò, F. (2002). Intelligent analysis of user interactions with Web applications. In *Proceedings ACM IUI 2002* (pp. 111–118). New York: ACM Press.

Paris, C., Tarby, J., & Vander Linden, K. (2001). A flexible environment for building task models. In *Proceedings of the HCI 2001,* Lille, France. (pp. xxx).

Paternò, F. (1999). *Model-based design and evaluation of interactive application*. Springer-Verlag.

Paternò. (2001). Towards a UML for interactive systems. In *Proceedings Engineering HCI '01* (pp. 175–185). Toronto: Springer-Verlag.

Paternò, F., & Santoro, C. (2002). One model, Many interfaces. In *Proceedings CADUI 2002* (pp. 143–154). Kluwer.

Paternò, F., Santoro, C., & Sabbatino, V. (2000). Using information in task models to support design of interactive safety-critical applications. In *Proceedings AVI'2000* (pp. 120–127). New York: ACM Press.

Pribeanu, C., Limbourg Q., & Vanderdonckt, J. (2001). Task modelling for context-sensitive user interfaces. In *Proceedings DSV-IS 2001* (pp. 49–68). Heidelberg: Springer-Verlag.

Sage, M., & Johnson, C. (1997). Haggis: Executable specifications for interactive systems. In *Proceedings DSV-IS'97* (pp. 93–109). Heidelberg: Springer-Verlag.

Scapin, D., & Pierret-Golbreich, C. (1989). Towards a method for task descrption: MAD. In *Proceedings Work with Display Unit* (pp. 78–92). The Netherlands: Elsevier.

Shepherd, A. (1989). Analysis and training in information technology tasks. In D. Diaper (Ed.), *Task analysis for human-computer interaction* (pp. 15–55). Chichester, England: Ellis Horwood.

Tam, R.C.-M., Maulsby, D., & Puerta, A. (1998). U-TEL: A tool for eliciting user task models from domain experts. In *Proceedings IUI'98* (pp. 77–80). New York: ACM Press.

van der Veer, G., Lenting, B., & Bergevoet, B. (1996). GTA: Groupware task analysis-modelling complexity. *Acta Psychologica, 91*, 297–322.

van Welie, M., van der Veer, G. C., & Eliëns, A. (1998). An Ontology for task world models. In *Proceedings DSV-IS'98* (pp. 57–70). Heidelberg: Springer-Verlag.

Wilson, S., Johnson, P., Kelly, C., Cunningham, J., & Markopoulos, P. (1993). Beyond hacking: A model-based approach to user Interface design. In *Proceedings HCI'93* (pp. 40–48). New York: Cambridge University Press.