

Environments for the Construction and Use of Task Models.

Cécile Paris¹, Shijian Lu¹, and Keith Vander Linden²
CSIRO and Calvin College

¹ CSIRO/CMIS, Locked Bag 17, North Ryde, NSW 1670, Australia.
Email: {cecile.paris, shijian.lu}@csiro.au

²Department of Computer Science, Calvin College, Grand Rapids, MI 49546, USA.
Email: kvlinden@calvin.edu

Task models are useful because they support the design and evaluation of interactive systems from the point of view of the intended users. Unfortunately, they are also difficult to build. To help alleviate this problem, tools have been built that aid in the construction of task models in a variety of ways, in the hope that designers will be encouraged to build explicit task models and reap the benefits of their use and reuse. This chapter will review the range of tools that have been built, and, as an example, will discuss ISOLDE, an environment that integrates an heterogeneous set of such tools and supports the construction, use, and reuse of task models.

Keywords: Task Models, Task Modelling Tools.

Introduction

Task models are explicit representations of user tasks that can help support certain rigorous forms of task analysis. They are recognised as useful constructs that can be exploited throughout the Software Development Life Cycle (SDLC) (Balbo *et al.*, 2002, this volume), either on their own or as components of full interface design models. For example, designers and implementers of interactive systems can use them to assess the complexity of the tasks that the user is expected to perform, to anticipate the amount of time required to perform the tasks (Card *et al.*, 1983), to simulate the behaviour of the resulting system (e.g., Bomsdorf and Szwillus, 1996), to automatically build interfaces based on the model (e.g., Puerta, 1996), and to generate user-oriented instructions (e.g., Paris *et al.*, 1998). In addition, task models can be used to facilitate communication between the various stake-holders in the design process (e.g., Balbo and Lindley, 1997; Balbo *et al.*, 2002, this volume). Though there are a variety of task modelling languages (see Chapters 4 & 5, this volume), they can all support each of these goals to some degree.

While known to be useful, task models are also known to be difficult to build and to maintain. This is, to some degree, true of informal, textual models, but is particularly true of the formal or semi-formal models required to support the benefits mentioned above. To support these benefits throughout the full SDLC, task models must be built at different levels of abstraction (Balbo *et al.*, 2002, this volume). While it might not be difficult to model a set of anticipated user goals at a high level, it becomes progressively more difficult as more levels of detail are included. Representing the task model for a large interactive system all the way from the highest-level user goals down to the keystroke level could be a daunting task indeed. In addition, there may be

separate task models for different views of the interaction, one, say, from the user's point of view, and another from the system's point of view. Finally, in those cases where models are actually built, it is likely that different notations will be used to represent different aspects of the interaction, thus rendering the resulting model incoherent and difficult to use fully. For example, high-level views of users and their goals may be recorded in one notation (e.g., use-case diagrams), while low-level views of the explicit user interface actions may be recorded in another (e.g., UAN). Taken together, these issues tend to prevent explicit and coherent task models from being built and used extensively in practice (Paris *et al.*, 2001).

Given that task models are useful for analysis but difficult to build, it is desirable to provide tools that aid in their construction and consolidation. Task model editors, for example, can help a task analyst build, manipulate and save explicit task models in some coherent modelling language. Such editors include features specifically designed to support task modelling, and are thus easier to use than more general purpose graphics or text editors. Furthermore, they can produce a coherent representation of the various aspects of the task model that could then be manipulated by other systems (e.g., an XML-based representation). As we will see in the next section, task model editors have indeed been built for a number of task modelling languages.

It is clear, however, that while specialised task model editors might be easier to use than general-purpose editors, they still don't render the construction of formal or semi-formal task models easy. The task analyst still has to construct the model from scratch. It would thus be helpful to provide additional tools that facilitate the extraction of task model information from other sources, say from models or artifacts already built for other purposes in the SDLC, or even from existing prototypes if they are available. Because no single source can provide all the input for the full task model, a set of extraction tools could be developed, each one focussing on a different knowledge source. An environment could then be built in which the output of the extraction tools could be consolidated and represented using a coherent task modelling language.

This chapter argues that the construction and consolidation of task models capable of supporting the benefits listed above require the use of an extensible task-modelling environment, that this environment must be built on top of a uniform modelling language, and that it must integrate a task model editor with an expandable set of heterogeneous task extraction/construction tools. Such an environment would thus provide task analysts with a higher-level interface to a task model editor, while allowing them to leverage information sources already built for other purposes. The more powerful, flexible and yet integrated the tools, the more likely designers are to build models and reap the benefits of using them. Having such an environment may also encourage a greater exploitation of task models, including for unforeseen innovative uses.

The chapter begins with a review of the range of tools that have been built to date, identifying the modelling language they support and their primary source of task knowledge. It then includes an extended discussion of ISOLDE, which is intended to serve as an example of the sort of environment being advocated.

A Review of Task Model Construction and Extraction Tools

A number of task modelling tools have been developed that facilitate the construction of task models, either by providing a dedicated editing tool or by providing tools that extract task knowledge from other sources. Each of these tools is designed to represent the task model in a particular formal or semi-formal modelling language. This section reviews the range of available tools, starting with task model editors. Table 1 summarises this review.

Construction Tool	Description	Language	Environment	Reference
CAT-HCI	Task editor	GOMS	-	Williams, 2000
CRITIQUE	Evaluation Tool	GOMS	-	Hudson <i>et al.</i> , 1999
CTTE	Task editor	CTT	CTTE	Paternò <i>et al.</i> , 1997; 2001
EL-TM	Text analysis	CTT	CTTE	Paternò & Mancini, 1999
RemUSINE	Evaluation tool	CTT	CTTE	Paternò & Ballardini, 1999
Euterpe	Task editor	GTA		van Welie, <i>et al.</i> , 1998
GLEAN3	Task editor	GOMS		Kieras, <i>et al.</i> , 1995
GOMSED	Task editor	GOMS		Wandmacher, 1997
IMAD	Task editor	MAD	Alacie	Gamboa R. <i>et al.</i> , 1997
Mobi-D Editors	Editor(s)	MIMIC	Mobi-D	Puerta, 1998
U-Tel	Text analysis	MIMIC	Mobi-D	Tam, <i>et al.</i> , 1998
QGOMS	Task editor	GOMS		Beard, <i>et al.</i> , 1996
Quantum	Task editor	UAN	Ideal	Hix & Hartson, 1994
TAMOT	Task editor	Diane+	Isolde	Paris, <i>et al.</i> , 2001
Isolde:U2T	UML to task	Diane+	Isolde	Lu, <i>et al.</i> , 1999
Isolde:IR	Event recorder	Diane+	Isolde	Paris, <i>et al.</i> , 2001
Isolde:T2T	Text analysis	Diane+	Isolde	Brasser & VLinden, 2002
VTMB	Task editor	-		Bier, <i>et al.</i> , 1999

Table 1: A Summary of Task Model Editors and Extraction Tools

The most common task modelling tools are the specialised task model editors. These tools support the task analyst, in consultation with other stakeholders in the SDLC, in building task models with features tailored to a specific modelling language. Though different in many ways, these editors all include special features for entering and representing user tasks, rendering them more supportive for task modelling than more general purpose editors like MS Powerpoint, VISIO or standard text editors. Such editors exist for several modelling languages, including CTT (Paternò *et al.*, 1997; 2001), Diane+ (Tarby and Barthet, 1996), GTA (van der Veer *et al.*, 1996), GOMS (Baumeister *et al.*, 2000), MIMIC (Puerta, 1996) and UAN (Hix and Hartson, 1994). The special features that they provide tend to center around a particular task modelling language (Bentley and Johnston, 2001). For example, CAT-HCI, GOMSED, QGOMS, and GLEAN3 (see Table 1) support task models written in the GOMS language and thus provide facilities to enter GOMS attributes. Editors have also been designed for other related modelling languages, e.g., UML (Rumbaugh *et al.*, 1999), including use-cases, scenarios and an adaptation of Statecharts (Harel, 1987) and PetriNets (Navarre *et al.*, 2001). Table 1 provides a list of current task model editors (marked as "Task editor"), together with the task modelling language they support. This table indicates the larger environment in which a task editor may be a component. As we will see below, these environments may contain additional tools to aid in the construction of task models.

With the aid of these task model editors, task analysts can build the models that support the range of practical uses in the SDLC discussed above, but they must do so manually. While this is good in that it will most likely force the designer to wrestle with key issues in the design of an interface, it is also clear, unfortunately, that building a complete task model for any reasonably complex device is a daunting enough prospect to prevent designers from doing it in practice, even with the support of a task model editor. This is particularly the case with larger or more complex interactive devices. Researchers have, therefore, sought to develop tools that take advantage of other sources of information in the construction of the model. These sources, which include

written task descriptions, other models of various sorts, and prototype demonstrations, will now be discussed in turn.

One commonly available knowledge source is the written task description, the most informal way of representing a task. These descriptions can come in a number of forms, including task scenarios and use-cases (e.g., Lewis and Rieman, 1994; Constantine and Lockwood, 1999; Jacobson *et al.*, 1992), instructional texts (Carroll, 1990), or other more informal descriptions. These descriptions provide the objects and actions of the domain of the application and also specify the series of interactions between the intended user and the application. When present, therefore, they represent an important source of information from which to build a task model. The advantage of this source is that it is relatively easy to produce. The primary disadvantage is that natural language is too informal to be used rigorously. A task modelling environment can, however, provide support for extracting more formal task information from written task descriptions. Table 1 calls these tools “text analysis” tools. Two of these tools, U-Tel (Tam *et al.*, 1998) and EL-TM (Paternò and Mancini, 1999), support the task analysts as they *manually* identify the objects and actions in a task description and assemble them into tasks and task models in the task model editor. In this approach, the task analysts perform the text analysis. The third text analysis tool, Isolde:T2T (Brasser and Vander Linden, 2002), attempts to extract task model knowledge *automatically* from written text, using a natural language parser to perform the text analysis.

Another viable source of task knowledge is a model represented in any one of a number of other modelling languages. For example, UML (Rumbaugh *et al.*, 1999) is primarily a system modelling language, but it does include some task-oriented information, primarily in the use-case diagrams and potentially in the scenario diagrams and state diagrams. It also includes some domain knowledge in its class diagrams. It is thus possible to extract task modelling knowledge from these diagrams. Table 1 labels such tools as “UML-to-Task” tools. Isolde:U2T is an example of such a system (Lu *et al.*, 1999). These tools can support the construction of task models in situations where other models are created as part of the SDLC. Similarly, a task analyst may be able to consolidate task knowledge from models represented in other task modelling languages (Limbourg *et al.*, 2001).

A final source of task knowledge is a system prototype, when one exists. Prototypes are especially useful in capturing low-level user actions. To extract these actions, the task analyst can hook an existing prototype up to any one of a number of user interface event recorders, and the recorder will then collect GUI events as potential users or designers use the prototype (Hilbert and Redmiles, 2000). Because this approach requires the existence of a running system, it has been more commonly applied to the evaluation of existing user interfaces. For example, RemUSINE (Paternò and Ballardini, 1999) uses an event recorder to support evaluation (see Table 1). It can, however, also be applied to the construction of task models provided that a prototype is available early enough in the design process. Isolde:UIR (Paris *et al.*, 2001) is an example of such a tool (see Table 1). One key issue with these tools is linking the low-level interface events collected by the recorder with higher-level user goals (Hoppe, 1988). This typically requires resorting to other knowledge sources and linking them together using a task editor.

Each of the potential sources of knowledge given above provides different aspects of the required task knowledge and is appropriate in different contexts. For example, objects, actions, and high-level user goals may be obtained from a written task description or a UML use-case diagram, , but not from event recorders. Conversely, low-level interface actions can be obtained more easily by event recorders. It is thus clear that no single source or tool will support the construction of all

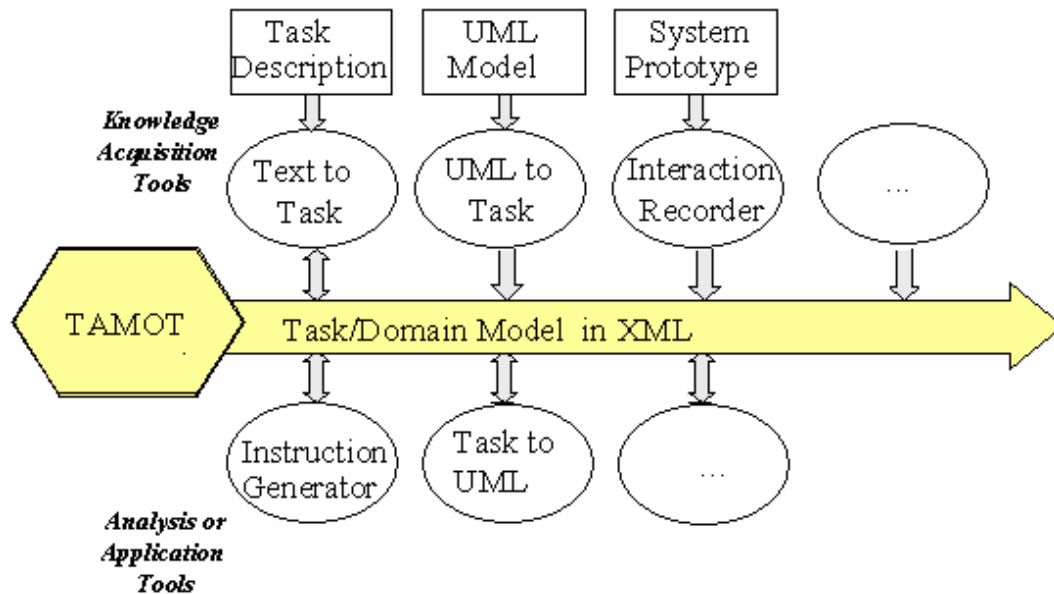


Figure 1: The Isolde Environment

the elements of a complete task model at all levels of abstraction. Thus, it is more appropriate to provide a task modelling environment that integrates an extensible set of heterogeneous tools within a unified framework. This framework would most likely designate a selected task modelling language that would represent information collected from any of the tools supported by the environment. Examples of this sort of environment include CTTE (Paternò *et al.*, 1997; 2001), Mobi-D (Puerta, 1997), and Isolde (Paris *et al.*, 2001) (see Table 1).

The ISOLDE Environment

The previous sections advocated the use of an extensible, heterogeneous environment that provides specialised tools for extracting task information from a variety of sources. The environment should also provide a task model editor that allows a task analyst to integrate, refine and extend the extracted knowledge within a common modelling format, or still build the model from scratch if desired. As an example of this concept, this section presents the Isolde environment. Isolde's architecture, shown in Figure 1, is centred around the Tamot task model editor for the Diane+ modelling language (Tarby and Barthet, 1996) and its underlying XML-based representation language. Isolde includes tools to facilitate the construction of task models (shown on top of the figure as “Knowledge acquisition tools”) and tools that exploit task models (shown at the bottom of the figure as “Analysis or Application tools”).

This chapter now presents the various modules of this environment, concentrating on the tools that facilitate the construction of a task model: the task model editor (Tamot) and three acquisition tools specific to the three knowledge sources mentioned above: written text (Text-to-Task: T2T), other models (UML-to-Task:U2T) and system prototypes (User Interaction Recorder: UIR). The tools are presented in the context of a running example, one showing the use of Tamot itself. Isolde currently also includes two application tools: the Instruction Generator and the Task-to-UML tool. These are presented briefly, only as illustration of how a task model can be exploited in a variety of ways, once it has been built. Importantly, as illustrated in the figure, the environment is extensible, both in terms of the acquisition tools and the application

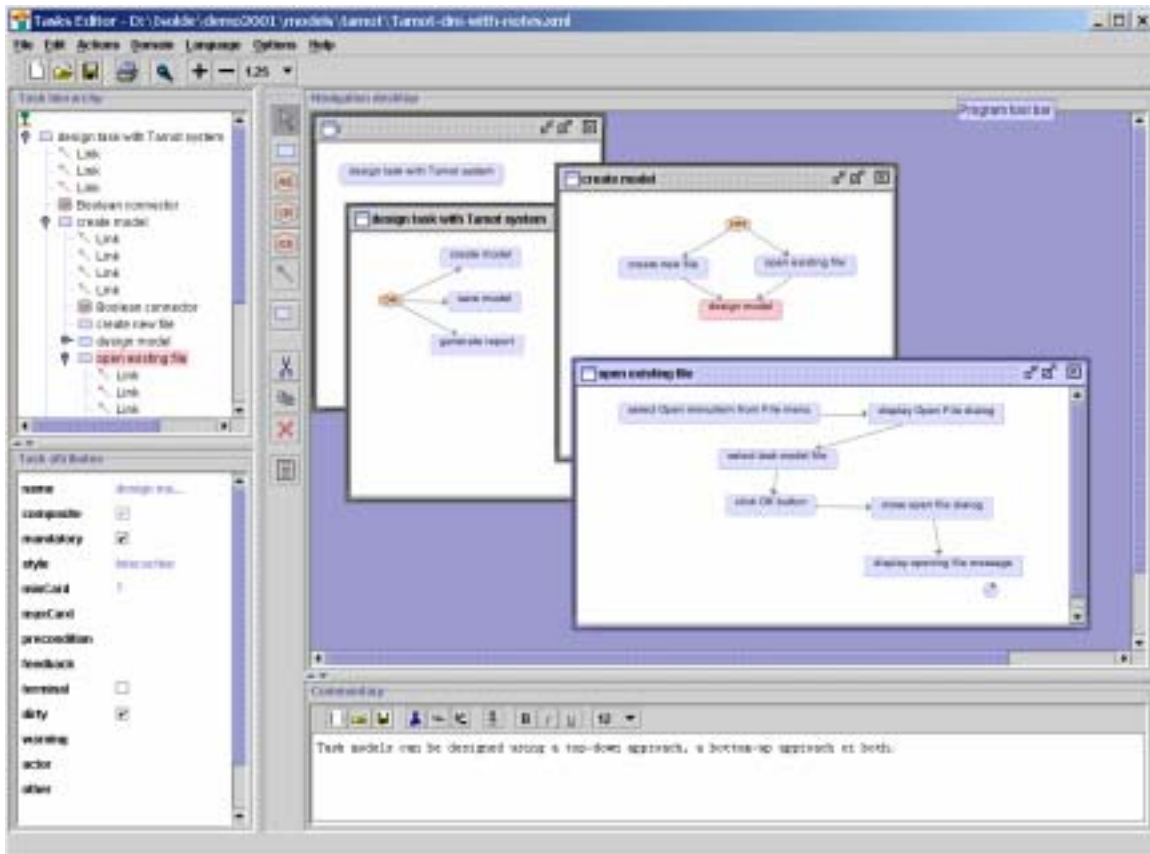


Figure 2: The Tamot Task Model Editor

tools. The only requirement on an extraction tool is to produce a Diane+ model in Isolde's open, standardised XML-based representation. Similarly, an analysis or application tool must be able to take such a representation as input.

The Task Model Model Editor: Tamot

Tamot is a general-purpose task and domain model editor. Its task representation is based on the Diane+ modelling formalism (Tarby and Barthet, 1996), which supports task annotations (e.g., repetition and optionality) and procedural relationships (e.g., goals and sequences). Diane+ was originally designed to support the early phases of the SDLC (e.g., requirement analysis and design), but has recently been proposed for use in all stages of the SDLC (Paris *et al.*, 2001).

Tamot is similar to other graphically based task editors (e.g., Beard *et al.*, 1996; van Welie *et al.*, 1998; Paternò *et al.*, 2001). It enables users to construct a task model manually. Tamot has been used by task analysts at CSIRO, in commercial projects, and it has also been shown to be usable by technical writers (Ozkan *et al.*, 1998). Within the Isolde environment, however, Tamot's primary role is to allow users to consolidate, modify and extend the task knowledge produced by the extraction tools. As such, Tamot reads in and outputs files in Isolde's open, standardised XML-based representation.

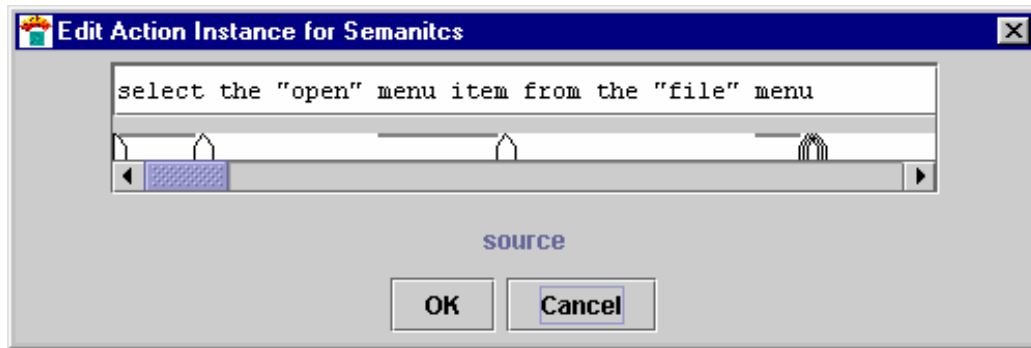


Figure 3: The Interactive T2T Interface

Figure 2 shows Tamot editing a task model. As mentioned earlier, this paper uses a task model for Tamot itself as an example. On the upper left, Tamot displays a hierarchy of user tasks. On the upper right, it displays a set of windows showing the graphical representation of the tasks at various points in the hierarchy. In this example, we see the high-level task of “design a file with Tamot”, which is decomposed into several subtasks, one of which is creating a model. This subtask is further decomposed, and the figure also shows the sequence of primitive actions that occur when performing the task “Open an existing file”: the user must choose the open menu item, select the file, and click the OK button. (User actions are shown as boxes with rounded corners.) The systems responses to these actions are shown as rectangles.

In addition to supporting all the features of the Diane+ language, Tamot includes a number of other features. It supports both bottom-up and top-down approaches to task modelling, and allows users to specify tasks either through a dialog box mechanism or through natural language input. In the natural language mode, the user can type sequences of sentences that are parsed into task information and loaded automatically into the task model as tasks, annotations, and procedural links (see the next section for a discussion of this mechanism). Tamot's representation includes both task knowledge (tasks and procedural relationships) and domain knowledge (e.g., actors, actions, and objects). Finally, Tamot can produce a customisable, html-based report from the task model.

The Text Analysis tool: Text to Task Extraction Tool (T2T)

As mentioned earlier, written texts are a commonly available source of task and domain knowledge. The HCI community has produced tools that allow users to *manually* acquire knowledge from such descriptions (e.g., Tam *et al.*, 1998; Paternò and Mancini, 1999). T2T, our Text-to-Task extraction tool, uses information extraction technology and natural language parsing to *automatically* acquire from the text knowledge that can then be re-used to create a task model task (Brasser and Vander Linden, 2002). This is also the facility used in Tamot to acquire objects and actions from the task names.

The core of T2T is a finite-state grammar built as a 30-state ATN (cf. Jurafsky and Martin, 2000). At the sentence level, this grammar identifies the basic elements mentioned in a text: the objects and actions of tasks and their relationships, in particular: *who* is doing the *action* to *what*, thus providing information about the process (action), its actor (who) and its actee (what). The roles of source, destination, instrument, and location are identified as well. Any remaining elements of the sentence are left as canned strings (e.g., complements or adverbs). T2T distinguishes between nouns and verbs based on the classifications of WordNet (Fellbaum, 1998). Although it can work automatically, it also allows the user to modify any incorrectly assigned constituent boundaries or

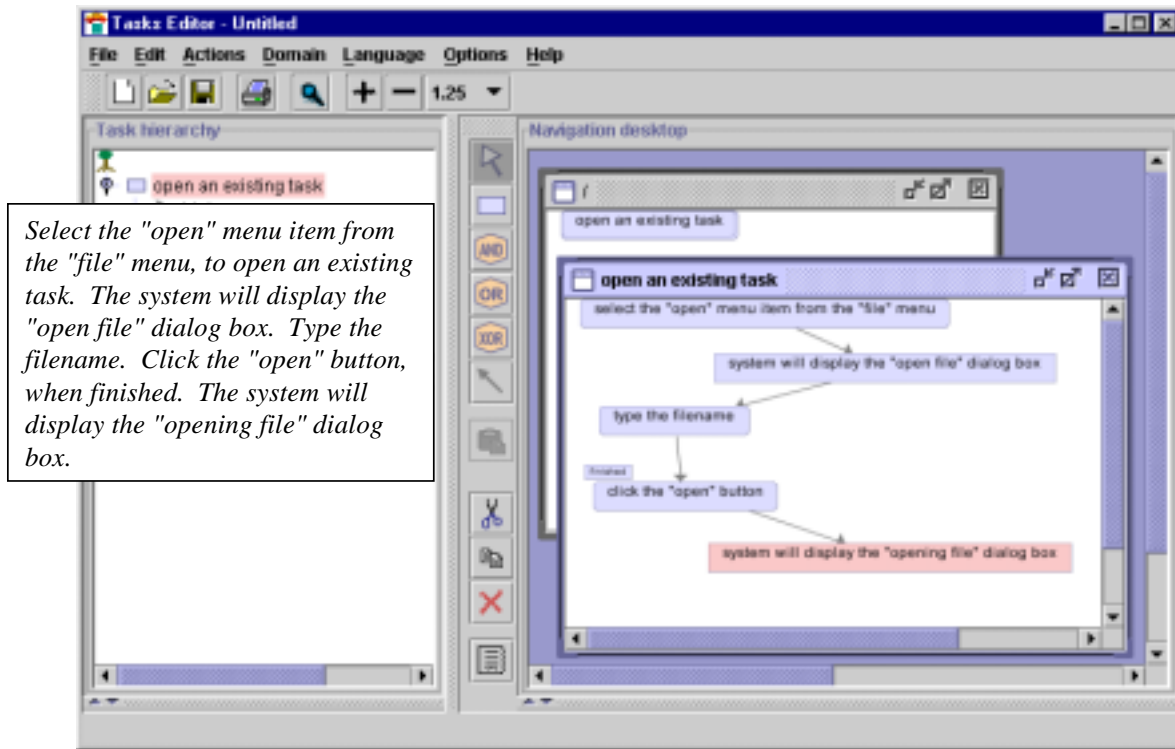


Figure 4: A Task Model Extracted by T2T

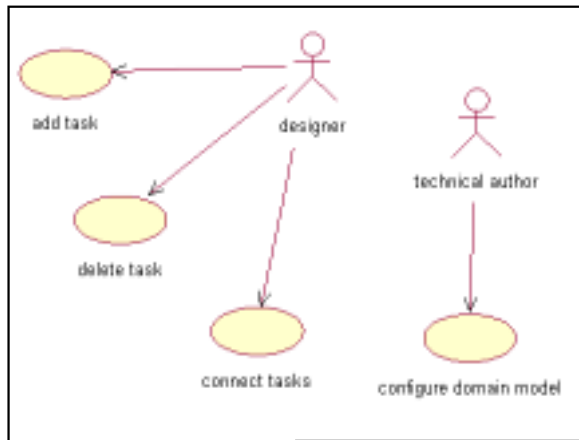
head-word assignments using an *interactive parsing tool*, shown in Figure 3. Here, T2T has identified the sentence's action ("select"), its object ("menu item"), and its source ("menu").

When there are expressions of multiple tasks, as will be the case for a written description (as opposed to a single task name), the grammar extracts the procedural relationships between them (e.g., sequence, preconditions, task/subtask relations). It then creates the appropriate domain knowledge, which includes representations of the actions and of their related objects. Figure 4 shows a sample instructional text and the task model that T2T derives from it. The system has extracted the purpose of the procedure (i.e., "create a sub-task"), the sequence of actions (i.e., "select", "click", "modify" and "click"), the condition (i.e., "when finished"), and the system actions (i.e., "display", "draw", and "close").

The T2T grammar was based on a corpus of 28 task descriptions taken from documentation written for a variety of Software Engineering courses, and was evaluated on an additional 9

	Recall	Precision	Problems
Elementary Task Units	90.1% (118/131)	68.2% (118/173)	Infinitives, "and" and punctuation
Purposes	100% (5/5)	17.8% (5/28)	"If you want to ..."
Preconditions	75% (3/4)	42.8% (3/7)	"First, you must ..."
Domain Entities	47% (155/324)	47.9% (155/323)	Phrasal verbs, anaphora

Table 2: Evaluation of T2T



(a) Use Case Diagram

(b) Class Diagram

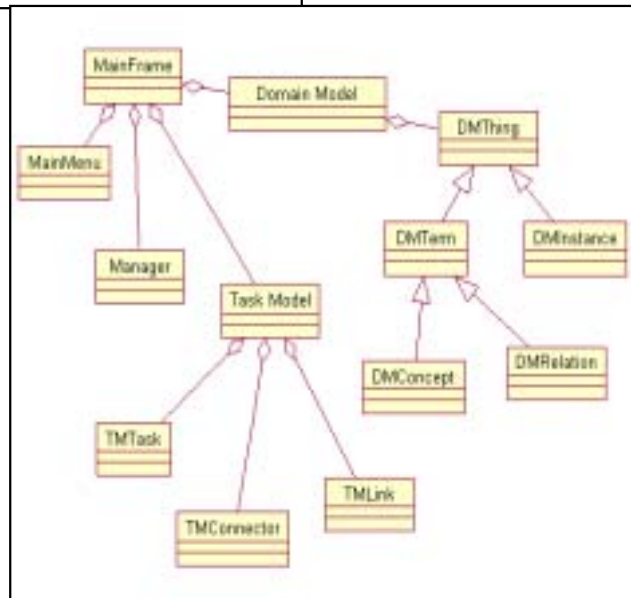


Figure 5: Sources of Task information in UML Models

descriptions, using the traditional *recall* and *precision* measures. These indicate how many of the tasks the system was able to recognise (recall) and the correctness of the tasks it recognised (precision) (cf. Jurafsky and Martin, 2000). The results of this preliminary evaluation are shown in Table 2, together with the problems encountered. Some of these problems are due to the small syntactic coverage of the parser at this point. Others are due to the fact that the parser currently does not attempt to resolve references (e.g., pronouns and anaphora).

The UML to Task Extraction Tool: U2T

Design models represented in the Unified Modelling Language (UML) are common in Software Engineering (Rumbaugh *et al.*, 1999), particularly in OO analysis and design. If they already exist, these models can serve as another source of task knowledge. The UML to Task tool, U2T, performs this extraction using the standard Rational Rose scripting language. It has been discussed in some detail elsewhere (Lu *et al.*, 1999; Vander Linden *et al.*, 2000). We will thus only briefly describe it here.

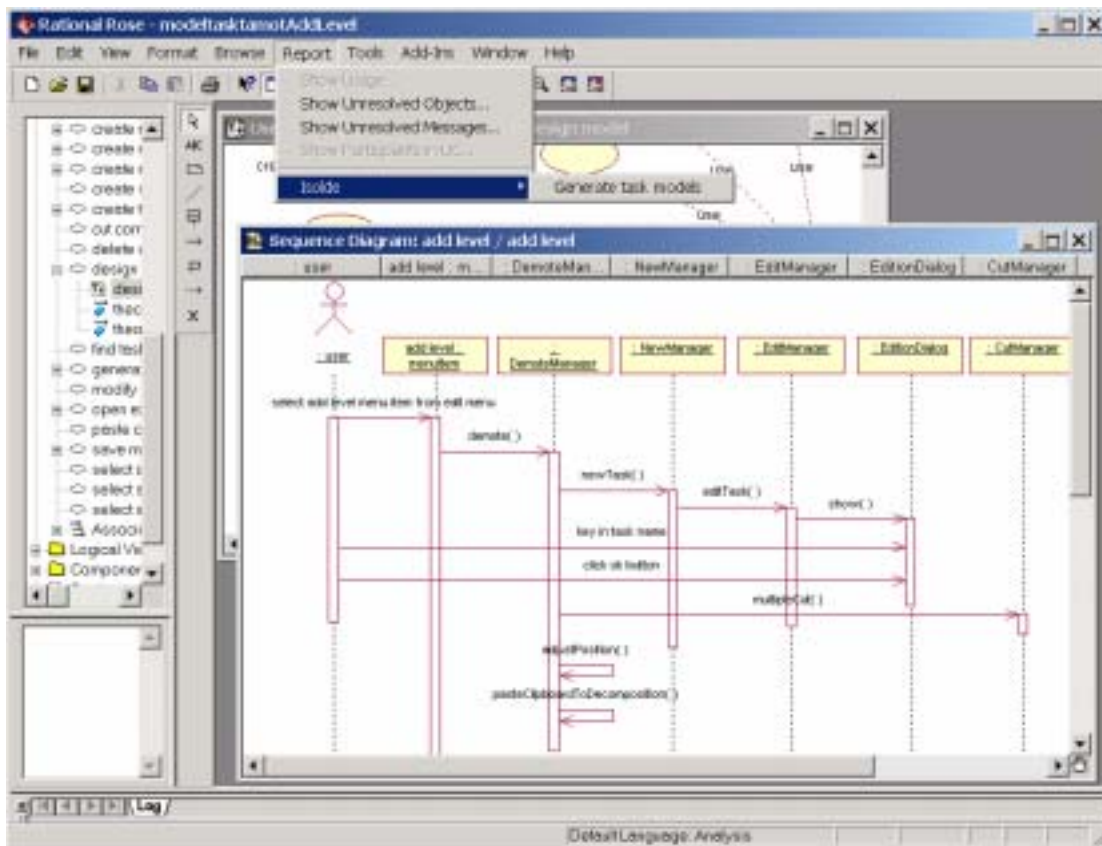


Figure 6: Rational Rose with the U2T extraction tool

UML design models typically include both system structure models, such as class diagrams, and system behaviour models, such as use-case diagrams and scenario/interaction diagrams. These three types of diagrams are the key sources of task knowledge in UML. Examples of them for our Tamot example are shown in Figures 5 and 6. Use-case diagrams (Figure 5(a)) identify the users and their goals with respect to an application. Class diagrams (Figure 5(b)) identify the basic classes of domain objects and the actions that can be performed on them. Interaction diagrams (Figure 6, lower right) are associated with basic use-cases and represent sequences of interactions between objects instantiated from the classes in the class diagram.

Although UML models are system-oriented, they do contain knowledge relevant for the task model. U2T extracts the task-oriented information from UML models based on a set of heuristics derived from an analysis of the common semantic ground between system behaviour models and task models (Lu *et al.*, 1999). In brief, the relationship between the two types of models is as follows:

- Use-cases can be seen as equivalent to composite tasks in a task model.
- The containment of use-cases by other use-case diagrams can be seen as equivalent to the hierarchical structure in a task model.
- Classes and methods in class diagrams can be seen as objects and actions common to the domain of application.
- Messages in scenario diagrams originating from a user object can be seen as user tasks in sequence.
- System-originated messages that operate on a visible system object and follow a user message can be seen as a system response to the user action.

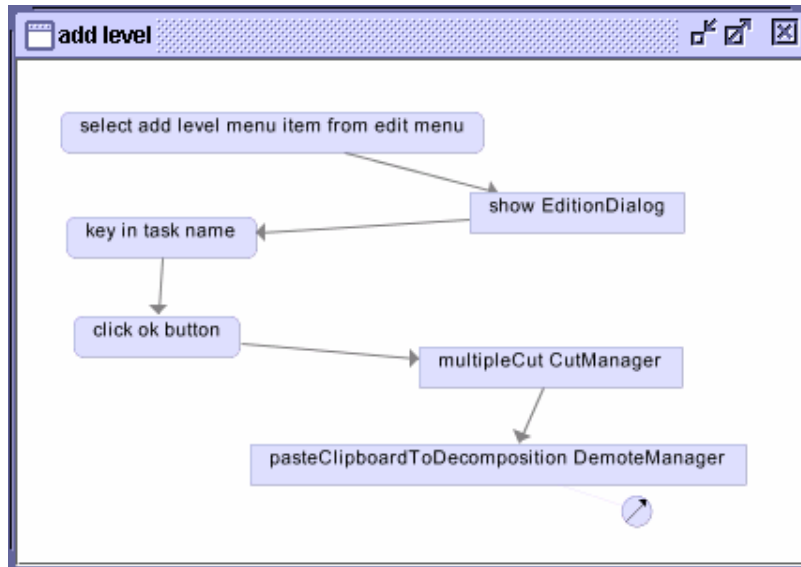


Figure 7: Generated task for “add level”

U2T, which is implemented in Rose script and can be seen as an extension to Ration Rose, extracts task information UML and exports it in Diane+ format to the Tamot task model editor. Figure 7 shows the result of U2T extracting the task model from the scenario diagram for “add level” of Figure 6.

The Event Recorder: User Interaction Recorder (UIR)

System prototypes, if they exist, implement the GUI objects used in their interface and the interaction dialog used to interact with them. Because tools exist that can extract the objects and record the dialog as the prototype runs (Hilbert and Redmiles, 2000), the prototype can also serve as a knowledge source. UIR, our User Interaction Recorder, uses an object extraction tool and an event recording tool, both developed at Sun Microsystems for the Java platform (Sun, 2002), to extract task and domain knowledge.

Figure 8 shows both parts of UIR in operation. The window on the lower right is the user event listener window, and the one on the upper left is the object extractor window. The user event listener shows a hierarchy of user tasks (in the Tasks pane). Because it cannot infer these high-level goals, it allows the user to build them manually or to import them from another source (e.g., an existing task model or UML model). In this example, the user goals include opening, saving and building a task model. The user can record the low-level steps required for each goal by selecting the goal, pressing the record button, and demonstrating the task in the real application (in this case, in Tamot itself). The sequence of events (for opening a task model, in this example) is recorded and shown as a vertical sequence of actor-action-object triples (in the Actions pane). The UIR object extractor, the window on the upper left of Figure 8, extracts the hierarchy of GUI objects used in the interface. We see that the "Tasks" editor contains a "menu bar", which contains the "Actions" menu. This allows UIR to determine the containment hierarchy of GUI objects, supporting expressions like "Select the task icon *from the tool bar*".

The Java tools on which UIR is based collect a wide variety of information, some of which is too low-level to be useful for task or domain modelling. For example, rather than recording an event

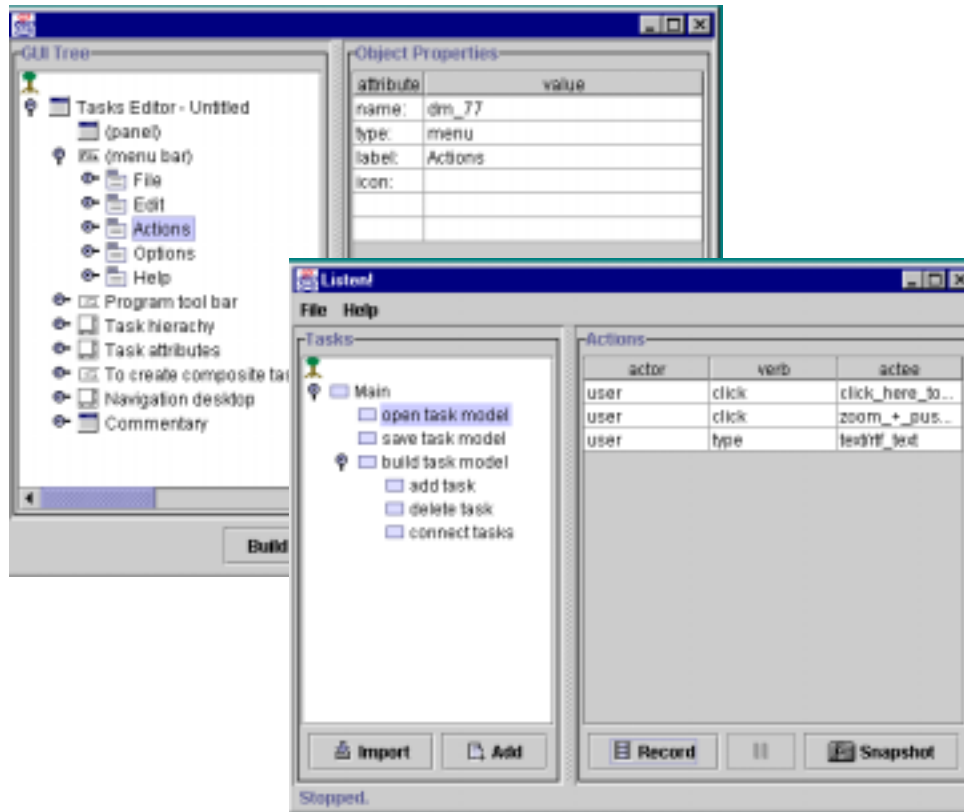


Figure 8: The User Interaction Recorder

like "the user types a filename", the event listener records a separate key-press event for every letter typed into the filename text field. UIR includes heuristics that help it translate from the low-level events and raw widgets it extracts to the higher-level, more domain-oriented actions and objects required by the task and domain models. Any knowledge that is still missing or incorrectly recorded in any way can be added or modified within Tamot.

Isolde: Integrating and enabling

The extraction tools and the task model editor described above, taken together, form an environment in which task knowledge from a variety of sources and in a variety of notations can be constructed, extended, refined, and finally consolidated. The result is a set of unified models, all represented in one modelling language that is stored in an XML-based notation. The environment is flexible in that it caters to a variety of situations, depending on what is most appropriate. The task modeller may build the task model by hand, extract task information from any of the sources discussed above, or any combination of these approaches. Importantly, the environment is also open in that new extraction tools can be added to the environment provided that they support to the XML-based representation.

Once task models are defined in the environment, they are ready to be exploited for a variety of purposes. This use and potential re-use is enabled in the Isolde environment by a set of application or analysis tools. The Isolde environment currently includes two such tools (see Figure 1), which will be discussed briefly here. More details on both of these tools can be found elsewhere.

The Instructional Text Generator (ITG) uses task models to drive the generation of user-oriented instructions for the task being modelled (Paris *et al.*, 1998). It takes as input a task model with its associated domain knowledge, and generates a draft of a hierarchical, html-based set of instructions. The task model must include both the high-level goals of the user and the low-level actions that achieve those goals. The domain model must include, for each task, a specification of the actor, the action being performed, the object being acted upon (if any), and other relevant information (e.g., location or manner). ITG also uses a number of linguistic resources (e.g., discourse, lexical and grammatical), some in deep representations and others in shallow form, and a natural language generation engine, which includes text planning, sentence planning and realisation. ITG provides one control over the linguistic resources via some *style parameters*, which allow the production of the text to adhere to different styles, at both text and sentence levels. Automating parts of this task is useful because it is widely accepted that writing and maintaining user-oriented documentation is crucial to the usability of a product but also that it is labor-intensive and tedious. A recent evaluation of the text produced, aimed at assessing their effectiveness compared to human-authored text, showed no significant differences between the two. (Colineau *et al.*, 2002).

The Task-to-UML tool (T2U), which can be seen as the inverse of the U2T tool discussed above, converts task models into UML system models (Lu *et al.*, 2002). The system models that are produced, however, are necessarily limited by the user-oriented nature of the task models it takes as input. T2U can, nevertheless, extract some basic system-oriented information on which a system designer can base a full UML model. This can provide some flexibility in the SDLC in that developers can build models using whatever tool or notation they prefer and then port the relevant information one way or the other.

Conclusion

This paper has observed that task models are useful in the development of interactive systems, but also difficult to build and to maintain. This difficulty is due to the fact that in order to support a variety of task applications and analyses, task models should include representations of various levels of information, from the highest level user goals down to the lowest level GUI events, and they should be represented in a single, coherent representation scheme. To help address this difficulty, the paper advocated the use of an extensible, heterogeneous task modelling environment that integrates tools that extract task information from a variety of existing sources along with tools that support the creation, manipulation and maintenance of task models, all within a coherent representation scheme. The paper reviewed the range of existing environments and tools, and presented the ISOLDE environment as an example.

Acknowledgements

The authors thank the past and present members of the ISOLDE team, including Sandrine Balbo, Todd Bentley, Nathalie Colineau, Thomas Lo, Nadine Ozkan, Maryline Specht, Robert Tot, and Jean-Claude Tarby. This work has been supported by the Office of Naval Research (ONR) grant N00014-96-0465, CSIRO and Calvin College.

References

- Baumeister, L.K., John, B.E., Byrne, M.D. (2000) *A comparison of tools for building GOMS models*, In Proceedings of the ACM Conference on Human Factors in Computer Systems (CHI-00) The Hague, The Netherlands, ACM Press, New York, 502:509.
- Balbo, S. and Lindley, C. (1997) *Adaptation of a Task Analysis Methodology to the Design of a Decision Support System*. In the Proceedings of Interact'97, Sydney, Australia. 355:361.
- Balbo, S., Ozkan, N. and Paris, C. (2002) *Choosing the right task modelling notation: a taxonomy*, Chapter 4, this volume.
- Beard, D., Smith, D. and Danelsbeck, K. (1996), *QGOMS: A direct-manipulation tool for simple GOMS models*. In the Proceedings of CHI-96 companion on Human factors in computing systems: common ground. April 13 - 18, 1996, Vancouver Canada. 25:26.
- Bentley, T. and Johnston, L., (2001) *Analysis of Task Model Requirements*, In the Proceedings of the Conference for the Computer-Human Interaction Special Interest Group of the Ergonomics Society of Australia (OzCHI), Fremantle, Australia, November 20-23.
- Biere, M., Bomsdorf, B. and Szwillus, G. (1999) *Specification and Simulation of Task Models with VTMB*, In Proceedings of the ACM Conference on Human Factors in Computer Systems (CHI-99 Demonstration) Pittsburg, May, ACM Press, New York, 502:509.
- Bomsdorf, B. and Szwillus, G. (1996). *Early Prototyping Based on Executable Task Models*. In Proceedings of the ACM Conference on Human Factors in Computer Systems (CHI-96) Vancouver, April 14-18 ACM Press, New York, v. 2, 254:255.
- Brasser, M. and Vander Linden, K. (2002). *Automatically Elicitation Task Models from Written Task Descriptions*. To appear in Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI'2002), Université de Valenciennes, France, 2002.
- Card, S. K., Moran, T. P. and Newell, A. (1983) *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates.
- Carroll, J. (1990). *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*. MIT Press, Cambridge, Massachusetts, 1990.
- Colineau, N., Paris, C. and Vander Linden, K. (2002). *An Evaluation of Procedural Instructional Text*. To appear in the Proceedings of the International Conference on Natural Language Generation, New York, USA, July 1-3.
- Constantine, L. and Lockwood, L. (1999) *Software for use: a practical guide to the Models and Methods of User-Centered Design*. Addison-Wesley - ACM press.
- Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*, MIT Press.
- Gamboa Rodríguez, F. and Scapin, D. (1997) *Editing MAD* task descriptions for modifying user interfaces at both semantic and presentation levels*, in Proceedings of the 4th International Eurographics Workshop on Design Specification and Verification of Interactive Systems (SDV-IS97), Granada, Spain. M. Harrison and J. Torres (eds), Springer-Verlag, pp 193:208.

- Harel, D. (1987) *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming (8), Elsevier, North Holland, 231:274.
- Hilbert, D. and Redmiles, D. (2000) *Extracting Usability Information from User Interface Events*, Computing Surveys, 32(4), Dec., 384:421.
- Hix, D. and Hartson, R. (1994) *Ideal: An environment to support usability engineering*, in Proceedings of the 1994 East-West International Conference on HCI, St. Petersburg, Russia.
- Hoppe, H.U. (1988) *Task-Oriented Parsing - A Diagnostic Method to be Used by Adaptive Systems*. In Proceedings of the ACM Conference on Human Factors in Computer Systems (CHI-88), Washington D.C. May 15-19, ACM Press, New York, 241:247.
- Hudson, S., John, B., Knudsen, K., Byrne, M. (1999) *A Tool for Creating Predictive Performance Models from User Interface Demonstrations*. In Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST-99), November 7-10, Asheville, USA, 93:102
- Jacobson, I., Christerson, W., Jonsson, P. and Övergaard, G. (1992) *Object-Oriented Software engineering, a use case driven approach*. Addison-Wesley.
- Jurafsky, D. and Martin, J. (2000). *Speech and Language Processing*, Prentice Hall.
- Kieras, D.E. , Wood, S.D., Abotel, K. and Hornof, A. (1995) *GLEAN*, In the Proceedings of the 8th ACM symposium on User interface and software technology, November 15-17, Pittsburgh, Pennsylvania, 91:100.
- Lewis, C. and Rieman, J. (1994) *Task-centered user interface design: A practical introduction*. Shareware Book available at <ftp://ftp.cs.colorado.edu/pub/cs/distribs/clewis/HCI-Design-Book/>.
- Limbourg, Q., Pribeanu, C. and Vanderdonckt, J. (2001) *Towards Uniformed Task Models in a Model-Based Approach*, In Interactive Systems: Design, Specification, and Verification, Proceedings of the 8th International Workshop (DSV-IS 2001), Johnson, C. (Ed.), Glasgow, June 13-15, 2001. Springer-Verlag, Berlin, 164:182.
- Lu, S., Paris, C. and Vander Linden, K. (1999) *Towards the Automatic Generation of Task Models from Object-Oriented diagrams*, In Engineering for Human-Computer Interaction, Chatty, S. and Dewan, P. (eds), Kluwer, Boston, 1999, 169:190.
- Lu, S., Paris, C., Vander Linden, K. and Colineau, N., (2002) *Generating UML Diagrams from Task Models*, submitted for publication.
- Navarre, D., Palanque, P., Bastide, R. and Sy, O. (2001) *A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications*. 12th IEEE International Workshop on Rapid System Prototyping , Monterey (USA). IEEE, June 25-27, 136:141.
- Ozkan, N., Paris, C. and Balbo, S. (1998) *Understanding a Task Model: An Experiment*. In People and Computers XII, Proceedings of Human-Computer Interaction 1998 (HCI'98), Johnson, H., Nigay, K. and Roast, C. (Eds), Springer, 23:138.

- Paris, C., Ozkan, N. and Bonifacio, F. (1998). *Novel Help for On-Line Help*. In Proceedings of ACM SIGDOC'98 (The Sixteen Annual International Conference on Computer Documentation). Quebec City, Canada. 70:79.
- Paris, C., Tarby, J. and Vander Linden, K. (2001). *A Flexible Environment for Building Task Models*. In *Proceedings of the ICM-HCI 2001*, Lille, France, 313:330.
- Paternò, F. and Ballardini, G. (1999) *Model-aided Remote Usability Evaluation*, in Proceedings of the Seventh IFIP Conference on Human-Computer Interaction (INTERACT-99), Susse, A. and Johnson, C. (eds), Edinburgh, August 30- September 3, 1999, IOS Press, 431:442.
- Paternò, F. and Mancini, C. (1999) *Developing Task Models from Informal Scenarios*, In Altona, M. W. and Williams, M. G. (Eds), Companion Proceedings of the Conference on Human Factors in Computing (CHI'99) Late Breaking Results, Pittsburgh, PA, 1999, 228:229.
- Paternò, F., Mancini, C. and Meniconi, S. (1997) *ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models*. Interact 97. Chapman & Hall. Sydney, Australia, 362:369.
- Paternò, F., Mori, G. and Galimberti, R. (2001) *CTTE: An Environment for Analysis and Development of Task Models of Cooperative Applications*, In ACM Proceedings of (SIGCHI'2001), March 31-April 5, Seattle, WA. (Extended Abstracts). 21:22.
- Puerta, A.R. (1996) *The Mecano Project: Enabling User-Task Automation During Interface Development*. AAAI96: Spring Symposium on Acquisition, learning and Demonstration: Automating Tasks for Users. Stanford, 117:121.
- Puerta, A.R. (1997) *A Model-Based Interface Development Environment*. IEEE Software, 14(4), July/August, 41:47.
- Rumbaugh, J., Jacobson, I. and Booch, G. (1999) *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading, MA.
- Sun Microsystems (2002) *Java Event Listener and Java Monkey*, <http://java.sun.com>.
- Tam, R. C., Maulsby, D. and Puerta, A. R. (1998). *U-TEL: A Tool for Eliciting User Task Models from Domain Experts*. In the Proceedings of IUI 98: International Conference on Intelligent User Interfaces. 77:80.
- Tarby, J-C and Barthet, M-F. (1996). *The Diane+ method*. In *Computer-Aided Design of User Interfaces*, in Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces (CADUI'96). Namur, Belgium, 5-7 June, 1996. J. Vanderdonck (Ed.), Presses Universitaires de Namur, Namur, 1996, 95:119.
- Vander Linden, K., Paris, C. and Lu, S. (2000) *Where Do Instructions Come From? Knowledge Acquisition and Specification for Instructional Text*. In IMPACTS in Natural Language Generation: NLG Between Technology and Applications, Schloss Dagstuhl, Germany, July 26-28. Becker, T. & Busemann, S. (Eds). DFKI report D-00-01, 1:10.
- van der Veer, G. C., Lenting, B. F. and Bergevoet, B. A. J. (1996). *GTA: Groupware Task Analysis - Modelling Complexity*. Acta Psychologica, 91, 297:322.

- van Welie, M., van der Veer, G. C. and Eliens, A. (1998). *EUTERPE - Tool support for analyzing co-operative environments*, In the Proceedings of the Ninth European Conference on Cognitive Ergonomics, August 24-26, Limerick, Ireland, 25:30.
- Wandmacher, J. (1997) *Ein werkzeug für GOMS-analysen für simulation und bewertung von prototypen beim entwurf*, Tagungsband PB97: Prototypen für Benutzungsschnittstellen 19, 35:42.
- Williams, K. (2000) *An Automated Aid for Modeling Human-Computer Interaction*. In Cognitive Task Analysis, Jan Maarten Schraagen, Susan Chipman and Valerie Shalin (Editors), Laurence Erlbaum Associates, New Jersey, 2000. 165:180.