CTTE: Support for Developing and Analyzing Task Models for Interactive System Design

Giulio Mori, Fabio Paternò, and Carmen Santoro

Abstract—While task modeling and task-based design are entering into current practice in the design of interactive software applications, there is still a lack of tools supporting the development and analysis of task models. Such tools should provide developers with ways to represent tasks, including their attributes and objects and their temporal and semantic relationships, to easily create, analyze, and modify such representations and to simulate their dynamic behavior. In this paper, we present a tool, CTTE, that provides thorough support for developing and analyzing task models of cooperative applications, which can then be used to improve the design and evaluation of interactive software applications. We discuss how we have designed this environment and report on trials of its use.

Index Terms—Task models, models for interactive software systems, automatic tools for human-computer interaction, user interfaces.

1 Introduction

1.1 User Interface Tools

The pervasiveness of software applications requires user interfaces able to support a wide variety of tasks, in a wide variety of contexts, and accessible through many possible devices. The user interface component of interactive applications is acquiring ever more importance. This is because, often, many different applications are available to perform similar tasks and users choose those that are easier to understand and interact with and which, consequently, increase efficiency, productivity, and acceptance while reducing errors and the need for training.

The main solution that has been proposed to effectively support user interface design and development has been through visual tools such as Visual Basic or visual environments for Java that allow designers to easily develop a user interface by direct manipulation techniques. The advantage is that a set of instances of predefined interaction techniques can be easily arranged to define the layout of the user interface. However, in this type of approach, there is a lack of support for identification of the presentation and interaction techniques that are more effective in supporting the activities users intend to perform.

1.2 Task Models

To overcome such limitations, interest in model-based approaches [22] has been increasing in recent years. The basic idea is to identify useful abstractions highlighting the main aspects that should be considered when designing effective interactive applications. Of the relevant models, task models play a particularly important role because they indicate the logical activities that an application should support. A task is an activity that should be performed in order to reach a goal. A goal is either a desired modification of state or an inquiry to obtain information on the current state. For example, querying the available flights from Pisa to London is one task that must be performed in order to book a flight to London (the relative goal). Tasks can range from a very high abstraction level (such as deciding a strategy for solving a problem) to a concrete, action-oriented level (such as selecting a printer).

In order to be meaningful, the development of the task model of a new application should be the result of an interdisciplinary discussion involving the various viewpoints that should be considered (such as designers, developers, end users, managers, and domain experts). This would be useful in order to obtain user interfaces able to easily support the desired activities, thus implying that the user task model (how users think that the activities should be performed) and the system task model (how the application assumes that activities are performed) correspond closely.

Task models have been used in various approaches to support different phases of the software development life cycle:

- *Requirement analysis*: where, through a task analysis, designers identify requirements that should be satisfied in order to perform tasks effectively (GTA [31]).
- *Design of interactive applications* (Adept [33], Trident [5], Mobile [25]): in this case, the goal is to use information contained in logical models to identify the interaction and presentation techniques best suited to support the tasks at hand and the dialogues whose temporal constraints are closest to those of the model.
- *Usability evaluation*: it can be supported in various ways: for example, identifying task efficiency (such as in the KLM [9] approach) or analyzing logs of user interactions with the support of task models (see, for example, RemUSINE [20]).

• *The authors are with ISTI, an Institute of the National Research Council of Italy, Via G. Moruzzi 1, 56124 Pisa, Italy.*
E-mail: giulio.mori@guest.cnuce.cnr.it., {fabio.paterno, carmelina.santoro}@cnuce.cnr.it.

Manuscript received 6 July 2000; revised 16 Feb. 2001; accepted 3 Jan. 2002. Recommended for acceptance by P. Johnson.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 112398.

Task models can be used for descriptive and prescriptive purposes, for example, to describe how tasks are supported by an existing system or to prescribe how tasks should be supported by a new system to develop. They can be developed at different levels of abstractions (artifact-independent or related to the use of specific devices).

1.3 Software Engineering and Human-Computer Interaction

There is a need for greater integration of research in software engineering and human-computer interaction. The field of interactive systems development will benefit considerably if the different theories, models, techniques, and tools can be brought together effectively.

Task models represent the intersection between user interface design and more formal software 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. 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:

- 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.
- Need for *systematic methods* to support the specification, analysis, and use of task models, to facilitate their development, and support designers in using the knowledge that they incorporate to address the design of the user interface and its evaluation. We note that, often, even designers who developed some task analysis and modeling did not use them for the detailed design of the user interface because of this lack of structured methods, which should give rules and suggestions about how to use information in the task model for the concrete design.
- Support for the *reuse* of good design solutions to problems which occur across many applications [15]. This is relevant, especially in an industrial context, where developers often have to design applications which address similar problems, thus it would be useful to have design solutions structured and documented in such a way as to support easy reuse and tailoring in different applications.
- Availability of *automatic tools* to support the various phases of the design cycle. Also, the development of these tools should pay attention to user interface aspects so as to have intuitive representations and provide information useful for the logical activities of designers.

While these concepts have long been considered in software engineering, there is a lack of approaches that have considered how to apply them to address the design of human-computer interactions. This goal requires attention to aspects that have been poorly considered in software engineering: the user, how users perceive and interpret information, organize their activities, and interact with systems in order to perform them. Task models can be a bridge between the two communities and their advancement would benefit highly from the integration and enhancement of concepts and techniques utilized in both.

One of the main problems in task modeling is that it is a time-consuming, sometimes discouraging, process. To overcome such a limitation, interest has been increasing in the use of tool support [8]. Despite this widespread interest, tools developed for task models have been rather rudimentary, mainly research tools used only by the groups that developed them, whereas more systematically engineered tool support is strongly required in order to ease the development and analysis of task models and make them acceptable to a large number of designers. If we consider the first generation of tools for task models we can notice that they are mainly research prototypes aiming at supporting the editing of the task model and related information. This has been a useful contribution, but further automatic support is required to make use of task models acceptable to a large number of designers. Thus, there is a strong need for engineered tools with the ability to support task modeling of applications, including cooperative and multiuser ones, which are on the rise, and the analysis of such models' contents, which may can be complex in the case of many real applications.

To overcome such problems, we have designed and developed the ConcurTaskTrees Environment (CTTE), which is a tool that can be used to support the design of interactive applications, for teaching purposes, and to produce interactive documentation of applications. In developing the approach, we have also paid attention to the usability of the notation and related environment in order to ease the work of designers.

1.4 Structure of the Article

In the paper, we start with a discussion of related works, and introduce the approach and the notation used for task modeling of multiuser applications (single user applications can be seen as a simplified case of this more general case). Then, we move on to describe and discuss the automatic tool that we have designed and developed to give support in the development and analysis of such models. We also give brief examples of specifications, discuss the applications designed with the support of the tool, and report more generally on its use. We conclude by discussing the lessons learned and providing some concluding remarks.

2 Related Works

2.1 Task Analysis and Modeling

The earlier works in model-based design of user interfaces mainly focused on abstractions related to the objects that have to be manipulated. For example, in UIDE [13], the authors structure the logical models in terms of objects associated with pre and post conditions. Their purpose is to indicate what should be verified in order to enable interaction with the object and to indicate the effects triggered by such interactions. A set of similar abstractions was developed also for other model-based approaches, such as Humanoid [30].

Some research work has been focusing on identifying the goals that users want to achieve and how to perform activities able to reach such goals. One of the first contributions in this area was GOMS (Goals, Operators, Methods, Selection Rules) [9] with which it is possible to describe the action sequences required to reach a goal and group them into methods that may have an associated selection rule to indicate when their performance is recommended. However, most GOMS approaches consider only sequential tasks. In addition, the tool support provided for them is still far from what designers and evaluators expect from GOMS tools [3] because of limited pragmatics, partial coverage of the GOMS concepts, limited support for cognitively plausible models, and lack of integration in the development process.

A richer set of temporal operators can be specified in UAN (User Action Notation) [17]. It is a textual notation where it is possible to describe the temporal relationships among asynchronous tasks and, finally, the basic tasks (those that are no longer decomposed) are described by tables indicating: user actions, system feedback, and modifications in the state of the application. However, the textual syntax of UAN and its lack of tool support has been a strong limitation to its diffusion.

GTA [31] is another relevant work in the area; it mainly focuses on the analysis phase aiming at eliciting requirements that can be important in the user interface design. Some tool support for this approach has started with Euterpe [32], but it is still at a prototypal phase.

Also, the need for more human-centered specifications is argued in [18], where intent specifications are proposed, with particular attention to the design of safety-critical systems. However, tool support for such a framework has not been provided.

In the meantime, the use of task models for supporting design has been recognized and interesting methods have been developed for this purpose, but the tool support provided was still either limited [33], [28] or wholly missing [29]. The tool support for models for user interface has been provided in Pet-Shop, which is an environment allowing development of a technique combining Petri Nets and object-oriented techniques [6]. The dialogue of the user interface is modeled by Petri Nets; more precisely, for each widget in the user interface, there is a Petri Net modeling its dynamic behavior. Such a tool has shown to be useful for modeling the runtime user interface behavior. It also supports the possibility of connecting the user interface and the related Petri Net and shows how the tokens evolve while the user interacts with the application. Thus, the purpose is to support the analysis of the user interface system, rather than the related task model: It provides a description of the objects making up the user interface and the underlying system and how they behave, whereas, in the task model, there is a description of the logical activities supported. There is an intersection between the aspects described by the two models (the actions that can be performed at the user interface level), but each of them also captures aspects that are not represented in the other one: The task model also describes cognitive user activities (such as deciding a strategy to find information) and the system model contains a description of the system architecture. Another tool has been developed in the Teallach project [4], where task models are considered but the tool support focuses on the generation of the user interface for database applications rather than the analysis of the task models. While we think tool support for the user interface generation is useful and important, we believe that there is also a strong need for helping designers to develop and analyze the task models and, in this phase, there is a strong lack of tool support.

2.2 UML and Task Models

Model-based approaches have often been used in software engineering. If we consider UML [7], the most successful model-based approach for the design of software systems, we can notice a considerable effort to provide models and representations to support the various phases and parts of the design and development of software applications. However, despite the nine representations provided by UML, none of them 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. Aimed at integrating the two approaches (task models represented in ConcurTaskTrees and UML), there can be various basic philosophies that are outlined below. Such approaches can exploit, to different extents, the extensibility mechanisms built into UML itself (constraints, stereotypes, and tagged values) that enable extending UML without having to change the basic UML metamodel. The types of approaches are:

- *Representing elements and operators of a task model by an existing UML notation*, for example, considering a ConcurTaskTrees model as a forest of task trees, where ConcurTaskTrees operands are nodes and operators are horizontally directed arcs between sibling nodes, this can be represented as UML class diagrams. Specific UML class and association stereotypes, tagged values, and constraints can be defined to factor out and represent properties and constraints of CTT elements [21].
- *Developing automatic converters from UML to task models*, for example, using the information contained in system-behavior models supported by UML (use cases, use case diagrams, interaction diagrams) to develop task models.
- *Building a new UML for interactive systems*, which can be obtained by explicitly inserting ConcurTaskTrees in the set of available notations while still creating semantic mapping of ConcurTaskTrees concepts into 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 case, 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 [21]:

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, the key issue is not only a matter of the expressive power of notations, but it is also a matter of representations that should be effective and support designers in their work, rather than complicate it. The usability aspect is not only important for the final application, but also for the representations used in the design process. For example, activity diagrams are general and provide good 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 from the second approach. It is difficult to first model a system in terms of object behaviors and then derive a meaningful task model from such models because 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 be more promising in capturing the requirements for an environment supporting the design of interactive systems. However, attention should be paid so that software engineers who are familiar with traditional UML can make the transition to this new method easily while limiting the degree of extension from the current UML standard. More specifically, use cases could be useful in identifying tasks to perform and related requirements, but then there is no notation suitable to representing task models, while there are various ways to represent the objects of the system to design. This means that there is a wide gap that needs to be filled in order to support models important for the design of user interfaces. However, the purpose of this paper is to present CTTE and not its integration with UML. A proposal aimed at obtaining a UML for interactive systems based on the integration of CTT and current UML is outlined in [23].

2.3 Modeling Multiuser Interactions

A need for approaches able to consider cooperative applications as well is also motivated by the current toolkits for multiuser interface development: Even when they have proposed innovative solutions [27], they have provided designers and developers with rather low-level constructs. As described in [12] and [10], a groupware system covers three domain specific functions: production, coordination, and communication. The *production* space denotes the set of domain objects that model the multiuser manipulation of common artifacts, such as documents, or that motivate a common undertaking, such as flying an airplane between two locations. The *coordination* space covers dependencies among activities including temporal relationships between the multiuser activities. The *communication* space supports person-to-person communication. Coordination seems to be a rather important part in designing cooperative applications. Task models with descriptions of temporal relationships among tasks can provide useful logical descriptions of such coordination. They also address the identification of domain objects and communication aspects.

3 How the Task Model Is Represented

In our environment, the task models considered are structured in a hierarchical fashion according to the ConcurTaskTrees notation that was introduced in [22]. It provides a rich set of operators to describe the temporal relationships among tasks. In addition, for each task, further information can be given, such as its type, the category (indicating how the performance is allocated), the objects manipulated, and attributes, such as frequency of performance.

ConcurTaskTrees was developed after first studies developed to specify graphical user interfaces by using the LOTOS notation [1]. LOTOS is a concurrent formal notation that seemed a good choice to specify user interfaces because it allows designers to describe both event-driven behaviors and state modifications. However, it showed some limitations that make it unlikely to be widely used in the human-computer interaction domain. It was soon realized that there was a need for new operators to express a richer set of dynamic behaviors in task models in a compact way (such as optional tasks and order independence performance) and additional information useful in analyzing and representing task models capturing their relevant attributes. Moreover, LOTOS has a textual syntax that can easily generate complex expressions, even when the behavior to describe is quite simple. Thus, a new notation was developed, ConcurTaskTrees. Its main aim 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.

The main features of ConcurTaskTrees are:

- *Focus on activities*. It allows designers to concentrate on the activities that users aim to perform that are the most relevant aspects when designing interactive applications that encompass both user and system-related aspects, avoiding low-level implementation details that, at the design stage, would only obscure the decisions to take.
- *Hierarchical structure*. A hierarchical structure is something very intuitive; in fact, often, when people have to solve a problem, they tend to decompose it into smaller problems still maintaining the relationships among the various parts of the solution. 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 level.
- *Graphical syntax*. Often, a graphical syntax is easier to interpret; in this case, it reflects the logical structure, so it has a tree-like form.
- *Concurrent notation*. A rich set of possible temporal relationships between the tasks can be defined. This sort of aspect is usually implicit, expressed informally in the output of task analysis. Making the analyst use these operators is a substantial change to normal practice. The reason for this innovation is that, after an informal task analysis, we want designers to clearly express the logical temporal

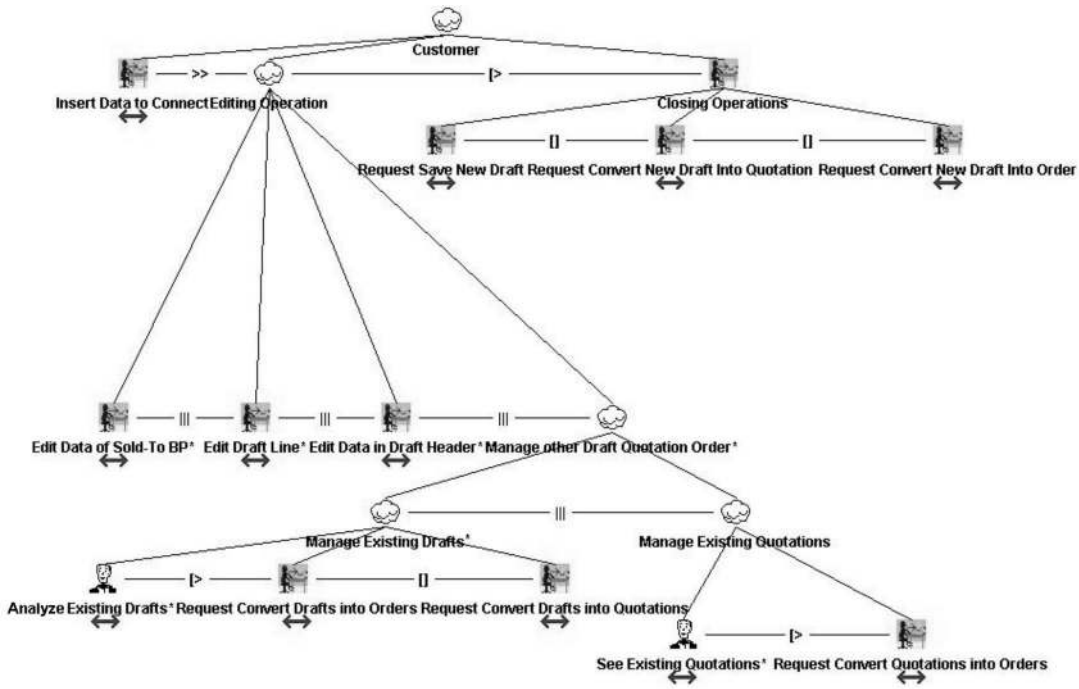

Fig. 1. Example of task model.

relationships. This is because such 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 it is explicitly represented by using icons. There are four possibilities: user task (only internal cognitive activity such as selecting a strategy to solve a problem), application task (only system performance such as generating the results of a query), interaction task (user actions with the possibility of immediate system feedback such as editing a diagram), and abstract tasks (tasks that have subtasks belonging to different categories).
- *Objects*. 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: the user interface objects and the application domain objects. Multiple user interface objects can be associated to a domain objects (for example, temperature can be represented by a bar-chart of a textual value).

The resulting notation allows designers to describe a greater range of behaviors than those that can be described by Hierarchical Task Analysis or GOMS. Richer sets of operators were introduced in UAN, but this notation supports only asynchronous tasks, whereas, we also give the possibility of describing how tasks exchange information. In addition, UAN is a textual notation with no tool support, thus unsuitable for developing and analyzing large specifications. In Fig. 1, there is an excerpt of a task model. We consider the tasks associated to a customer in an ERP application. At the beginning, the customer can insert data to connect. After that (>> is the enabling operator) he can perform some editing operations. This activity can be interrupted ([> is the disabling operator) by closing operations. The operations that the customer can perform are editing customer data, editing draft lines, editing data in the header, and managing other operations. Most of these tasks can be performed repeatedly (* is the iterative operator) and can be performed in any order (||| is the operator for concurrent performance). The *Manage other Draft Quotation Order* task can be decomposed to describe the possibility of requesting conversion of a draft into an order or quotation or a quotation into an order.

The tasks inherit the temporal constraints of the parent tasks. For example, in Fig. 1, the *Closing Operations* task disables the *Editing Operation* and all its subtasks. When sequential tasks are considered, the temporal evolution of the task model can be followed by reading the model from left to right. For example, *Editing Operation* is performed after *Insert Data to Connect*. However, iterative tasks (those indicated with the * symbol) allow going back within the scope of their subtasks (for example, *Manage Existing Drafts* is iterative and, thus, once one occurrence is terminated) by performing either *Request Convert Drafts into Orders* or *Request Convert Drafts into Quotations,* ([] is the choice operator), then its first subtask, *Analyze Existing Drafts*, is enabled again.

The priority order among operators is: choice ([] operator) > parallel composition (||| and |[]| operators) > disabling ([> operator) > suspend-resume (|> operator) > order independence (|=| operator) > enabling (>> and []>> operators).

An extended description of the temporal operators can be found in the appendix (Table 1).

3.1 Description of Single Task

For each single task, it is possible to directly specify a number of attributes and related information. They are classified into three sections (Fig. 2 shows how CTTE supports access to each of them):

- *General information*. It includes the identifier and extended name of the task, its category and type, frequency of use, some informal annotation that the designer may want to store, indication of possible preconditions, and whether it is an iterative, optional, or connection task. While the category of a task indicates the allocation of its performance, the type of a task allows designers to group tasks, depending on their semantics. Each category has its own types of tasks. In the interaction category, examples of task types are selection (the task allows the user to select a piece of information), control (the task allows the user to trigger a control event that can activate a functionality), and editing (the task allows the user to enter a value). This classification is useful to drive the choice of the interaction or presentation techniques more suitable to support the task performance. Frequency of use is another useful information because the interaction techniques associated with more frequent tasks need to be better highlighted to obtain an efficient user interface.
- *Objects*. It is possible to indicate the objects that have to be manipulated to perform a task. Objects can be either user interface or domain application objects. It is also possible to indicate the access rights of the user to manipulate the objects. In multiuser applications, different users may have different access rights.
- *Time performance*. It is also possible to indicate estimated time of performance (including a distinction among minimal, maximal, and average time of performance) to allow some performance evaluation of tasks.

3.2 Representation of Multiuser Cooperation in ConcurTaskTrees

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 multiuser 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 salesman. Cooperative tasks are represented by a specific icon with two persons interacting with each other.

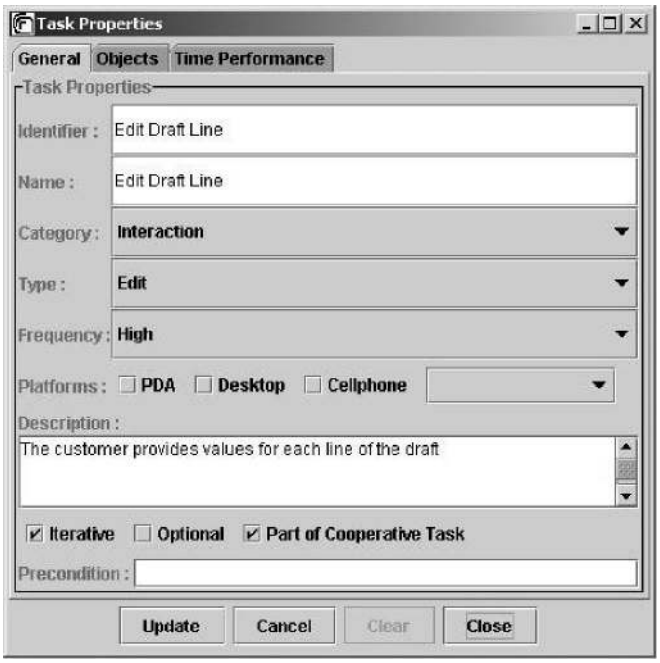

Fig. 2. Template to provide information concerning a task.

In the cooperative part, cooperative tasks are decomposed until we reach tasks performed by a single user that 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, for example, Fig. 3, top part), we can identify connection tasks because they are annotated by a double arrow under their names.

The effect of this structure of representation is that, in order to understand whether a task is enabled to be performed, we have 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 regarding the single user part is not enabled because there is a constraint in the cooperative part indicating that another user must first perform another task. If we consider the example in Fig. 3, we can see the *Search Product* task performed by a customer and the *Provide products info* task performed by a salesman. 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 bottom part of the figure, we can see that the *Provide products list* task (by the salesman) needs to first 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. Cooperative work combines communication, individual action, and collaboration. Our task models aim to provide

an abstract representation of these aspects. The task model associated with a certain role enables distinguishing both individual tasks and tasks that are related to 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 describing flexible situations in which the performance of the different activities depends on the occurrence of specific conditions. Of course, specifying flexible behaviors implies increasing 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, such as the importance for the current project, resources available, and experience in task modeling.

The tool allows browsing the task model of cooperative applications in different ways. The designer can interactively select the frame associated with the part of the task model of interest (in Fig. 4, there is an example with a task model 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 then possible to automatically visualize where they appear in the cooperative part and vice versa.

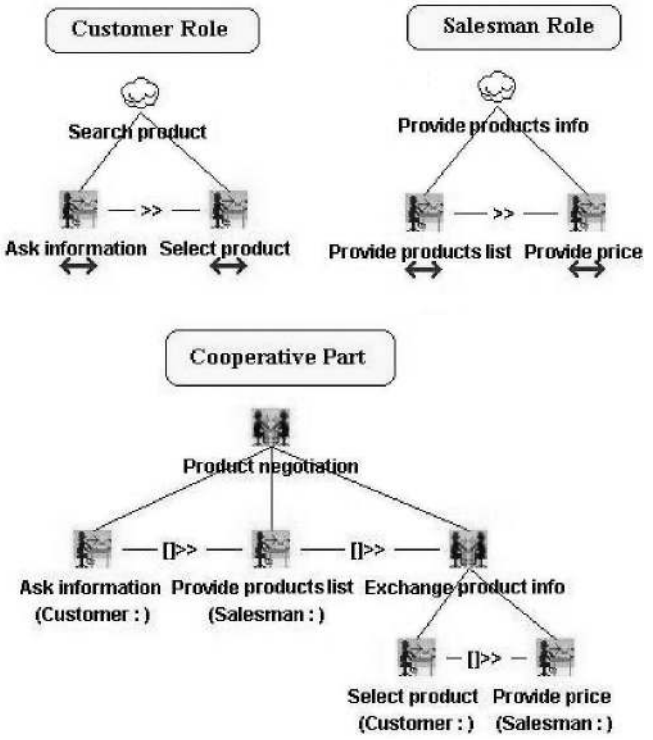

Fig. 3. Simplified example of cooperative task model.

4 How the Tool Supports the Editing and Analysis of Task Models

The tool provides a large number of possibilities for supporting easy development and analysis of task models, the richest currently available. It leverages the ConcurTaskTrees notation, which allows designers to represent the structure of the task model in a hierarchical way, which has been shown to be an intuitive and modular way (one part for each role plus the cooperative part). In addition, the tool supports different views of the task model (complete view, summary view, view by levels, folding and unfolding of subtrees, etc.). Moreover, the task model's structure can be easily modified with the additional support of an interactive simulator that allows designers to better understand the dynamic behavior of the model specified.

In this section, we describe the features of the tool, how they are supported, and the motivations underlying their inclusion. Section 5 is dedicated to the simulator: its functionality and underlying algorithm.

4.1 Editing of Task Models

New tasks are added by selecting the category icon; depending on the selection mode, the new task is added either as the last right child or as a left sibling of the current selected task. It is possible to line up a group of tasks, to change the distance between two levels of the tree, to automatically change the layout to avoid overlapping nodes.

The tool allows for the easy copying and pasting of entire subtrees. Nodes, including whole subtrees, can be disconnected and reconnected arbitrarily, so it is easy to restructure the tree. It is possible to construct a library of patterns associated with subtrees for common tasks which the analyst can reuse in future models. It is possible to interactively select a node and expand or collapse the related subtree. Folding a subtree can be useful when designers do not want to be disturbed by too many details.

The tool is also able to show only a specific number of levels of the task model that is interactively indicated by the user. When the model extends beyond the bounds of the initial window, the tool automatically enables vertical and horizontal scrollbars to allow the designer to still access the entire model.

The tool also supports automatic search of a task in the graphical representation of the model.

4.2 Multiple Interactive Views of the Specification

One usability problem often occurring with tools graphically representing models of realistic systems is that such models do not fit in a window. The usual solution is to include scroll-bars to navigate in the representation. These are useful but often not sufficient to support designers in analysing such models. Thus, to better support the analysis of large specifications, we added an overview window (top right part in Fig. 4), where the logical structure of the task model is represented without reporting details, such as the task names or the icons indicating task allocation performance. In the overview window, the part currently represented in the large window is highlighted by a red rectangle. Thus, designers can have, at the same time, a detailed view on a part of interest in the task model that can be edited by direct manipulation and then have an overview on where such a part is located in the overall structure of the model. It is possible to change the part

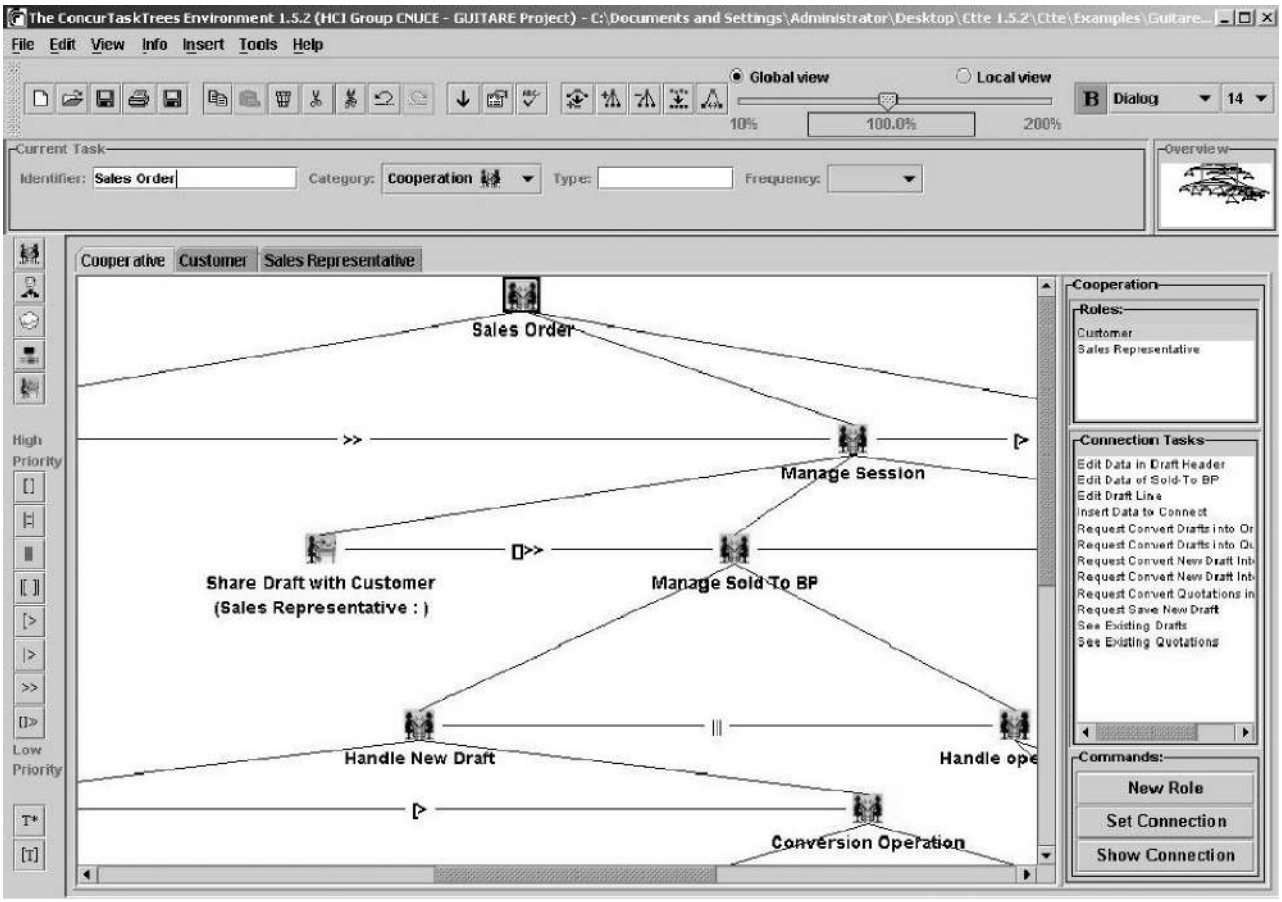

Fig. 4. The editor of task models for cooperative applications.

displayed in the detailed view by just selecting another part in the overview window. In addition, it is possible to zoom the view in or out on the detailed representation and the center of the zooming can be either the center of the specification or a task selected by the designer.

4.3 Editing of Task Models for Cooperative Applications

When cooperative task models are developed, the overall model is structured in one model for each role involved and one part that is associated with modeling the cooperations among users belonging to different roles. There is a tabbed window associated with each part (possible parts are the roles defined and the cooperative part) and, at any time, designers can select which one to bring to the front. To better analyze relationships between cooperative and single user parts, when a single user task in the cooperative part (a connection task) is selected, then the tool automatically shows the task model of the corresponding user and it highlights where it is located within it.

To facilitate the consistent development of the various parts of the task model of a cooperative application when the cooperative part is selected, the designer can select a role from the list on the right side and the tool lists the connection tasks that have been identified for that role (for example, in Fig. 4, the roles are Customer and Sales Representative, Customer has been selected, and, below, the list of related connection tasks is displayed). This allows designers to easily check whether there is any connection task that has not yet been included in the cooperative part and, in that case, they can immediately add it. They just have to create a new task and the name and the role of the connection task will automatically be associated to the new task by selecting the associated connection task.

4.4 Using Informal Descriptions in Supporting Modeling

Often, it is difficult to immediately create a model from scratch. Thus, to support the initial modeling work, we give the possibility of loading an informal textual description of a scenario or a use case and interactively selecting the information of interest for the modeling work. In this way, the designer can first identify tasks, then create a logical hierarchical structure, and, finally, complete the task model.

To develop a task model from an informal textual description, designers first have to identify the different roles (they are in box 2 and they are derived from the part of the scenario highlighted by box 1 in Fig. 5). Then, they can start to analyze the description of the scenario, trying to identify the main tasks that occur in the scenario's description and refer each task to a particular role. For example, as you can see from Fig. 5, the *looking at the taxiways* task has been identified within the scenario's description of an example in the air traffic control domain (see box 3). As this task refers to the ground controller, the

Ground role has been selected in the list of roles and the task has been added to it (see circle 4). It is possible to specify the category of the task in terms of performance allocation. In this case, the task has been considered as a user task. In addition, not only a description of the task can be specified, but also the logical objects used and handled (see circle 5). Reviewing the scenario description, the designer identifies the different tasks and then adds them to the list. This must be performed for each user in the environment.

When each user's main tasks in the scenario have been identified, it might be necessary to make some slight modifications to the newly defined task list. This allows designers to avoid task repetition, refine the task names so as to make them more meaningful, and so on. For example, in Fig. 5, two similar tasks have been identified: the *communicating to the pilot the tower radio frequency* and *issues the instructions to shift to the tower controller radio frequency* tasks (see circle 6-7). They both concern the task of communicating the tower frequency to a departing aircraft that has reached the starting point of the runway, so they can be merged into a single task.

4.5 Checking Completeness of the Specification

There is also automatic support to check that the specification is complete according to the syntax and the semantics of the notation. This means, for example, that the temporal relationships for all tasks are defined as follows: If a task is a leaf in the cooperative part, then it should appear in a single user part and, vice versa, if a task is defined as a connection task in a single user part, then it should appear in the cooperative part. Another control is that each nonbasic task has at least two children because it is not meaningful to have a decomposition into one child.

Fig. 6 shows an example of the results provided by the automatic check. As can be seen, there are errors that stop the user from going any further with the analysis, whereas warnings, which are errors as well (such as a basic task assigned to an abstract category), allow the user to continue with the analysis, for example, by activating the simulator.

4.6 Saving the Specification in Various Formats

At any time, it is possible to save all or parts of the specification and, vice versa, to insert parts of specifications previously saved into the current specification. It is also possible to save all or parts of the specification as jpeg images. This is particularly useful when inserting them in documents, manuals, or reports related to the application that is being considered.

It is also possible to save the task model in XML format. To this end, the DTD format for task models specified by ConcurTaskTrees has been developed. Its purpose is to indicate the syntax for XML expressions that correctly represent task models. This can be useful to facilitate the possibility of analyzing its information from other environments or to build rendering systems able to generate user interfaces for specific platforms using the task model as abstract specification.

Automatic generation of an HTML version of the hierarchical structure of the task model is also supported.

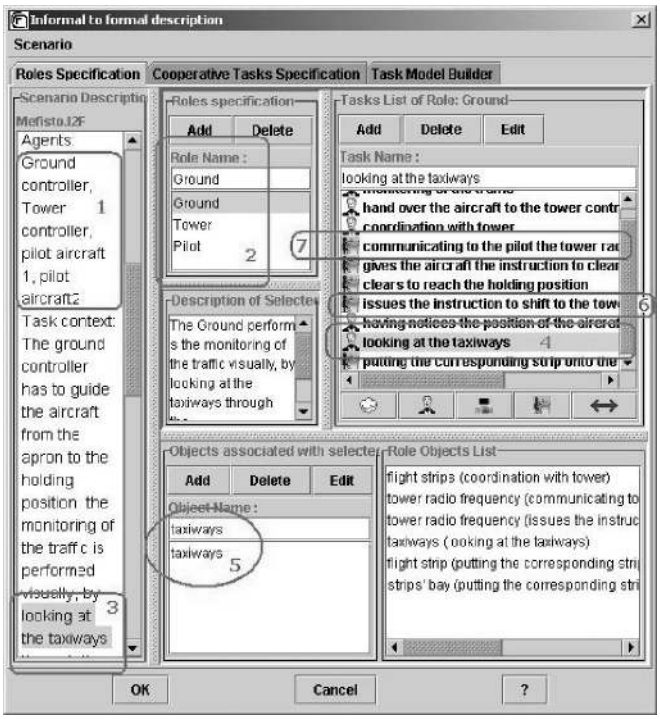

Fig. 5. Moving from informal to formal.

4.7 Comparing Task Models

There is often a need to compare task models. This occurs when people want to compare how people work in the current system and how they could work in a new envisioned system or the designer could be interested in comparing the implications of two alternative designs at task level. Previously, no tool has given this support for task models. CTTE gives some information that can be helpful for this purpose. To be comparable, the two task models should consider the same roles. The comparison is performed in terms of number of tasks, number of basic tasks (the tasks that are no longer decomposed), allocation of tasks, number of instances of temporal operators, and the structure of the task models (number of levels, maximum number of sibling tasks). This information can also be given for single task models in order to analyze them. By comparing this type of information, it is possible to deduce some general features of a solution with respect to another one. For example, a higher number of application tasks and a lower number of user tasks imply that there is a strong shift toward allocating task performance to the system or a higher number of sequential operators implies that the solution supports a higher number of modes in its dialogues with the user.

In Fig. 7, there is an example of comparison; designers have to select which part of the task model they want to compare (the tower controllers, TWR, in the example) and then the result of the comparison of the information relative to that part in the two task models appears. There is also the possibility of activating the presentation of the details related to some parameters. For example, if the details of interaction tasks are selected, then the tool shows the interaction tasks of the selected role that are in one task

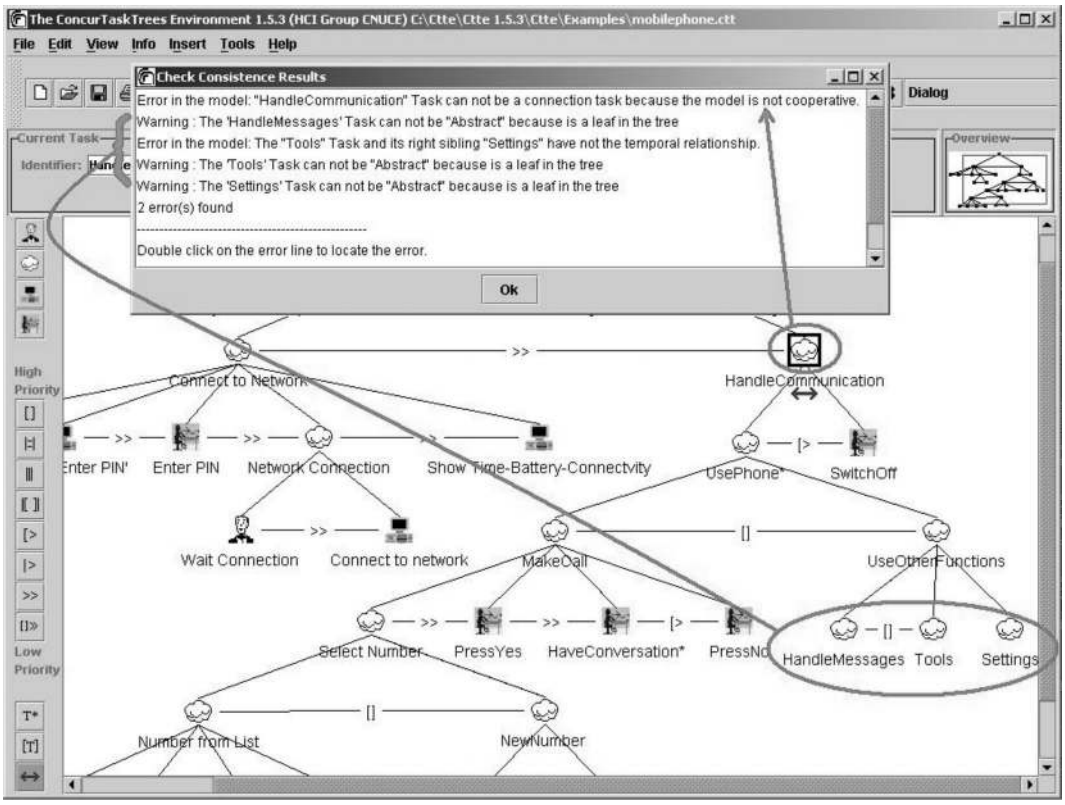

Fig. 6. Example of errors identified while checking specification completeness.

model but not in the other and vice versa (see, for example, Fig. 7, lower part).

We would rather avoid having the tool provide definitive interpretations of these results because they often depend on the type of application considered and features that would characterize a good solution in one application domain may represent a bad solution in another. On the other hand, the automatic analysis highlights specific features of the solutions considered that otherwise would have been difficult to identify, especially when large models of real applications are considered.

4.8 Automatic Expansion of Task Patterns

The tool supports the possibility of automatically expanding task patterns. This means that, if the designer defines the structure of a high-level task in a point of the model and if the same task has to occur somewhere else, then they do not have to again provide all the specification, but it is sufficient to indicate the name of the high-level task. In fact, the tool is able to automatically identify if there are high-level tasks that have been defined in one point and occur in others and then expand them also in the other occurrences (Fig. 8 shows an example). This is not performed in case of recursive tasks in order to avoid infinite expansions. If two tasks have the same name but different categories (task allocation), they are considered different tasks (a search performed by the user or the system is two different activities). If the tool detects that the same task is defined in different ways at different points, it generates an error indicating where it occurs.

4.9 Additional Features

The tool supports additional features such as automatic translation of ConcurTaskTrees specifications into LOTOS specifications. One reason for such a translation is that there are a number of model checking tools (for example, CADP [14]) that can receive LOTOS specifications as input. These tools allow designers to check formal properties, usually expressed in temporal logics, over the specification. Examples of properties that can be verified are reachability properties (once a task has been performed, designers want to know whether it will be possible to perform another task). With our approach, it is also possible to consider properties of multiuser interactions (such as mutual awareness whenever one user performs an interaction the other users receive feedback).

Another possibility of the tool is to automatically calculate enabled task sets for each level of the task tree. Each set is composed of tasks that are enabled over the same period of time according to the constraints indicated in the model. This information is useful during user interface design because the interaction techniques supporting the tasks belonging to the same enabled task set should often be part of the same presentation. Such enabled task sets can be also saved in XML format.

5 The Simulator

5.1 Motivations and Overall Description

A simulator for task models can be useful to better analyze the dynamic behavior of task models, including those for cooperative applications. This is a support that only a few

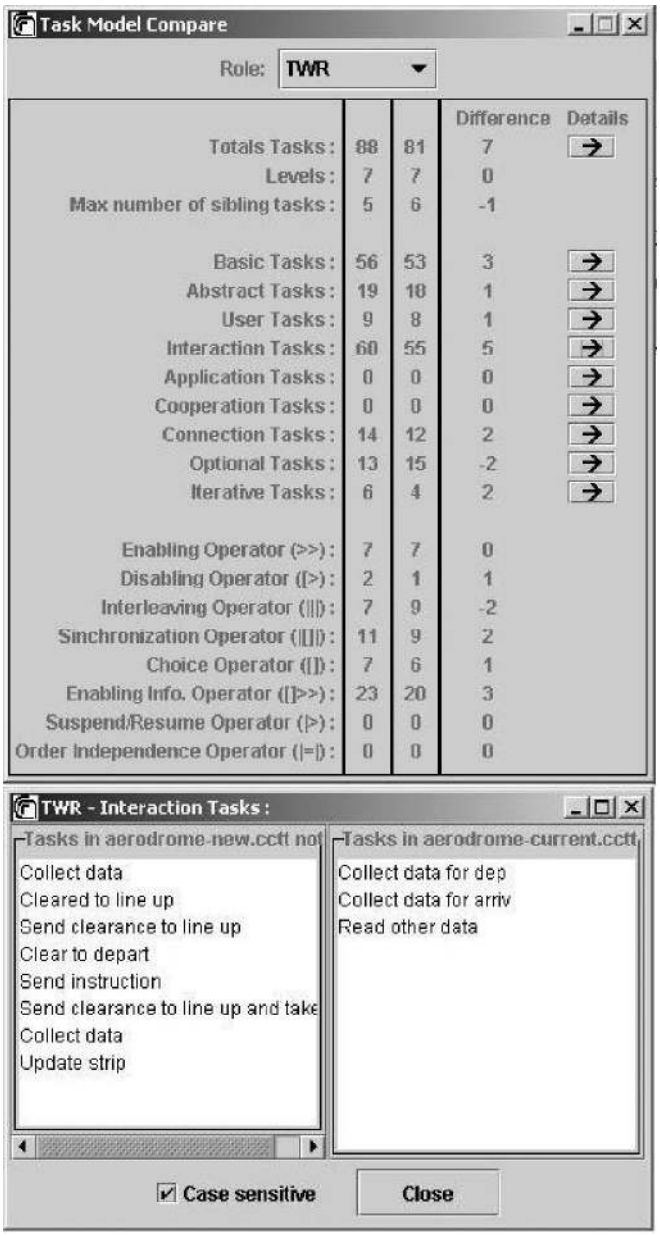

Fig. 7. Example of comparison of task models.

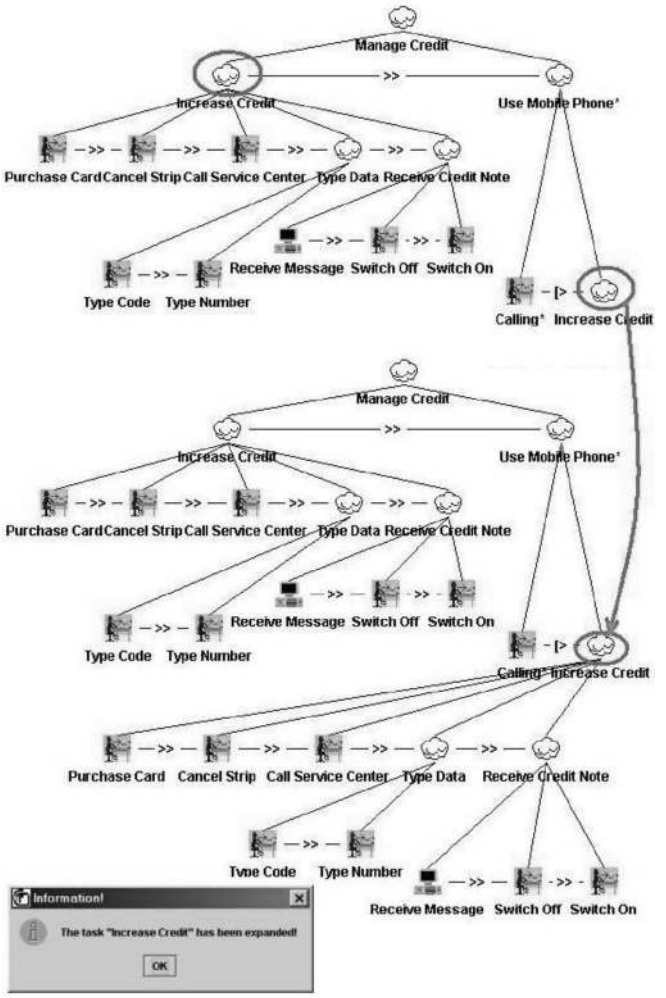

Fig. 8. An example of task pattern automatically identified and expanded.

tools provide [2]. Also, in the case of tools for UML, this is a feature usually missing. In addition, CTTE gives the possibility of simulating task models of applications where the resulting behavior depends on the interactions of various users.

When analyzing an existing application or designing a new one, it can be rather difficult for the designer to understand the dynamic behavior resulting from the temporal relationships specified in the task model. The reason is that, especially for real applications, the number of ways in which the application can evolve is high and it is difficult to mentally remember the various temporal constraints among tasks and their possible effects. It becomes important to support a what-if analysis aimed at identifying what tasks are logically enabled if one specific task is performed. To support this analysis of the dynamic behavior of task models, interactive simulators can be helpful. The basic idea is that, at any time, they show the list of enabled tasks according to the constraints specified in the task model. In the list of enabled tasks, only basic tasks, those that are not further decomposed in the model, are considered. Before starting the simulation, the tool automatically checks that the task model is complete and consistent. The designer can interactively select one of them and the simulator shows, after the performance of the selected task, what the enabled tasks are. This interaction can be iteratively performed a number of times. Since we can describe task models for cooperative applications, in the simulation, we can also see the role of the user involved by the performance of the selected task.

When the simulator is started, then the window on the right displays the list of tasks enabled (see Fig. 9). The tasks that appear in such a list are basic tasks, tasks that are not further decomposed in the model. They are grouped according to the role to which they are assigned. In addition, the tasks that are part of cooperative tasks are listed again under the *Cooperations* label. The enabled tasks are also highlighted in the task model by a green frame.

Then, the user can interactively select a task to perform and the simulator shows what the next enabled tasks are. At any time, it is possible to go back through the performance

of the tasks, which means that the effects of the performance of the last task are undone and the list of enabled tasks becomes the same as that previous to the performance of the last task. At this point, the user can choose to go further backward in the task sequence or forward, either through the same path or a different one.

At any time, the designer can also display the specific sequence of tasks that has been performed in the current simulation (see, for example, Fig. 10). They appear with an indication of the role of the user that performs the task. This is a way to interactively identify an abstract scenario that can be saved in a file and used to compare different task models. The tool is able to load a scenario created with another task model in order to simulate performance of the same sequence of tasks. If this is not possible, either because a task is not supported in the other model or because the temporal relationships in that task model do not allow such a sequence, then it means that the scenario is not supported. This can be useful to compare the task model of an existing application with that of an envisioned one or two different task models that are related to two different designs of the same application. Further possibilities are supported when loading a previously created scenario, such as extending it by adding further tasks or exploring variants of that scenario by performing it partially and then choosing different possible evolutions.

The simulator has shown to be useful in several cases:

- Designers can check whether the specified behavior is really what they intended to describe. This is important because, especially in the case of large specifications, it is difficult to immediately understand the overall behavior deriving from the combination of the hierarchical structure and the temporal operators.
- It can support a multidisciplinary discussion where people with different backgrounds (designers, software developers, end users) discuss design decision at the task level.
- It can be employed as interactive documentation of an application to explain to end users how to use it (indicating in which order tasks can be performed, possible choices, and other dynamic information).

5.2 Underlying Algorithm

The inputs for the simulator are the task model and the semantics of the temporal operators. The formal semantics of each temporal operator was specified using labeled transition systems.

The first step aims to build the data structures required by the simulator. The purpose is to create a binary tree corresponding to the structure of the model according to the operator priorities. For each part of the task model (each role and the cooperative part), there is a top-down visit of the corresponding tree. During this visit, a new data structure is created that stores the corresponding binary tree identified by following the priorities among operators and going from left to right when multiple instances of the same operator are found.

Next, the main goal of the simulator is to identify the set of enabled tasks at any time. Each task is associated with a unique key. Also, each part of the task model has an associated, unique index (*indexRole* variable). The method *BottomUp*(*exp*, *indexRole*) has to identify the enabled tasks where *exp* is either a basic task or a high-level task. To this end, two steps are performed: When the simulator is started, all the basic tasks for each part of the model are analyzed and the method *BottomUp* is applied to them. Then, during the interactive simulation, when the user selects a task, the simulator then checks what the next enabled tasks are by applying the method *BottomUp* to each basic task.

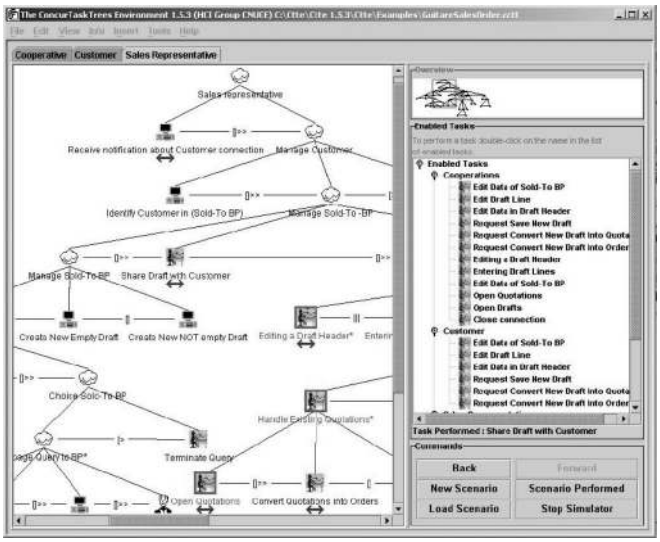

Fig. 9. The interactive simulator of task models.

The method *BottomUp* first checks whether the task in question is a connection task. If so, then it has to call the method *BottomUpEnabled* on both the cooperative part and the part of the model corresponding to the role to which that task belongs (in the case of a nonconnection task, only the latter needs to be performed). In turn, the *BottomUpEnabled* method can use three other methods: *DownHard,* which returns true if the task has been completely performed; *DownSoft,* which returns true if it is a high-level task with at least one of its subtasks already performed or if it is a basic task that has already been performed; *DownSoftHard,* which checks if the task is part of an order-independent expression and returns true if the corresponding expression has been performed and the other one not at all.

The *BottomUpEnabled* method performs a bottom-up visit of the binary tree version of the model. For each task, it analyzes what the corresponding temporal operator is. Depending on the operator, it first checks whether it is the left or the righthand part of the expression and, then, depending on the result, it starts a specific analysis that aims to identify if the parent tasks and the task currently under analysis are enabled.

The type of analysis depends on the type of operator associated with the task and its formal semantics; in all cases, the method first checks recursively whether all the ancestor tasks are enabled:

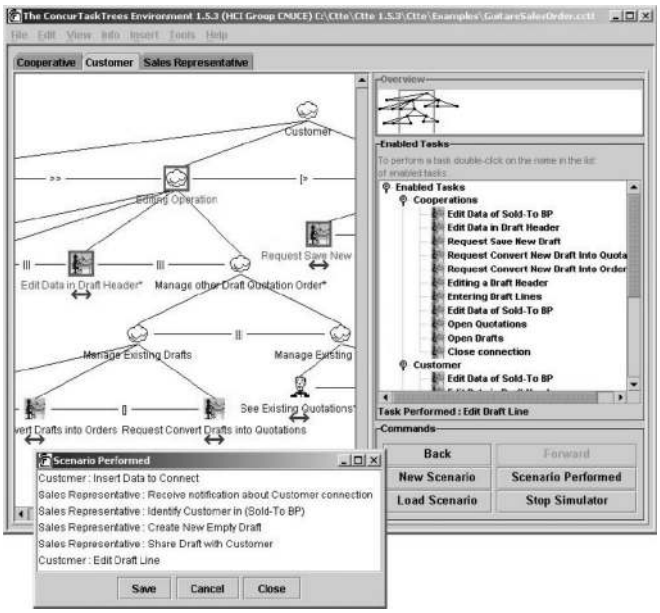

Fig. 10. Example of scenario derived from simulation.

- In the case of the choice operator, it also checks that performance of the other part of the choice has not started.
- In the case of order independence, it also checks whether performance of the other part has either already been completed or has not yet started.
- In the case of interleaving, there is no additional check.
- In the case of disabling, if it is the left part of the operator, then it checks that the right part has not been started, whereas, if it is the right part, then the left part must not be completed.
- In the case of suspend-resume, if it is the left part, then either the right part should not have started or the right part should be completed, whereas, if it is the right part, then the left part should not be completed.
- In the case of enabling, if it is the left part, then performance of the right part should not have started, whereas, if it is the right part, then the left part should be completely performed.

Before starting the analysis of the enabled tasks, the tool analyzes the list of iterative tasks, then checks which of them have completely performed one iteration and, last, removes the associated basic tasks from the list of performed basic tasks so that a new iteration can start.

Optional tasks are analyzed by the *DownHard* method and, in these cases, the tool checks whether at least one subtask has been performed, then it is treated like the other tasks; whereas, if it has not been performed at all, then it is treated as if it were completed and the next tasks at the same level are thereby enabled. By way of example, suppose we have an optional high-level task A followed by task B. At the beginning, both are enabled and this means that the tool treats A as if it had been performed; whereas, if A has already started, then it must complete in order to enable task B again.

If it is possible to reach the root of the tree in this bottom-up analysis, then the task is enabled (Fig. 11 summarizes the algorithm of the *BottomUpEnabled* method). If the user elects to have a task performed, this task is added to an array of executed basic tasks.

5.3 A Small Example of Simulator Analysis

In order to better explain how the simulator works, we can consider a small example. Fig. 12 shows a task model where various temporal relationships are included. The tool first transforms it into the corresponding binary version (see Fig. 13).

Suppose that the following tasks have been performed: *Connection to the site*, *Show Home Page*, and *Type Name* and the simulator has to check whether *Close session* is enabled.

Close session is a subtask of *D0* and it is in its righthand part. Thus, the simulator checks whether *D0* is enabled through the *BottomUp* method and that execution of the left part of *D0* (*Search*) has not completed (see the branch of the algorithm related to disabling operator). The second condition is true because *Search* has not been completely performed. Thus, only the first condition has to be checked. *D0* is the righthand part of *Access Virtual Museum*, thus the simulator has to check if *Access Virtual Museum* is enabled and that its left part (*E0*) has been performed. The second condition is true because *Connection to the site* and *Show Home Page* have both been performed. Thus, we only need to check that *Access Virtual Museum* is enabled. However, *Access Virtual Museum* is the root and reaching it is the condition sufficient to consider the *Close session* task enabled that is, thus, inserted into the list of enabled tasks.

If we consider another task, *Return to Home Page*, the result is different. It is the right part of *Search*. Thus, we need to check that both *Search* is enabled and that the left part of *Search* has been performed by using the *DownHard* method. However, it can be immediately detected that *E4* has not been completely performed because the subtree *Select parameter* has not been completed and the basic tasks, *Send Request*, *Show result*, and *Analyse result*, that are composed through a sequential operator, have not been performed. Thus, *Return to Home Page* is not enabled.

6 Applications of CTTE and Experiences of Use

The tool has been used in a number of projects and at several universities for teaching purposes.

For example, it was used [24] to support the design of an adaptable web museum application. The application provided different ways to navigate and present information, depending on the current user model (tourist, expert, student). We developed a task model for each type of user. The task models also shared some portions for tasks that were common to different types of users.

In the MEFISTO project (http://giove.cnuce.cnr.it/mefisto.html), CTTE has been used to model various air traffic control applications and support their design and evaluation. Large specifications including hundreds of tasks were developed. In this project, the tool was proposed to several teams belonging to organizations that design and develop systems for air traffic control; in some cases, the

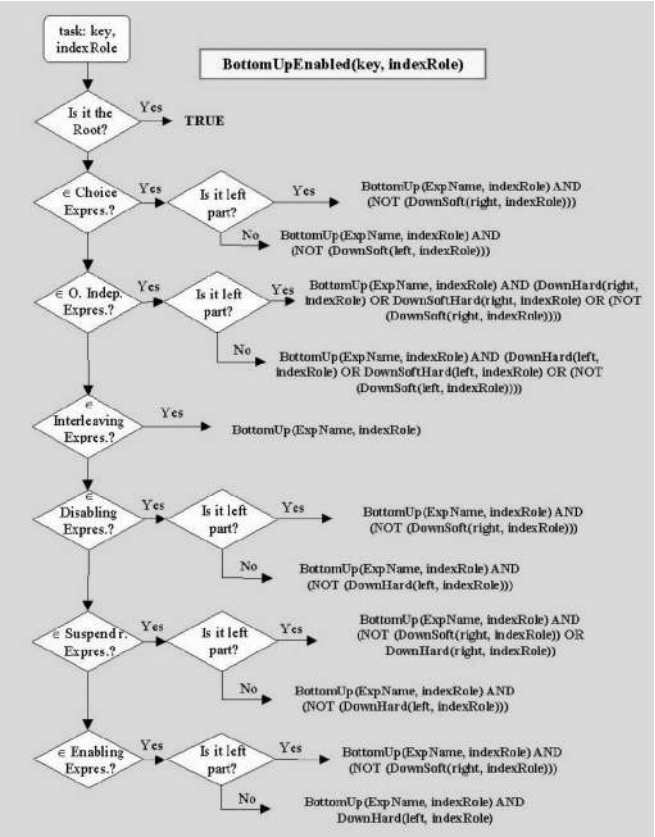

Fig. 11. The algorithm of the *BottomUpEnabled* method.

teams also included people with different backgrounds. One team expressed their appreciation of a tool for task modeling that can be easily downloaded and installed and allow task model-specification at various abstraction levels (including a level independent of the interaction technology considered). However, they found it less easy to understand the exact semantics associated with temporal operators and the development of descriptions for complex multiagent interactions. On another team, the tool was used to develop a task model of an envisioned system from which a number of design recommendations were drawn. The task model was then used in the evaluation of a prototype that was developed. Designers found developing models useful to better understand the problem domain and were stimulated to think more carefully about tasks and their relationships. They found that having some formal background was useful in following the proposed approach, but the notation's hierarchical structure and graphical syntax allowed reducing the designers' efforts to understand the models. One aspect that created problems was when multiple operators appear on the same task level with possible ambiguity in the interpretation of the expression. The notation defines the priorities among operators, thus each expression is associated with a single meaning. However, often designers forget these priorities. Thus, we modified the layout of the tool, arranging the list of operators in descending order of priority, with labels indicating the sorting order. This small modification was useful, but proved to be insufficient. Thus, we added a view of the model (View priority tree) in which, whenever

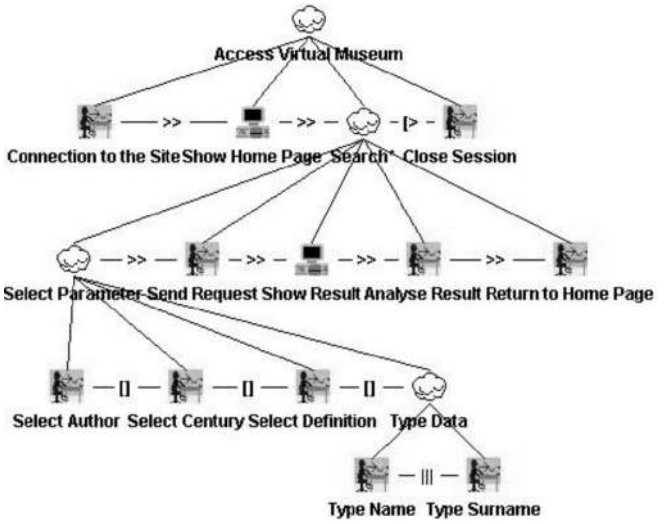

Fig. 12. An example of a task model.

multiple operators are present at the same level, a new task node is included and the lower priority operator is demoted by one level together with its related tasks. Of the features of the tool, those that they liked most were the ability to check whether a scenario is complete according to the task model, if the model is consistent with the notation, and to interactively simulate the performance of scenarios. They also would have liked tool support to move from the task model to the concrete user interface implementation.

More generally, the experience showed that software engineers who were used to working with tools and structured representations during software design and development found it easy to learn and use. People from other backgrounds, such as psychologists or end users (in this case, they were air traffic controllers), had more problems, but this seemed to depend more on the fact that they were not at all used to working with formal representations or systematic methods. However, software designers found it useful to interact with these people and then use the results of the discussion to improve their task models.

At the University of York, an evaluation exercise was developed using a number of techniques (including cognitive dimensions for notations [16] and cooperative evaluation). The trials included computer science students and research associates (thus, people with a background similar to that of industry software engineers). The exercises provided useful feedback and suggestions for small improvements in the usability of the tool: Some icons were changed to better represent the related functionality and some minor functionality was added to further ease editing of the models. Once again, the representations of the temporal operators were not found to be particularly intuitive (we added text tooltips to the icons representing temporal operators to indicate the meaning of the operators). The functionalities that they liked most were the global view of the specification, the automatic support for improving layout of the specification, and the availability of good support for editing the model.

In the GUITARE project, various teams from software companies have used the tool for different application

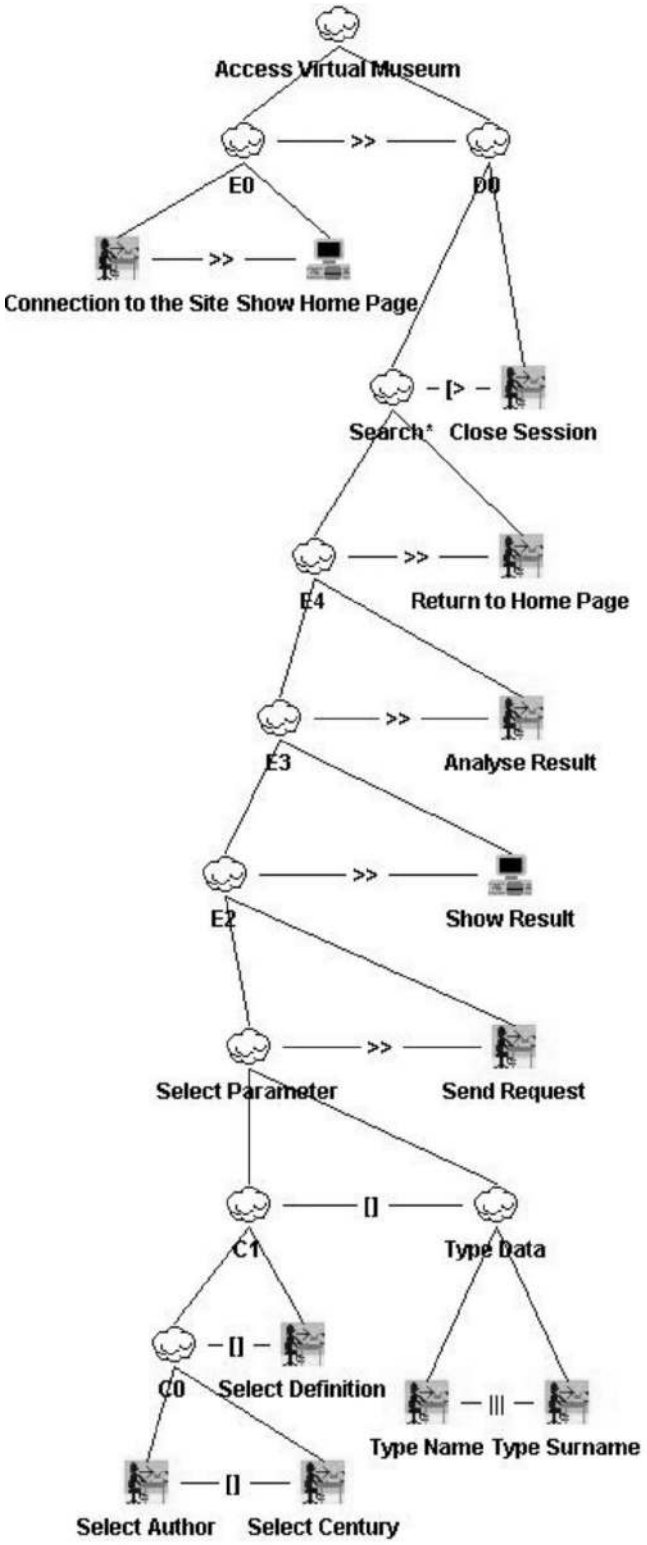

Fig. 13. The binary version of the example.

domains. Some of these teams included people without any background in computer science, who nevertheless were able to use the tool satisfactorily. One of these companies developed a single-user interface generator for ERP (Enterprise Resource Planning) applications starting with ConcurTaskTrees task models. Such a generator also applies organization guidelines to the user interface generation process. Another company was particularly intrigued by the ability to save the task model specification in XML. So, they developed a tool that produces Web-browsable documentation based on the information contained in the task model and a prototype version of a user interface prototyping tool based on the use of task models and related enabled task sets. This team liked the support that the CTTE tool provides for carefully weighing the activities to be performed and their relationships and then refining them until the basic interactions have been described. They too would have liked to have some support for rapid user interface prototyping consistent with the task model earlier in the design process.

Methods have also been developed [19], [26], [11] for supporting user interface design and generation starting with task models specified by CTTE.

7 Summary and Lessons Learned

In the initial part of the paper, we introduced how task models can be useful in the software development life cycle. CTTE is particularly oriented to supporting the design phase because it allows designers to specify and analyze how the activities should be structured and dynamically evolve. The information that is obtained with CTTE can also be used to support usability evaluation through other tools (see, for example, [20]).

Often, the social and time constraints in software design and development result in designers' feeling pressured when adopting model-based approaches. The cost of using models is often criticized. The use of tools strongly decreases such cost. Some application areas (such as interactive safety-critical applications) can better justify such effort. In addition, tools can also facilitate reuse of the specification across various phases, including documentation.

We have seen how CTTE represents an engineered approach to task models, thus overcoming the limitations of previous approaches, which either had no or very limited tool support or used unsuitable notations with limited operators (and, in some cases, no precise semantics). This engineered approach is of particular importance for software engineers who have to develop interactive software systems, often with no systematic support for obtaining effective solutions.

The groups that have thoroughly tested the tool (including ourselves, because, apart from being the designers and developers of the tool, we are also users, as we have used it in various projects) have found it useful in various ways to obtain better task models, especially when the size of the specification becomes larger. It speeds up the process of developing and analyzing models that can be used for evaluation, documentation, and user interface generation. The ability to interactively identify abstract scenarios through the simulator provides useful insights into the dynamic behavior of the model. The identification of metrics concerning the task model and the possibility of comparing two models against them is useful in understanding the characteristic features of a given design from the specification. The types of model representations provided are particularly useful when analyzing large specifications: There is both a global and a local view of the part of the model considered, the model is split into different parts (one for each role and the cooperative part),

and there is support to clearly understand the relationships among the various parts.

In addition, the rich set of temporal operators associated with the hierarchical structure allows designers to describe flexible behaviors of both individual and multiuser activities, taking into account possible interruptions and the various ways in which such behaviors may evolve.

The overall feature set represents a substantial advancement to approaches to task modeling. Thus, ConcurTaskTrees, and the relative tool, can better provide the missing link between modeling a user interface and building/evaluating it.

8 Conclusions and Future Work

We have described CTTE, a tool that provides feature-rich automatic support for developing and analyzing task models. The tool's functionalities give designers useful indications concerning aspects that they should consider when developing interactive software applications.

For this purpose, we have outlined how task models are represented in ConcurTaskTrees and then described how the tool can be used to easily edit and analyze such models in order to gain useful insights, as well as the underlying algorithm of the interactive simulator. Last, we have briefly reported on how the tool has been applied in various projects to indicate its level of maturity.

The tool has been used in various projects for several application areas by many groups in academia and industry. It has been implemented in Java 1.3 beta and has been found useful in clarifying design issues and supporting analysis and evaluation of design options. It is freely available at http://giove.cnuce.cnr.it/ctte.html and has already been downloaded by hundreds of sites all over the world.

Future work will be dedicated to extending the tool in order to obtain an environment for the design of multicontext applications that can be accessed from a wide variety of devices (ranging from cellular phones to large, flat screens) and environments. The new tool will be able to take information from the task model and the description of the target platforms and provide suggestions for implementing corresponding easy-to-use interfaces. The approach will be semiautomatic in order to allow designers to tailor solutions to the specific case studies.

We also plan to integrate this tool with UML environments (such as Rational Rose) in order to obtain more thorough support in designing interactive software systems. In particular, we envision a solution where Use Cases can be used to collect the high-level requirements to be considered in the development of task models. The objects associated with the task model can be integrated with those belonging to a domain model described with UML notations.

Appendix

An extended description of the temporal operators can be found in Table 1.

TABLE 1
Temporal Operators Used in ConcurTaskTrees

Independent Concurrency (T1 ||| T2)	Actions belonging to two tasks can be performed in any order without any specific constraints, for example monitoring a screen and speaking in a microphone;
Choice (T1 [] T2)	It is possible to choose from a set of tasks and, once the choice has been made the task chosen can be performed and other tasks are not available at least until it has been terminated. This operator is useful in the design of user interfaces because it is often important to enable the user to choose from various tasks. An example is, at the beginning of a word processor session when it is possible to choose whether to open an existing file or a new one. Also the system can choose to perform one task from a set of application tasks depending on its current state.
Concurrency with information exchange (T1|[]|T2)	Two tasks can be executed concurrently but they have to synchronise in order to exchange information. For example, a word processor application where editing a file and scrolling its contents can be performed in any order and they exchange information when they are performed because it is possible to edit only the information that the scrolling has made visible.
Order Independence (T1 |=| T2)	Both tasks have to be performed but when one is started then it has to be finished before starting the second one.
Deactivation (T1 [> T2)	The first task is definitively deactivated once the first action of the second task has been performed. This concept is often used in many user interface implementations when the user can deactivate the option of performing a set of tasks and enable a new set of possible task accomplishments by a specific action (for example by selecting a button).
Enabling (T1 >> T2)	In this case one task enables a second one when it terminates, for example, a database where users have first to register and then they can interact with the data.
Enabling with information passing (T1 []>> T2)	In this case task T1 provides some information to task T2 other than enabling it. For example, T1 allows the user to specify a query and T2 provides the query result that obviously depends on the information generated by T1.
Suspend-resume (T1 |> T2)	This operator gives T2 the possibility of interrupting T1 and then when T2 is terminated, T1 can be reactivated from the state reached before the interruption. For example, the editing text task which, in some applications can be suspended by a modal printing task, and once the printing task is accomplished then editing can be carried on from the state reached beforehand. For example, this operator can be used to model a type of interruption.
Iteration T*	In the tasks specification we can have some tasks with the * symbol next to their name. This means that the tasks are performed repetitively: when they terminate, the performance of their actions automatically starts to be executed again from the beginning. This continues until the task is deactivated by another task.
Finite Iteration (T1(n))	It is used when designers know in advance how many times a task will be performed.
Optional tasks ([T])	They give the possibility of indicating that the performance of a task is optional. Optional tasks are indicated in square brackets. For example, we have optional tasks when we fill a form in and there are some fields that are mandatory and others optional.
Recursion	This means that in the subtree originated by the task considered there is another occurrence of it. This possibility is used, for example, with tasks that, for each recursion, allows performance of the recursive tasks with the additional possibility of performing some new tasks, until a task interrupting the recursion is performed.

ACKNOWLEDGMENTS

The authors wish to thank Giulio Ballardin and Riccardo Galiberti for their help in the implementation of an early version of the environment. We gratefully acknowledge support from the European GUITARE project (http://giove.cnuce.cnr.it/guitare.html).

REFERENCES

[1] T. Bolognesi and E. Brinksma, "Introduction to the ISO Specification Language LOTOS," *Computer Network ISDN Systems,* vol. 14, no. 1, pp. 25-59, 1987.
[2] M. Biere, B. Bomsdorf, and G. Szwillus, "Specification and Simulation of Task Models with VTMB," *Proc. Computer-Human Interaction Conf. (CHI '99),* pp. 1-2, 1999.
[3] L. Baumeister, B. John, and M. Byrne, "A Comparison of Tools for Building GOMS Models," *Proc. Computer-Human Interaction Conf. (CHI '00),* pp. 502-509, 2000.
[4] P. Barclay, T. Griffiths, J. McKirfy, N. Paton, R. Cooper, and J. Kennedy, "The Teallach Tool: Using Models for Flexible User Interface Design," *Proc. Int'l Conf. Computer-Aided Design of User Interfaces (CADUI '99),* pp. 139-158, 1999.
[5] F. Bodart, A. Hennerbert, J. Leheureux, and J. Vanderdonckt, "A Model-Based Approach to Presentation: A Continuum from Task Analysis to Prototype," *Proc. Int'l Eurographics Workshop Design, Specification, and Verification of Interactive Systems (DSV-IS '94),* pp. 77-94, 1994.
[6] R. Bastide and P. Palanque, "A Visual and Formal Glue between Application and Interaction," *Int'l J. Visual Language and Computing,* vol. 10, no. 6, pp. 481-507, 1999.
[7] G. Booch, J. Rumbaugh, and I. Jacobson, *Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.
[8] B. Bomsdorf and G. Szwillus, "Tool Support for Task-Based User Interface Design," *Proc. Computer-Human Interaction Conf. (CHI '99),* pp. 169-170, 1999.
[9] S. Card, T. Moran, and A. Newell, *The Psychology of Human-Computer Interaction.* Hillsdale: Lawrence Erlbaum, 1983.
[10] G. Calvary, J. Coutaz, and L. Nigay, "From Single-User Architectural Design to PAC*: A Generic Software Architectural Model for CSCW," *Proc. Computer-Human Interaction Conf. (CHI '97),* pp. 242-249, 1997.
[11] G. Calvary, J. Coutaz, and D. Thevenin, "A Unifying Reference Framework for the Development of Plastic User Interfaces," *Proc. Eng. Human-Computer Interaction Conf. (HCI '01),* pp. 218-238, 2001.
[12] C. Ellis and J. Wainer, "A Conceptual Model of Groupware," *Proc. ACM Conf. Computer Supported Cooperative Work (CSCW'94),* R. Furuta and C. Neuwirth, eds., pp. 79-88, 1994.
[13] J. Foley and N. Sukaviriya, "History, Results, and Bibliography of the User Interface Design Environment (UIDE), an Early Model-Based System for User Interface Design and Development," *Interactive Systems: Design, Specification, Verification,* F. Paterno ed., pp. 3-14, 1994.
[14] J. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu, "CADP (CAESAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox," *Proc. Eighth Conf. Computer-Aided Verification,* pp. 437-440, 1996.
[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.
[16] T. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework," *J. Visual Languages and Computing,* vol. 7, no. 2, pp. 131-174, June 1996.
[17] R. Hartson and P. Gray, "Temporal Aspects of Tasks in the User Action Notation," *Human Computer Interaction,* vol. 7, pp. 1-45, 1992.
[18] N. Leveson, "Intent Specifications: An Approach to Building Human-Centered Specification," *IEEE Trans. Software Eng.,* vol. 28, no. 1, pp. 15-35, Jan. 2000.
[19] Q. Limbourg, B. Ait El Hadj, J. Vanderdonckt, G. Keymolen, and E. Mbaki, "Towards Derivation of Presentation and Dialogue from Models: Preliminary Results," *Proc. Int'l Eurographics Workshop Design, Specification, and Verification of Interactive Systems (DSV-IS '00),* pp. 227-248, 2000.
[20] A. Lecerof and F. Paternò, "Automatic Support for Usability Evaluation," *IEEE Trans. Software Eng.,* vol. 26, no. 10, pp. 863-888, Oct. 1998.
[21] N. Numes and J. Falcao, "Towards a UML Profile for User Interface Development: The Wisdom Approach," *Proc. Unified Modeling Language Conf. (UML '00),* pp. 50-58, 2000.
[22] F. Paternò, *Model-Based Design and Evaluation of Interactive Application.* Springer Verlag, 1999.
[23] F. Paternò, "Towards a UML for Interactive Systems," *Proc. Eng. Human-Computer Interaction Conf. (HCI '01),* pp. 175-185, 2001.
[24] F. Paternò and C. Mancini, "Designing Usable Hypermedia," *Empirical Software Eng.,* vol. 4, no. 1, pp. 11-42, 1999.
[25] A. Puerta, E. Cheng, T. Ou, and J. Min, "MOBILE: User-Centred Interface Building," *Proc. Computer-Human Interaction (CHI '99),* pp. 426-433, 1999.
[26] F. Paternò, C. Santoro, and V. Sabbatino, "Using Information in Task Models to Support Design of Interactive Safety-Critical Applications," *Proc. Advanced Visual Interfaces Int'l Working Conf. (AVI '00),* pp. 120-127, May 2000.
[27] M. Roseman and S. Greenberg, "Building Real-Time Groupware with GroupKit, A Gropware Toolkit," *ACM Trans. Computer-Human Interaction,* vol. 3, no. 1, pp. 66-106, Mar. 1996.
[28] D. Scapin and C. Pierret-Golbreich, "Towards a Method for Task Descrption: MAD," *Proc. Work with Display Unit,* pp. 78-92, 1989.
[29] A. Sutcliffe, "Task-Related Information Analysis," *Int'l J. Human-Computer Studies,* vol. 47, pp. 223-257, 1997.
[30] P. Szekely, P. Luo, and R. Neches, "Beyond Interface Builders: Model-Based Interface Tools," *Proc. Conf. Human Factors and Computing Systems (INTERCHI '93),* pp. 383-390, 1993.
[31] G. van der Veer, B. Lenting, and B. Bergevoet, "GTA: Groupware Task Analysis—Modeling Complexity," *Acta Psychologica,* vol. 91, pp. 297-322, 1996.
[32] M. van Welie, G.C. van der Veer, and A. Eliëns, "An Ontology for Task World Models," *Proc. Int'l Eurographics Workshop Design, Specification, and Verification of Interactive Systems (DSV-IS '98),* pp. 57-70, 1998.
[33] S. Wilson, P. Johnson, C. Kelly, J. Cunningham, and P. Markopoulos, "Beyond Hacking: A Model-Based Approach to User Interface Design," *Proc. Eng. Human-Computer Interaction Conf. (HCI '93),* pp. 40-48, 1993.

Giulio Mori received a degree in informatics enginering from the University of Pisa and is a research assistant at the Human-Computer Interaction Group of the National Research Council of Italy, ISTI-CNR working on the design and development of interactive applications.

Fabio Paternò is the head of the Human-Computer Interaction Group of the National Research Council of Italy, ISTI-CNR. He has been the coordinator of a number of European and national projects on user interface-related topics. He is a member of the IFIP TC13 technical committee. He is the President of ACM-SIGCHI, Italy. He has been member of the program committees of the main international HCI conferences. He has published more than 80 papers in refereed international conferences or journals. His main interests are in methods and tools for design and evaluation of usable interactive systems accessible from many types of contexts.

Carmen Santoro received a degree in computer science from the University of Pisa, Italy, and is a research assistant with the Human-Computer Interaction Group of the National Research Council of Italy, ISTI-CNR. Her current research interests are design and development of effective user interfaces for mobile devices.