

# Automated Knowledge Acquisition for Instructional Text Generation

Cécile Paris  
CSIRO/CMIS  
Locked Bag 17  
North Ryde, NSW 1670, Australia  
61-2-9325-3160  
cecile.paris@csiro.au

Keith Vander Linden  
Calvin College  
Department of Computer Science  
Grand Rapids, MI 49546, USA  
1-616-957-7111  
kvlinden@calvin.edu

Shijian Lu  
CSIRO/CMIS  
Locked Bag 17  
North Ryde, NSW 1670, Australia  
61-2-9325-3152  
shijian.lu@csiro.au

## ABSTRACT

Language generation systems have often been advocated for use in the generation of user documentation, but in practice, it is hard to use them because of the amount of knowledge they require as input. Without a readily available source of input, the knowledge acquisition bottleneck will prevent generation technology from ever being used. Unfortunately, knowledge acquisition for any domain-varying language generation application is difficult because there is seldom any single knowledge source or acquisition method that is adequate for acquiring the appropriate knowledge. This paper argues that this required knowledge can, in practice, be acquired from a heterogeneous set of complementary sources. It then presents *Isolde*, a support environment for technical writers that can mine these sources using an extensible set of specially tailored acquisition tools. These tools acquire instructional knowledge on-the-fly and consolidate it into a knowledge base that can drive instructional text generation.

## Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User Interfaces – *Training, help, and documentation.*

## General Terms

Documentation, Design

## Keywords

Language generation, knowledge acquisition, instructional text

## 1. INTRODUCTION

Language generation systems have often been advocated for use in documentation generation, but in practice, it is hard to use them because of the amount of knowledge they require as input. This problem can be overcome somewhat when the domain of application is static (e.g., [9]), but when the domain varies, as it does for most realistic applications, knowledge acquisition is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*SIGDOC '02*, October 20-23, 2002, Toronto, Ontario, Canada.  
Copyright 2002 ACM 1-58113-543-2/02/0010...\$5.00.

more difficult. Unfortunately, automated generation systems cannot work until they have a well-formed input. This paper argues that the knowledge required for the generation of instructional text includes both concepts and instances, and that this knowledge can be acquired on-the-fly from a heterogeneous set of available complimentary knowledge sources. These sources include written texts, existing design and task models, and functional prototypes, all of which can be found in practice and mined using specialized acquisition tools.

Because there is no single knowledge source or interface mechanism that is sufficient for this purpose, the paper argues that we need a flexible and extensible environment comprising a variety of tools that support both the automatic acquisition of knowledge from diverse sources and its consolidation into a Knowledge Base (KB) that can drive the generation process. Such an environment would allow an instruction generation system to be ported to new domains with minimal effort.

The paper begins with a specification of the types of knowledge required for generating instructions (Section 2). It then reviews approaches to acquiring that knowledge taken thus far in Natural Language Generation (Section 3). Finally, it presents *Isolde* as a proof-of-concept support environment in which the knowledge sufficient for instruction generation can be acquired using a heterogeneous and extensible set of acquisition and editing tools (Section 4).

## 2. KNOWLEDGE RESOURCES FOR INSTRUCTIONAL TEXT

Instructional text generation requires, as input, the following types of knowledge:

- **Semantic knowledge** - This knowledge is represented in two forms: *terminology* and *assertions*. Terminology (i.e., T-box *concepts/relationships*, see [17]) represents reusable types such as the action of “pressing” or the concept of a “button”. Assertions (i.e., A-box *instances*, see [17]) represent individual tokens such as “Mary’s pressing a particular button” or “the red button on the console”. These two forms are used to represent the following types of knowledge:
  - **Task knowledge**: the procedures that must be carried out by the end-user. It represents the hierarchical and temporal structure of the user’s task [10]. Task concepts (e.g., user actions, feedback, preconditions) and relationships (e.g., purpose, procedural sequence) are invariant across instructional domains, but task instances will vary from procedure to procedure.

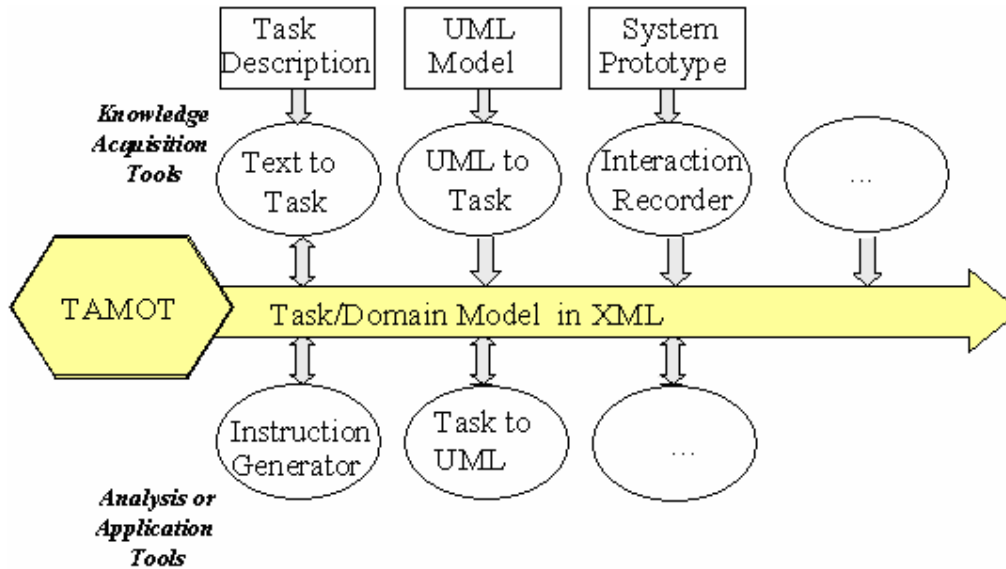


Figure 1. The ISOLDE Environment.

- **Domain knowledge:** the actions and objects used in the domain. All instructional domains share the semantic distinctions represented by general knowledge ontologies like the *Upper Model* [2] (e.g., nouns vs. verbs), but each domain requires its own set of concepts (e.g., "dial" and "number" from the mobile phone domain) and its own set of action and object instances.
- **Linguistic resources:** This knowledge comprises the *text and sentence plan operators* used by the automated planning engine, and the *realization grammar and lexicon* used by the tactical text generator [28]. The plan operators and the grammar are relatively invariant across instructional domains, but the lexical entries are associated with domain concepts, and will thus vary.

When moving to a new domain, therefore, one must acquire domain and task instances, domain concepts, and parts of the open-class lexicon.

### 3. RELATED WORK

Instructional text has been a relatively tractable target for generation systems – e.g., [19, 18, 32, 42, 14, 22, 30, 26, 36, 11]. In most of these systems, particularly the earlier ones, researchers were not concerned with the issue of acquiring the input to the generator, and, thus, they hand-coded the knowledge. The researchers' interests determined the level of detail of the various parts of the representation, and their amount of expertise in the domain determined whether or not they conducted interviews with domain experts (e.g., [31]).

Recognising the need to ease the construction of the KB, recent systems have developed various means to guide the user in entering some of the knowledge required for generation. Typically, however, these systems only allow the user to add task and domain instances, requiring concepts and lexical items to be

pre-coded. For example, Drafter and Agile used a controlled natural language interface for the domain instances and a graphical interface for the task instances [23, 11]. WYSIWYM uses a controlled language interface for both task and domain instances [26]. IDAS provided an action authoring tool that constructed instances from written texts [30].<sup>1</sup> Stede and Hensen's system [36] provides a graphical demonstration interface that records task instances as a user performs them. In all these systems, as well as in other generation applications (e.g., [34]), the concepts for the domain knowledge and the domain-specific lexicon are still hard-coded. This approach does not scale well in domain-varying applications like instructional text.

One approach to acquiring concepts is to use generic graphical KB editors (e.g., [21, 35]). We found, however, that while graphical formalisms are useful for specifying task knowledge [1, 20], they are overly cumbersome for acquiring domain knowledge. Another approach, exemplified by ModelExplainer [15], acquires lexical items for new domains by extracting text from object-oriented class diagrams. Although this is adequate for that application, the resulting KB is too "shallow" to support the generation of instructions. Instructions require an "intermediate" level of detail in the KB, which, although not as deep as the manually coded models, is sufficient and is cheaper to build [29].

### 4. THE ISOLDE ENVIRONMENT

For an instructional text generation system to be portable across domains, it must support the acquisition of task instances, domain concepts and instances, and lexical items. While there is no unique, practical source for this knowledge, there are a number of commonly available sources that can provide portions of that knowledge. These sources include written task descriptions,

<sup>1</sup> This tool may also have been able to acquire new lexical items, but this is not clear from the literature.

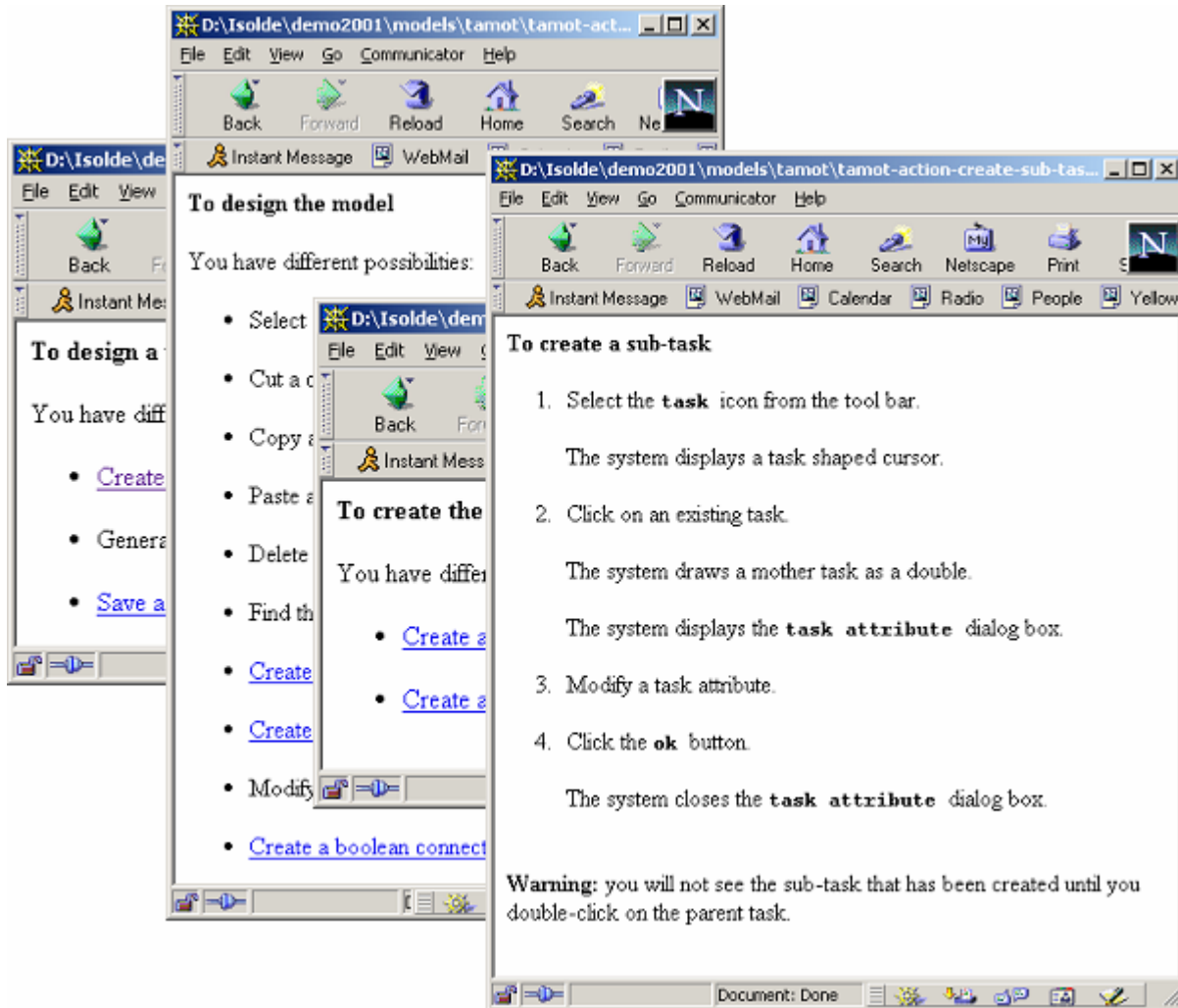


Figure 2: The Text Output by ITG

software models built using the Unified Modeling Language (UML), task models defined during user requirements analysis, and interaction event logs captured during sessions with system prototypes. These sources can be mined using specialized tools, and the resulting knowledge integrated and refined using a knowledge editor. We therefore argue that a generation system should be part of a larger environment that contains a heterogeneous and extensible set of acquisition and editing tools.

As a proof of this concept, we have developed Isolde, a knowledge acquisition and generation environment that supports the acquisition, integration, manipulation and use of knowledge within a common modeling environment. Isolde's architecture, shown in Figure 1, is centred around the Tamot task and domain model editor and its underlying XML-based representation language. The representation language is based on task models and is capable of representing the KB required to drive the instruction generation tool, shown on the bottom of the figure. Tools that support other applications of task models are also supported [24].

This paper now presents the various modules of this environment, starting with the text generator (Section 4.1), the task and domain model editor (Section 4.2), and, finally, three acquisition tools specific to the three knowledge sources mentioned above (Sections 4.3-4.5). The tools are presented in the context of a running example, one showing the use of Tamot itself.

#### 4.1 The Instructional Text Generator: ITG

The text generated by the Instructional Text Generator (ITG) is heavily based on the procedure being explained and does not contain extensive elaborations or explanations. Figure 2 shows a generated text that explains how to create a sub-task using the Tamot task model editor. The procedural relationships expressed are based on the knowledge stored in the task model (e.g., the "create a sub-task" goal and its four-step procedure with system feedback), and the action and object expressions are based on the domain knowledge associated with each task (e.g., the "create" action and the "sub-task" object). ITG supports the integration of canned text, thus allowing the inclusion of information not easily

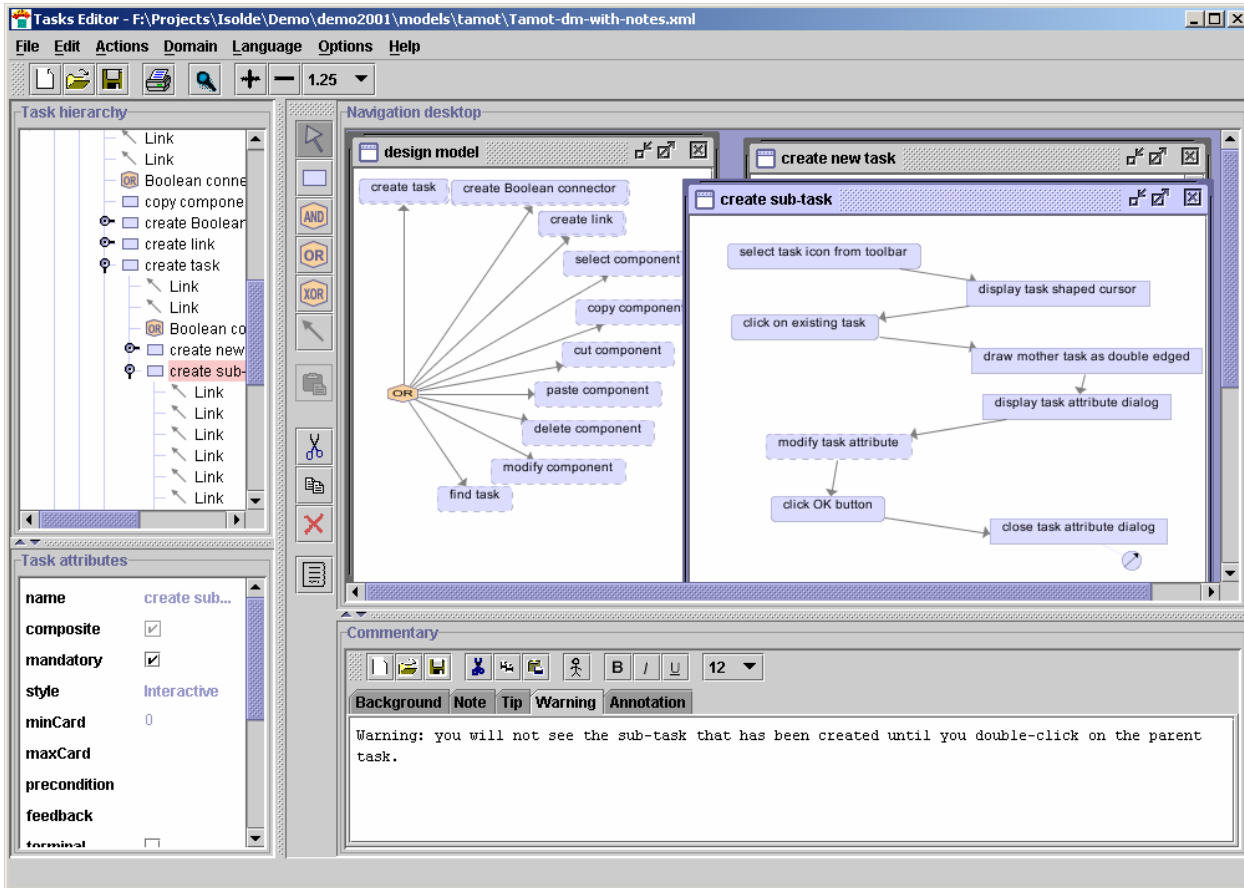


Figure 3: The Tamot Task Model Editor

acquired by Isolde's tools or represented using Isolde's task or domain models. Examples include the canned "task shaped" premodifier for the cursor in step 1 and the warning at the end of the text. Though rather terse, this form of instructions makes up the core of a system's help text and conforms to the minimalist approach to written instructions, which has been proven effective in the Human-Computer Interaction (HCI) literature [5, 6].<sup>2</sup>

The basic hierarchical structure of the hypertext shown in Figure 2, as well as many of its domain model elements and associated lexical items, were automatically extracted from a UML model by U2T, the UML-to-Task extraction tool (see Section 4.4). This extraction was done largely on-the-fly; the system did not, for example, know anything about "create" actions or "sub-task" objects before seeing the model. The resulting KB was edited using the task and domain model editors (Section 4.2). Much of this information could also be extracted from written task descriptions (Section 4.3) or from the system's prototype (Section 4.5).

## 4.2 The Task Model and Domain Model Editors: Tamot

Tamot is a general-purpose task and domain model editor. Its task representation is based on the Diane+ formalism [39], which supports task annotations (e.g., repetition and optionality) and procedural relationships (e.g., goals and sequences). Tamot is similar to other graphically based task editors (e.g., [1, 40, 26]), but its role in Isolde is, primarily, to allow users to consolidate, modify and extend the task knowledge produced by the KA tools, although it also enables them to enter the knowledge manually. Its only requirement is that each acquisition tool support Isolde's open, standardised XML-based representation. Tamot has been used as a task model editor by task analysts, and it has also been shown to be usable by technical writers [20].

Figure 3 shows Tamot editing the task model used to generate the text in Figure 1. On the upper left, Tamot displays a hierarchy of user tasks, including "create sub-task". 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 sequence of steps required to achieve the goal of creating a new sub-task: the user must select the task icon, click on an existing task, optionally modify the task attributes, and click the OK button. The systems responses to these actions are shown as rectangles.

<sup>2</sup> ITG's text has also been evaluated favourably in the language generation literature [7].

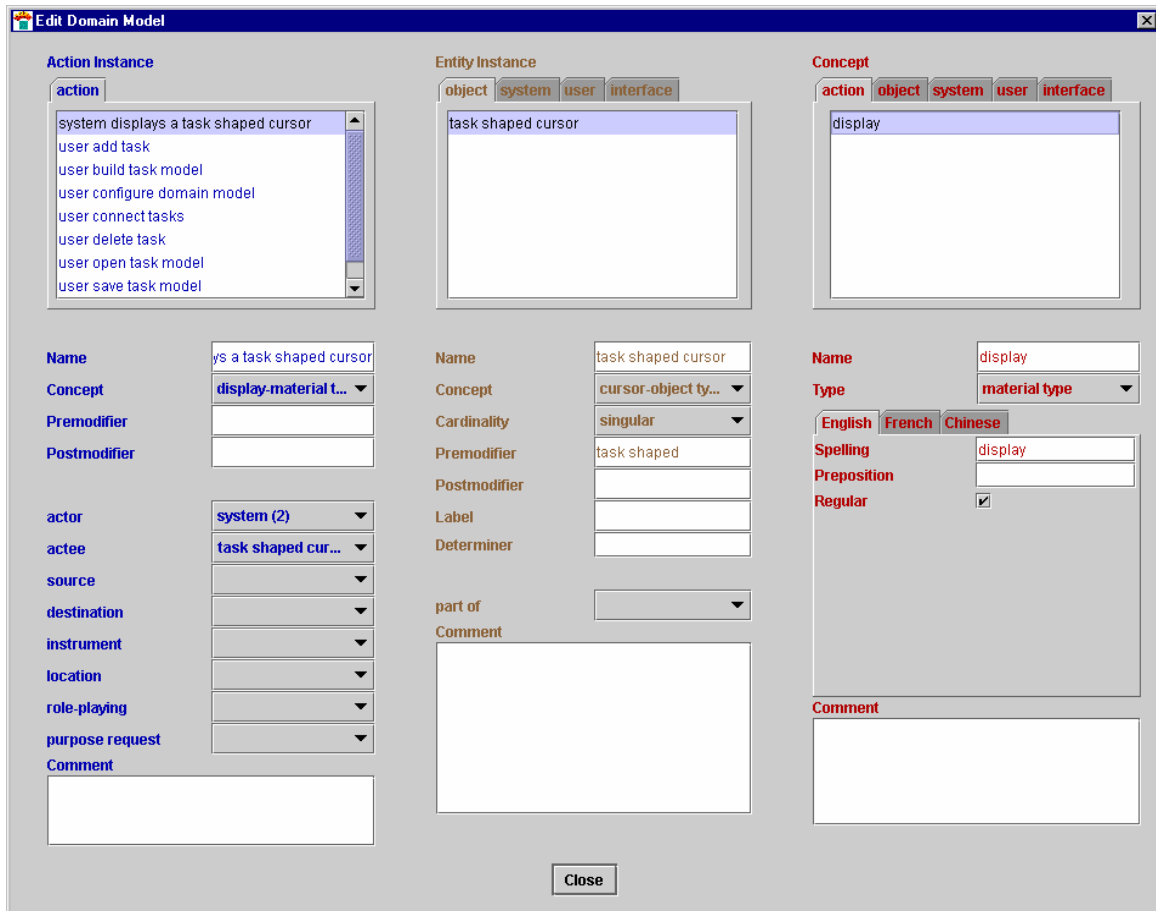


Figure 4: The Domain Model Editor

In addition to task modeling, Tamot also supports the representation and manipulation of the domain knowledge associated with the tasks. This knowledge is not necessarily required for task analysis, but it is required for text generation. The domain model editor is shown in Figure 4. It allows the user to build or modify the domain knowledge associated with each task. The figure shows a single editing window with three columns for editing action instances, entity instances, and concepts respectively. On the top left, the editor shows a list of all the action instances in the domain model. The "system displays a task shaped cursor" action is selected, and the details, shown below the list box, include its action concept ("display-material-type"), its actor ("system"), and its actee ("task shaped cursor"). The editor is now focussing on entities and concepts

referred to by this action, ignoring all the others in the model. It displays the "task shaped cursor" in the middle and the "display" action concept in the right column. The cursor is of type "cursor-object type" and has a pre-modifying string of "task shaped". Labels for interface objects (e.g., the OK button) are handled explicitly because they occur frequently in instructions. Finally, a technical writer can set a specific determiner for a specific instance, which will override ITG's default choice.

Note that many of these domain instances and concepts were derived on-the-fly by the tools discussed in the next three sections; they were not hard-coded in the system. At acquisition time, Isolde subordinates new concepts to the appropriate classes in the Upper Model [2], and creates the lexical items. The user can then correct errors with the domain model editor.

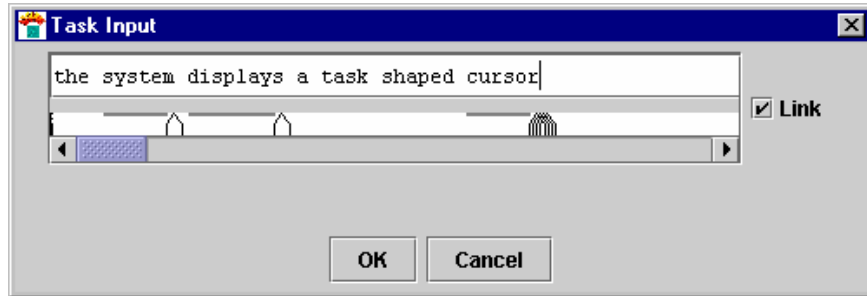


Figure 5: The Interactive T2T Interface

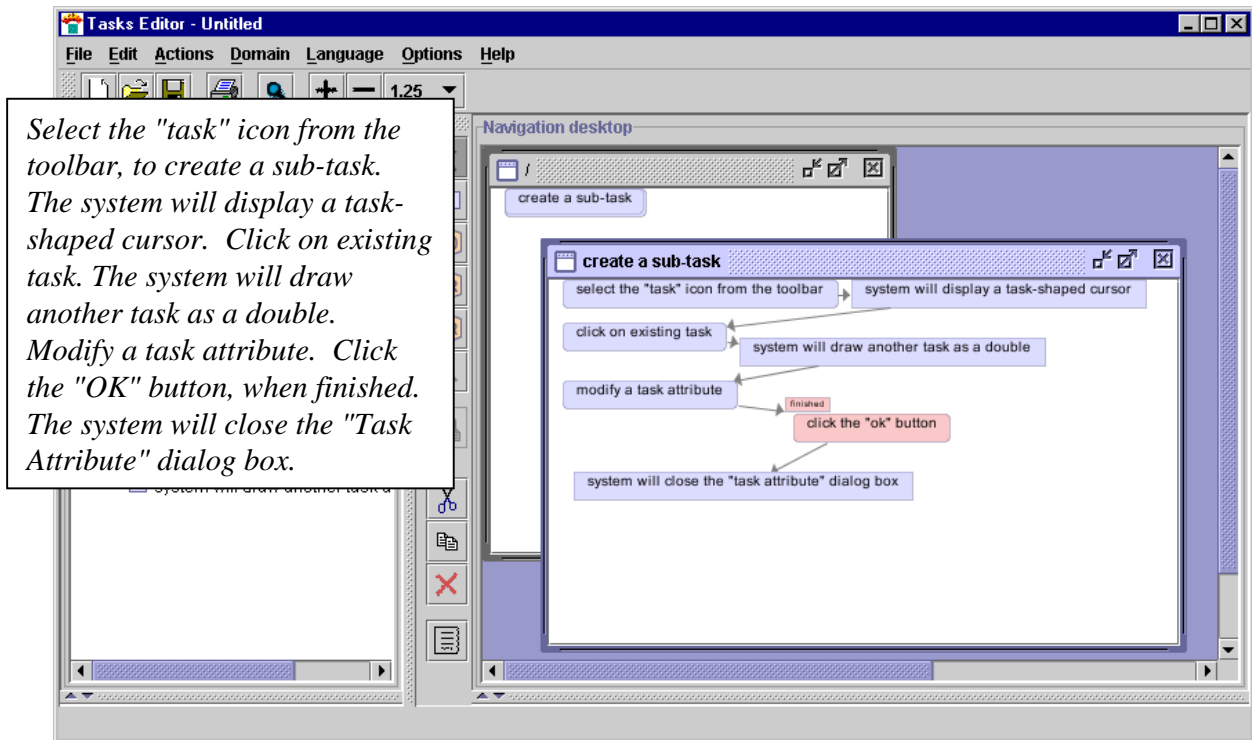


Figure 6: A Task Model Extracted by T2T

### 4.3 The Text to Task Extraction Tool: T2T

One commonly available source of task and domain knowledge is written texts, such as the textual names given to tasks created by task analysts using Tamot or more lengthy written task descriptions. The HCI community has produced tools that allow users to manually acquire knowledge from such descriptions [37, 26]. T2T, our Text-to-Task extraction tool, uses information extraction technology to automatically acquire task, domain and lexical knowledge from these sources [4].

The core of T2T is a finite-state grammar built as a 30-state Augmented Transition Network [13]. This grammar identifies the basic domain and lexical knowledge, including the actor, process and actee, as well as other constituents (e.g., instruments and locations). The rest of the sentence is left as canned strings, as discussed in Section 4.1 (cf. [30]). T2T distinguishes between nouns and verbs based on classifications of WordNet [8]. Although it can work automatically, it also allows the user to modify any incorrectly assigned constituent boundaries or head-word assignments using an interactive parsing tool, shown in Figure 5.

**Table 1: Evaluation of T2T**

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

When there are expressions of multiple tasks, the grammar extracts the procedural relationships between them, and then creates the appropriate task concepts and instances, domain concepts and instances, and lexical items. Figure 6 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 descriptions. The results of this preliminary evaluation are shown, in terms of the common metrics *precision* and *recall* [13], in Table 1.

#### 4.4 The UML to Task Extraction Tool: U2T

Design models represented in the Unified Modeling Language (UML) are common in Software Engineering [33], and can serve as another source of 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 [16, 41].

The key sources of instructional knowledge in UML are use-case diagrams, class diagrams and scenario diagrams. Examples of these are shown in Figure 7. Use-case diagrams, shown on the upper left of the figure, identify the users and their goals with respect to an application. Class diagrams, shown on the upper right, identify the basic classes of domain objects and the actions that can be performed on them. Scenario diagrams, shown on the bottom, specify the sequences of user and system events required to achieve selected use-cases.

Although UML models are system-oriented, they can contain user-oriented knowledge. U2T uses heuristics to filter out system-oriented information and export the useful task, domain and lexical knowledge to Isolde.

#### 4.5 The 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 [12], 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 [38], to extract task and domain knowledge [24].

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 modeling. For example, rather than recording an event like "the user types a filename", the event listener records a separate key-press event for every letter that 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.

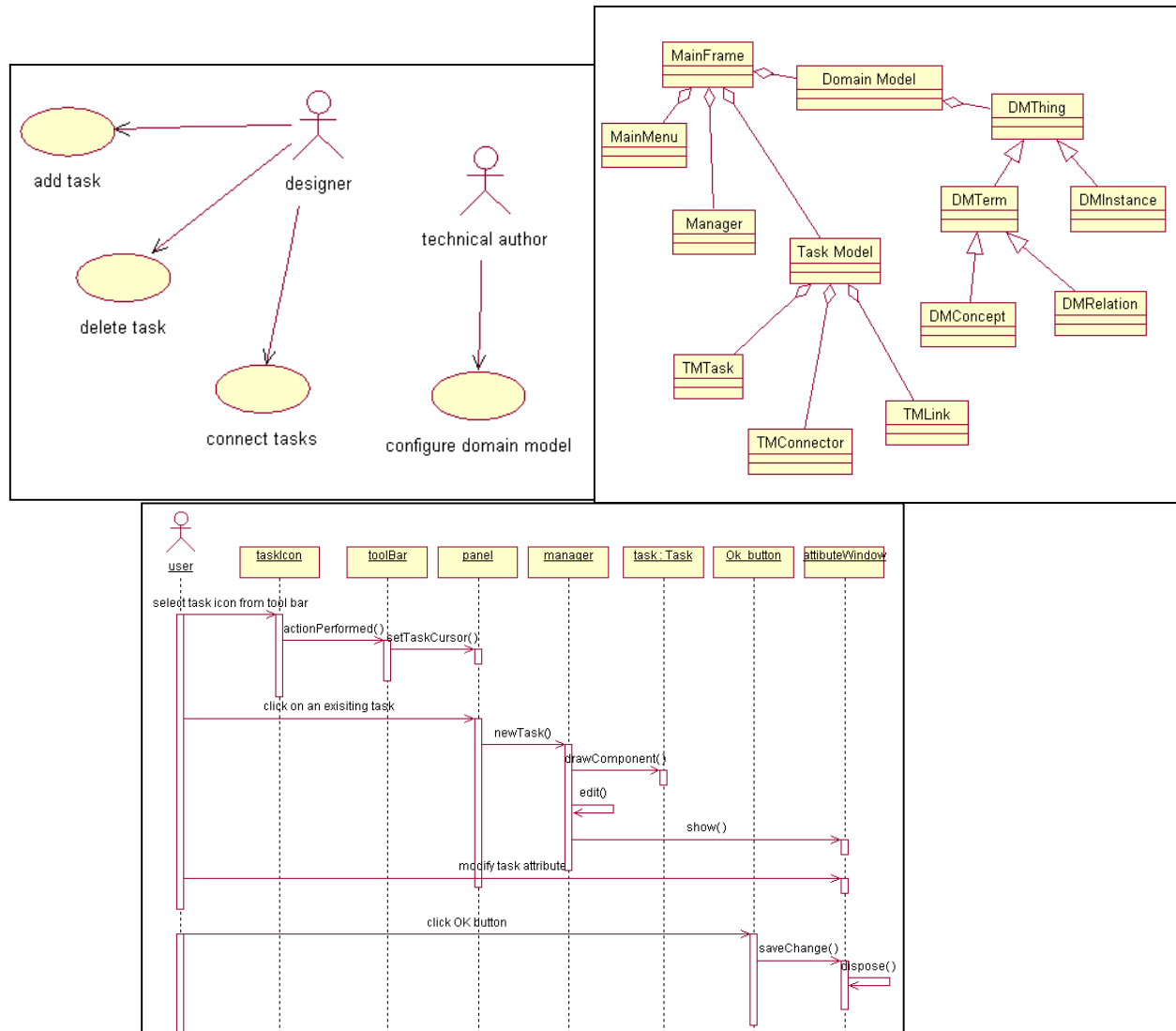


Figure 7: UML Models with User-Oriented Knowledge

## 5. CONCLUSION

This paper has discussed knowledge acquisition, one of key issues preventing the automation of parts of the technical documentation process. It presented an environment in which knowledge can be acquired from a variety of knowledge sources and then integrated into a KB capable of driving the generation of instructional text. The knowledge sources discussed were written texts, existing design and task models, and functional prototypes, all of which can be found in practice. The knowledge that can be extracted from these sources includes concepts, instances and lexical items, allowing the system to be ported to new domains with minimal effort. The environment also includes a general-purpose task and domain model editor that can be used to enter the information manually, or to modify and extend knowledge extracted from other sources.

The goal of this work is not to replace the technical writer with an automated tool. It is, rather, to provide technical writers with a support tool that automates some of the more routine aspects of their work. This is similar to the way in which Computer-Aided translation tools support translators, or even to the way in which development tools like Rational Rose and Visual Studio support software engineers by automatically generating parts of the system code.

## 6. ACKNOWLEDGEMENTS

The authors wish to thank the Isolde team, including Michael Brasser, Nathalie Colineau, Sandrine Balbo, Thomas Lo, Nadine Ozkan, and Jean-Claude Tarby. We also acknowledge the support of ONR (grant #N00014-99-0906), CSIRO and Calvin College.

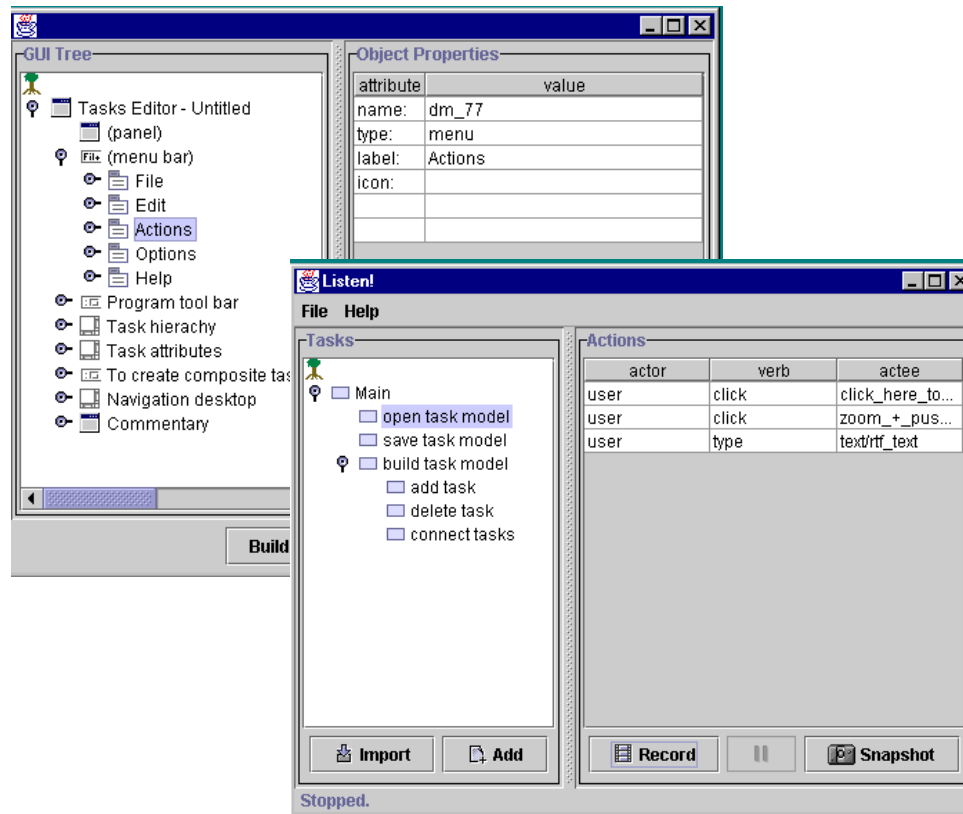


Figure 8: The User Interaction Recorder

## 7. REFERENCES

- [1] Balbo, S. & Lindley, C. (1997). Adaptation of a task analysis methodology. In Proceedings of the human-computer interaction congress : Interact '97. Sydney: Australia, Chapman and Hall.
- [2] Bateman, J. A., Kasper, R. T., Moore, J. D. & Whitney, R. A. (1990), A general organization of knowledge for natural language processing: the PENMAN upper model, Technical report, USC/Information Sciences Institute, Marina del Rey, California.
- [3] Beard, D., Smith, D. & 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.
- [4] Brassier, M. & Vander Linden, K. (2002). Automatically Elicitation Task Models from Written Task Descriptions. In Proceedings of the 4th International Conference on Computer-Aided Design of User Interfaces (CADUI'2002), Université de Valenciennes, France, 2002, 83-90.
- [5] Carroll, J. (1990). The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill. MIT Press, Cambridge, Massachusetts, 1990.
- [6] Carroll, J. (Ed) (1998). Minimalism Beyond the Nurnberg Funnel. MIT Press, Cambridge, Massachusetts, 1998.
- [7] Colineau, N., Paris, C. & Vander Linden, K. (2002). An Evaluation of Procedural Instructional Text. To appear in the Proceedings of the 2<sup>nd</sup> International Conference on Natural Language Generation, New York, July 1-3.
- [8] Fellbaum, C., editor (1998). WordNet: An Electronic Lexical Database, MIT Press.
- [9] Goldberg, E., Driedger, N. & Kittredge, R. (1994). Using natural language processing to produce weather forecasts. IEEE Expert, 9(2): 45-53.
- [10] Goldman, A. I. (1970). A Theory of Human Action. Englewood Cliffs, NJ: Prentice Hall.
- [11] Hartley, T., Scott, D., Kruijff-Korbayová, I., Sharoff, S., Sokolova, L., Dochev, D., Staykova, K., Čmejrek, M., Hana, J. & Teich E. (2000). Evaluation of the final prototype, deliverable Eval2-Bu of the AGILE project. Oct 2000, URL: <http://www.itri.brighton.ac.uk/projects/agile>
- [12] Hilbert, D. & Redmiles, D. (2000) Extracting Usability Information from User Interface Events, Computing Surveys, 32(4), Dec. 2000, 384 – 421.
- [13] Jurafsky, D. & Martin, J. (2000). Speech and Language Processing, Prentice Hall.
- [14] Kosseim L. & Lapalme G. (1994). Content and rhetorical status selection in instructional text. In Proceedings of the 7th Int. Workshop on NLG, Kennebunkport, ME, 53-60.
- [15] Lavoie, B.; Rambow, O., & Reiter, E. (1997). Customizable Descriptions of Object-Oriented Models. In Proceedings of

the Fifth Conference on Applied Natural Language Processing, Washington, DC, 265-268.

- [16] Lu, S., Paris, C. & Vander Linden, K. (1999): Towards the automatic generation of task models from object oriented diagrams. In Engineering for HCI, S.Chatty and P.Dewan (Eds), Kluwer, Boston, 1999, 169 – 189.
- [17] MacGregor, R. & Bates, R. (1987). The Loom knowledge representation language. In Proceedings of the Knowledge-Based Systems Workshop, St Louis, April 23-23, Technical Report ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, Ca.
- [18] McKeown, K., Elhadad, M., Fukumoto, Y., Lim, J., Lombardi, C., Rogin, J. & Smadja, F. (1990). Natural language generation in COMET, Current Research in Natural Language Generation, R. Dale, C.S. Mellish, and M. Zock (Eds). Chapter 5, Academic Press, 103 – 140.
- [19] Mellish, C. & Evans, R., (1989). Natural language generation from plans, Computational Linguistics, 15(4), 233-249.
- [20] Ozkan, N., Paris, C. & Balbo, S. (1998). Understanding a Task Model: An Experiment. In People and Computers XII, Proceedings of Human-Computer Interaction 1998 (HCI'98), H. Johnson. K. Nigay and C. Roast (Eds), Springer, 123-138.
- [21] Paley, S. (1996) Generic knowledge-Based editor user manual, Technical report, SRI International, California.
- [22] Paris, C., Vander Linden, K., Fischer, M., Hartley, A., Pemberton, L. Power, R. & Scott, D. (1995). A Support Tool for Writing Multilingual Instructions. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, 1398 – 140.
- [23] Paris, C. & Vander Linden, K. (1996). Drafter: An Interactive Support Tool for Writing Multilingual Manuals, IEEE Computer, 29 (7), July 1996, 49-56.
- [24] Paris, C., Tarby, J. & Vander Linden, K. (2001). A Flexible Environment for Building Task Models. In Proceedings of the ICM-HCI 2001, Lille, France, 313 – 330.
- [25] Paternò, F. & 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.
- [26] Paternò, F., Mori, G. & 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., 21 – 22.
- [27] Power, R., Scott, D. & Evans, R. (1998). What You See is What You Meant: direct knowledge editing with natural language feedback, In Proceedings of the 13th European Conference on Artificial Intelligence, ECAI'98, Brighton, UK, August, 677 – 681.
- [28] Reiter, E. & Dale, R. (2000). *Building Natural Language Generation Systems*, Cambridge University Press.
- [29] Reiter, E. & Mellish, C. (1993). Optimizing the Costs and Benefits of Natural Language Generation. In Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI'93), 1164-1169, Morgan Kaufmann.
- [30] Reiter, E., Mellish, C., and Levine, J. (1995) Automatic Generation of Technical Documentation, Applied Artificial Intelligence, 9(3), 259-287.
- [31] Reiter, E., Robertson, R., Lennox A. S. & Osman, L. (2001). Using a randomised controlled clinical trial to evaluate an NLG system. In Proceedings of ACL-2001, Toulouse, France, 434 - 441.
- [32] Rösner, D. & Stede, M. (1994). "Generating multilingual technical documents from a KB: The TECHDOC Project", In Proceedings of the International Conference on Computational Linguistics, COLING-94, Kyoto, 339 – 346.
- [33] Rumbaugh, J., Jacobson, I., & Booch, G. (1999). The UML Reference Manual, Addison-Wesley, Reading, MA.
- [34] Sheremetyeva, S. & Nirenburg, S. (1996). Knowledge Elicitation for Authoring Patent Claims. IEEE Computer, 29 (7), July 1996, 57 - 62.
- [35] Skuce, D. & Lethbridge, T. (1995). CODE4: A Unifies system for managing conceptual knowledge, International Journal of Human-Computer Studies, 42, 413-451.
- [36] Stede, M. & Hemsén, H. (2000). Instructing by doing: Interactive graphics- and knowledge-based generation of instructional text. In IMPACTS in Natural Language Generation, Schloss Dagstuhl, Germany, July 26-28. Becker, T. & Busemann, S. (Eds). DFKI report D-00-01, 11 - 21.
- [37] Tam, R. C., Maulsby, D. & 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.
- [38] Sun Microsystems (2002). Java Event Listener and Java Monkey, <http://java.sun.com> .
- [39] Tarby, J-C & Barthet, M-F. (1996). the Diane+ method. In Computer-Aided Design of User Interfaces, 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.
- [40] van Welie, M., van der Veer, G. C. & Eliens, A. (1998). EUTERPE - Tool support for analyzing co-operative environments, In the Proceedings of the Ninth European Conference on Cognitive Ergonomics, 25-30, August 24-26, 1998, Limerick, Ireland.
- [41] Vander Linden, K., Paris, C. & Lu, S. (2000). Where Do Instructions Come From? Knowledge Acquisition and Specification for Instructional Text. In Impacts in Natural Language Generation, Schloss Dagstuhl, Germany, July 26-28. Becker, T. & Busemann, S. (Eds). DFKI #D-00-01,1-10.
- [42] Wahlster, W, André, E, Finkler, W, Profitlich, H & Rist, T. (1993): Plan-based integration of natural language and graphics generation, Artificial Intelligence, 63(1-2), 387-428.

