

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE

Automatic Generation of Help

Anciaux, Valery; Plier, Christophe

Award date:
1998

Awarding institution:
University of Namur

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
Rue Grandgagnage, 21

B-5000 Namur (Belgium)

Automatic Generation of Help

By

Valéry ANCIAUX and Christophe PLIER

Co-Advisers: Professor François BODART and Doctor Jean VANDERDONCKT

Thesis submitted in fulfillment of the requirements for the degree of
Master of Computer Science

Academic year 1997 - 1998

Abstract:

The purpose of this thesis is to analyze the different tools available for automatic help generation and to describe the Isolde project (Integrated Software and On-Line Documentation Environment), a new tool for writing on-line help.

The objective of Isolde is to automatically generate hypertext based on-line help. This project exploits the common grounds between task model and system behavior models. A task modeling tool used to process a particular model has been implemented and is described in this thesis.

The task modeling tool is aimed at importing a particular task model into a graphical form and easy to modify structure, then exporting the model in a format usable for natural language generation. An example is provided, together with a demo on disk.

Résumé:

L'objectif de ce mémoire est d'analyser les différents outils de génération d'aide automatique ainsi que de décrire le projet Isolde (Integrated Software and On-Line Documentation Environment), un nouvel outil de génération d'aide en ligne.

Isolde a pour but de générer automatiquement de l'aide en ligne sous forme de fichiers hypertextes. Ce projet exploite les points communs entre un modèle de la tâche et les modèles de comportement de systèmes. Un éditeur graphique utilisé pour traiter un modèle particulier a été implémenté et est décrit dans ce mémoire.

L'éditeur réalisé a pour but d'importer un modèle, de le représenter sous forme graphique facilement modifiable et de l'exporter dans un format utilisable pour la génération de langage naturel. Un exemple est traité et une démo sur disquette est fournie.

Training period:

Commonwealth Scientific and Industrial Research Organization
Mathematical and Information Sciences
Locked Bag 17
North Ryde, NSW 2113
Australia

Thanks

We wish to thank Professor François BODART and Doctor Jean VANDERDONCKT for their precious advice.

We would also like to thank the staff of the CSIRO/MIS in Sydney, Australia and particularly Sandrine BALBO, Shijian LU, Nadine OZKAN and Cécile PARIS for their fabulous welcome and their warm spirit.

We gratefully acknowledge the participation of Keith VANDER LINDEN of the Calvin College, USA.

Finally, we want to give a special thanks to all the people who have helped us during our training and the preparation of our thesis.

Table of contents

1. INTRODUCTION.....	11
2. HELP SYSTEMS	13
2.1 HELP	15
2.1.1 <i>The different types of help</i>	15
2.1.1.1 The stimulus help (on the fly).....	15
2.1.1.2 The contextual help	15
2.1.1.3 The general help.....	16
2.1.1.4 Future directions.....	16
2.1.2 <i>Why do users avoid using help?</i>	17
2.2 HELP SYSTEMS	18
2.3 ISOLDE.....	18
2.3.1 <i>Type of documentation Isolde can produce</i>	19
3. ISOLDE VERSUS OTHER AUTOMATIC HELP GENERATION SOFTWARES.....	21
3.1 COMPARISON CRITERIA'S	23
3.2 CARTOONIST: COUPLING A UI FRAMEWORK WITH AUTOMATIC GENERATION OF CONTEXT- SENSITIVE ANIMATED HELP	25
3.2.1 <i>Help messages</i>	25
3.2.2 <i>Architecture</i>	25
3.2.3 <i>Improvements</i>	26
3.2.4 <i>Comparison</i>	26
3.3 CONTEXTUAL HELP FOR FREE WITH FORMAL DIALOGUE DESIGN	27
3.3.1 <i>Introduction</i>	27
3.3.2 <i>User access to contextual help</i>	28
3.3.3 <i>Formal description of the dialogue</i>	28
3.3.4 <i>Automatic help generation</i>	30
3.3.5 <i>Comparison</i>	31
3.4 HELPTALK: AUTOMATIC GENERATION OF TEXTUAL, AUDIO, AND ANIMATED HELP IN THE USER INTERFACE DESIGN ENVIRONMENT	32
3.4.1 <i>Introduction</i>	33
3.4.2 <i>What can HelpTalk generate?</i>	33
3.4.2.1 Textual Why help	33
3.4.2.2 Audio/animated HOW help.....	34
3.4.3 <i>Help Knowledge source</i>	34
3.4.4 <i>Architecture</i>	36
3.4.5 <i>Generation algorithms</i>	37
3.4.6 <i>Implementation</i>	37

3.4.7 <i>Comparison</i>	37
3.5 H3: AUTOMATIC GENERATION OF HELP FROM INTERFACE DESIGN MODELS.....	38
3.5.1 <i>Help messages</i>	38
3.5.2 <i>Architecture</i>	39
3.5.3 <i>Comparison</i>	39
3.6 COGENTHELP: A TOOL FOR AUTHORIZING DYNAMICALLY GENERATED HELP FOR JAVA GUIS.....	40
3.6.1 <i>Design goals</i>	41
3.6.2 <i>System</i>	41
3.6.3 <i>Technically</i>	42
3.6.4 <i>Concretely</i>	42
3.6.4.1 <i>The help window</i>	43
3.6.4.2 <i>Help snippets</i>	43
3.6.4.3 <i>Consistency checking</i>	44
3.6.4.4 <i>Editing the table of contents</i>	45
3.6.4.5 <i>Dynamic help topics</i>	45
3.6.5 <i>Conclusion</i>	46
3.6.6 <i>Comparison</i>	47
4. THE TASK MODEL.....	49
4.1 INTRODUCTION.....	51
4.2 A CASE TOOL : RATIONALE ROSE.....	51
4.2.1 <i>Use cases and use case diagrams</i>	52
4.2.2 <i>Use cases and Task models</i>	53
4.2.3 <i>Interaction diagrams</i>	54
4.2.4 <i>Interaction diagrams and Task models</i>	56
4.2.5 <i>State (transition) diagrams</i>	57
4.2.6 <i>State (transition) diagrams and Task models</i>	58
4.3 THE DIANE+H FORMALISM.....	58
5. DEVELOPMENT SOFTWARES.....	61
5.1 JAVA.....	63
5.1.1 <i>The origins</i>	63
5.1.2 <i>Why this name: Java</i>	63
5.1.3 <i>Advantages</i>	63
5.1.3.1 <i>Java is simple</i>	63
5.1.3.2 <i>Java is object-oriented</i>	64
5.1.3.4 <i>Java is compiled</i>	65
5.1.3.5 <i>Java is platform independent</i>	66
5.1.3.6 <i>Java is multi-threaded</i>	66
5.1.3.7 <i>Java is extensible</i>	67
5.1.3.8 <i>Java is robust</i>	67
5.1.4 <i>Disadvantages</i>	67
5.2 VISUALAGE.....	68
5.3 REMARK.....	68
6. INTERNAL MANAGEMENT.....	69
6.1 ARCHITECTURAL ANALYSIS (SCHEMA OF THE MODEL).....	71
6.2 SEMANTICS.....	71
6.3 FUNCTIONAL ANALYSIS (OO APPROACH : DESCRIPTION OF THE OBJETS AND THE HIERARCHY).....	73
6.4 THE ROOT AND ITS PROPERTIES.....	75
6.5 DESCRIPTION OF THE PROCEDURES.....	76
6.5.1 <i>Class IO</i>	76
6.5.2 <i>Class Widget</i>	77
6.5.3 <i>Class Sequence</i>	83
6.5.4 <i>Class Task</i>	85
6.5.5 <i>Class Boolean Connector</i>	88
6.6 HOW TO CONVERT FILES FROM ROSE INTO THE INTERNAL STRUCTURE.....	91
6.6.1 <i>Description of the file generated by Rationale Rose</i>	91
6.6.2 <i>The reading of the file and the transformation into internal structure</i>	91
6.6.3 <i>The coordinates problem</i>	92
6.7 MODIFICATIONS OF THE STRUCTURE BY THE TECHNICAL WRITERS.....	93

6.8 SAVING AND RE-OPENING A PARTICULAR MODEL	94
6.8.1 <i>Second type of file (with coordinates, etc.) : the user file</i>	94
6.9 HOW TO CONVERT THE INTERNAL STRUCTURE INTO A PREDEFINED (LISP) FORMAT	95
6.9.1. <i>Third type of file: the LISP (or export) file (with no coordinates, etc.)</i>	95
7. THE GRAPHICAL USER INTERFACE.....	97
7.1 INTRODUCTION	99
7.2 THE FUNCTIONS OF THE INTERFACE.....	99
7.2.1 <i>Buttons and checkboxes of the GUI</i>	99
7.2.2 <i>The menu item "File"</i>	101
7.2.3 <i>The menu item "Edit"</i>	102
7.2.4 <i>The menu item "View"</i>	102
7.2.5 <i>The expansion box</i>	103
7.2.6 <i>Various</i>	104
7.3 SUGGESTIONS FOR THE IMPROVING OF THE INTERFACE.....	107
8. THE NATURAL LANGUAGE GENERATION.....	109
8.1 INTRODUCTION	111
8.2 THE TEXT GENERATOR.....	111
8.2.1 <i>Illustration</i>	111
8.3 A REMARK CONCERNING THE HYPERTEXT SEMANTICS	114
9. AN EXAMPLE.....	115
9.1 UML MODEL	117
9.2 IMPORT	117
9.3 DISPLAY	118
9.4 EXPORT	118
9.5 LANGUAGE GENERATION	119
10. CONCLUSION.....	123
BIBLIOGRAPHY	125
TABLE OF FIGURES.....	131
APPENDIX A.....	133
APPENDIX B.....	155

CHAPTER 1

Introduction

This thesis is based on three main parts. First of all, we situate the field of help systems and the different representations of knowledge (chapters 2 to 4). Secondly (chapters 5 to 8), we describe the different tools we have used and the generation process of the Isolde project (Integrated Software and On-Line Documentation Environment). Finally, we illustrate the Isolde project by an example.

In chapter 2, we focus on the necessity of having help systems in computer applications and introduce the Isolde project: a new tool for writing on-line help, supported by a grant to CSIRO (Commonwealth Scientific and Industrial Research Organization) from the Office of Naval Research (ONR).

Chapter 3 compares Isolde with other softwares that generate help automatically.

Chapter 4 describes the formalisms considered in the Isolde project and the choice of task model for knowledge representation.

In chapter 5, we situate the environment we have worked in: the JDK (Java Development toolKit) compiler and VisualAge, the java builder from IBM. We also make some comments about these two softwares.

Chapter 6 describes the task model editor and shows the different steps required to process a particular model. A simple automatically generated model has to be converted into an editable, easy to read, graphical form.

The next step (chapter 7) is about displaying a model graphically, using the formalism defined in chapter 4. We also identify the different commands available, and mention some hints to improve the current interface.

Chapter 8 covers the last part of the process: the generation of on-line help in an hypertext form.

In chapter 9, we illustrate the Isolde project with an example, from an UML model to a task model and finally hypertext files.

During our training period at CSIRO in Sydney, we have realized the TAMOT (TAsk MOdeling Tool) for the Isolde project.

Chapters 2, 3, 5 and 7 have been written by Christophe Plier. Chapters 4, 6 and 8 have been written by Valéry Anciaux. The other chapters have been written by both of us.

CHAPTER 2

Help systems

Resume:

This chapter introduces the notion of help and help systems.

We will compare the existing help generation systems from different criteria's in the next chapter (chapter 3).

The section 2.1 covers the general notion of help.

In section 2.2, we explain the utility of help systems in computer softwares.

In section 2.3, we introduce Isolde, the automatic generation of help project in which we have worked during our training period at the CSIRO.

2.1 Help

Interacting daily with a computer usually presents us with no complicated problems or unpleasant surprises. We use certain applications with functions we understand for a small, familiar set of tasks. But ask us to try to accomplish an unfamiliar task or ask us to use a different word processor, or even to write the document on a different kind of computer, and the experience is quite different.

We need help!

2.1.1 The different types of help

Help facilities can only alleviate the problems experienced by new or casual users by providing the desired information when and where it is needed.

There are three forms of help:

2.1.1.1 The stimulus help (on the fly)

- **Form:** the help is appearing in the information area or in a little rectangle near the pointed object.
- **Contents:** short and concise help, which gives information about the pointed objects by the mouse.
- **Accessibility:** we put the mouse pointer on the object.

2.1.1.2 The contextual help

- **Form:** message box (fast contextual help) or help window from the general help¹.
- **Contents:** information about the box and the tasks units to execute in the box (or window) from which the help has been called.
- **Accessibility:** from the command buttons "Help", or the key F1.

The contextual help is specifically concerned with the current state of human-computer interaction and should ideally answer three basic questions that the user may be interested in: "What?", "Why not?" and "How?".

The question "What?" corresponds to the interrogation "What can I do from now on?" The question "Why not?" naturally comes to the user's mind as soon as he wishes to trigger an action whose triggering widget is currently grayed out. And the question "How?" stands for "How can I make that action available again?" The answer to this

¹ Cf. infra 2.1.1.3

question should naturally complement that of the question "Why not?" by providing the sequence of commands to trigger in order to enable the desired command.

2.1.1.3 The general help

- **Form:** arborescence of secondary windows structured in hypertext.
- **Contents:** information about what the user could do and how to do it.
- **Accessibility:** it is possible to have access to the different elements of the general help by the way of the unrolled menu. This one can be unroll from the "Help" or the "?" item of the menu bar.

When the user presses the HELP key, different kinds of help are available:

1. Overview help: describe the widgets in the current window ("Where we are") along with a goal-oriented list of all the functions which may be performed in the context ("What we can do" e.g., "To make a phone call: select an entry, then use Call") ;
2. Problem help: available when a problem (system or user-induced) has occurred and a message is currently displayed on the screen. Problem help expands on the cause of the problem and offers possible solutions ;
3. Command Help: lists all of the currently valid commands (function-oriented "What we can do" e.g., "The Call commands places a phone call to the selected directory entry). More detailed command help is available for each command listed ;
4. Field Help: describes the field, which the cursor is currently on, if one exists ;
5. Help about help: describes how to use the help service itself ;
6. Index: an alphabetical list of all the topics of the task or the object ;
7. Tutorial: educative information about the tasks and the objects ;
8. About...: display a window with information about the application like copyright, logo, ...

2.1.1.4 Future directions

Now people usually have to ask for help, but more intelligent systems could anticipate user needs. This would be particularly useful for situations in which users may recognize the need for help but not know how to ask for it. Truly intelligent, active help would reinterpret the actions according to user's intentions. More advanced systems would not only tell us how to carry out a task but actually perform it for us.

Active help systems may present their own special problems and obstacles. First, there is the difficulty of designing a system that can interpret user's intentions. In the absence of this capability, the system is likely to develop wrong interpretations, leading to diagnoses and advises that may be entirely inappropriate, further confusing the user. Another pitfall is that of the computer taking too much control from the user, leaving the user feeling like a bystander. The user is put in the position of being a passive observer rather than an active participant. Computer-initiated help in the form of unsolicited advice is potentially intrusive and irritating, just as a person looking over our shoulder offering advice can be.

We can use advances in other areas to make help more accessible. Different modes of access for help should be considered. Voice activation is one obvious possibility, gesture is another.

With regard to the method of presentation of information, animation deserves further exploration. Motion can effectively add a whole new dimension to the way information is conveyed.

2.1.2 Why do users avoid using help?

Even if there is on-line help available, the unfortunate fact is that most help systems tend to exacerbate users' problems. Finding the simplest piece of information can turn into a complicated exercise. If and when we find what we need, it will probably take a long time to get back to the task we were working on, if we can even remember what it was. The process is time-consuming and effortful, and is apt to leave us feeling ineffective and frustrated.

As a consequence, some people do not even try to use the help provided. People prefer to flounder around in their work environment, or better yet turn to an in-house expert, rather than use the on-line help provided by the system.

Here are the common complaints:

- Difficulty of finding information ;
- Failure to deliver relevant information ;
- Difficulty of switching between help and the working context ;
- Complexity of the help interface ;
- Quality and layout of help information.

So, to have a useful and USED help system, the design must consider these complaints. And the following five principles which are the general philosophy for the design must be respected:

1. On-line help should never be a substitute for good interface design.

2. Help should be context-sensitive; it should not take the user away from the task at hand.
3. Help systems should assist users in framing their questions and provide different help for different questions.
4. Help systems should be dynamic and responsive.
5. Users should not need help to get help.

2.2 Help systems

Help systems today are usually developed separately from the system they support. As a result, building and maintaining help systems requires substantial effort:

- There is a wide variety of tasks ;
- Some complex tasks can invoke several dialogue boxes, thus requiring the users to invoke help on all the dialogue boxes ;
- The design of the help system replicates reasoning that went into the design of the application ;
- Complex programming is needed because the help system must access internal application data structures ;
- There is no support for changing the help system when the application is modified ;
- Making help systems have a consistent interface across different applications is difficult.

That is the reason why a lot of researchers are working in the development of assisted/computer-aided or automatic generation of help software. That is the case for the project *Isolde*. We have made our training period in Sydney (Australia) at the CSIRO, the national Australian research institute. We were members of the *Isolde* team. In the next paragraphs of this chapter, we give an overview of this project. The project will be detailed in the next chapters.

2.3 *Isolde*

The technical writing team from IBM² Global Services has established a close relationship with researchers in software engineering, HCI³ and artificial intelligence

² International Business Machines

³ Human Computer Interaction

from CSIRO. The result of this collaboration is a new tool for writing on-line help: Isolde.

Isolde takes as input a high-level description of the functionality of the software to be documented (called a *task model*) and automatically generates hypertext for a subset of its on-line documentation. Hence Isolde envisages that instead of producing text directly, technical writers will produce a description of the system functionalities (in a task modeling notation), and, once the text is generated automatically, will revise and correct it.

Isolde is designed based on previous work in natural language generation and the collaboration between the technical writers and CSIRO. Isolde can be used in two modes:

1. **Stand-alone:** to specify formally the functionality of the software, i.e., to create task models, manipulate them and generate on-line help from them ;
2. **Coupled** to a CASE⁴ tool in order to bootstrap the process and provide the technical writer with a draft of the task models for the software to be documented.

Their goal is therefore to develop a software tool, which includes the following features:

- A *specification language* formalizing not only the appearance and behavior of the application but also the domain (or user) tasks to be achieved using the application, their decomposition and dependencies. They will augment formalisms already developed for specifying an application and provide a way of modeling it from the end-user's perspective ;
- *Modeling facilities* to allow one to create and modify the model of the application under development. These facilities will include display and acquisition methods and will allow for the direct manipulation of the model. The model will become an explicit statement of what the system is for, not merely a statement of what it does ;
- A *presentation tool*, which takes the extended application model and procedures documentation, in various styles if necessary. This tool will produce draft hypertext-based on-line help, which can be edited by the documentation expert. The on-line help generated will be concise, task-oriented and context-sensitive.

2.3.1 Type of documentation Isolde can produce

Isolde focuses on generating procedural help. By procedural help, we mean the help that basically enumerates the series of steps required to perform a user goal. Procedural help can be seen as an answer to the question "How to". So, if we

⁴ Computer Aided Software Engineering

compare this help with the different types of help⁵, the procedural help is typically a contextual help answering to the question "How?" Figure 2.1 shows an example of procedural help from MSWord97.

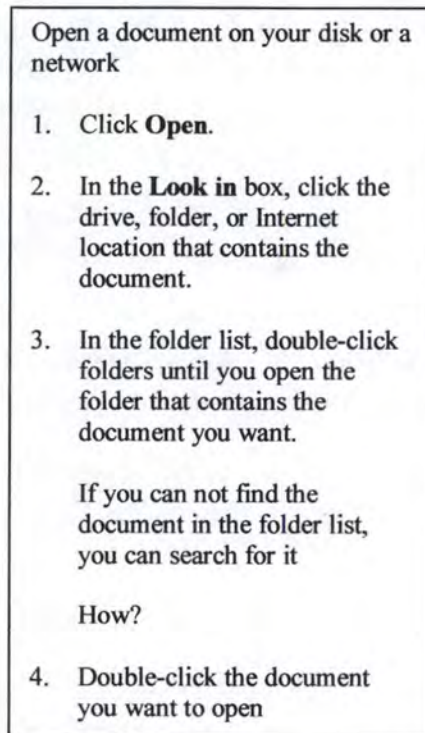


Figure 2. 1: Example of procedural help from MSWord97. Part of the screen for "How to open a file from the hard disk"

This type of help describes user actions in terms of system functions on the user interface. In this sense, procedural help, by its very nature, closely matches the functioning of the system. It thus seems feasible to automate its production.

The relevance of procedural type of help is supported by the current trend towards "minimalist instructions". The philosophy of minimalist instructions is based on the argument that learning software is more effective if software documentation is short, simple and directed towards real work activities. In the light of this philosophy, procedural help, which is task-oriented and therefore focused on the user's activities, is a central type of help.

⁵ Cf. supra 2.1.1

CHAPTER 3

Isolde versus other automatic help generation softwares

Resume:

After introducing the notion of help and presenting Isolde (chapter 2), this chapter analyzes other automatic help generation softwares and compares them with Isolde. This way, we make a state-of-the-art of the available possibilities to automatically generate help.

Chapter 4 will describe the formalism used for knowledge representation in Isolde.

To be able to compare the different softwares, we define some comparison criteria's in section 3.1.

The following softwares are described in details: Contextual help for free with formal dialogue design in section 3.3, HelpTalk in section 3.4 and CogentHelp in section 3.6. These last ones are just overviewed: Cartoonist in section 3.2 and H3 in section 3.5.

3.1 Comparison criteria's

1. The **type** of generated help. The help can be **task-oriented** and answer to questions like "What can I do from now on?" or "How can I make that action available again?" It can also be **widget-oriented** and answer to questions like "Where can I click, and what will happen?" or "What commands are available?"

The advantage of the task-oriented approach is that the types of questions we can ask are very close to human's way of thinking. The disadvantage is the difficulty of generating this type of help. Opposite to this is the widget-oriented approach with the advantage of easily generating the help. And with the disadvantage of only accepting questions related to widgets.

2. The **style** of generated help. It can be mono-media or multi-media. The media can be textual, audio or animated.
3. The help can be **hypertext**. It means that we can click on hypertext links and receive help concerning other topics.
4. The help can be **context-sensitive** or **context-insensitive**. It means that the generated help in a same situation can be different depending on the context. The contents change according to the current state of the application.
5. The **quantity** of work needed to automatically generate help. To have a precise idea of how long it takes to prepare the automatic generation of help, we are going to score this quantity of work (Figure 3.1).

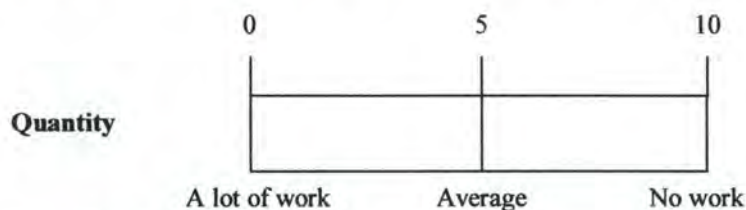


Figure 3.1: Quantity of work needed to automatically generate help

6. The **quality** of generated help (must be as closest as possible to natural language). For example: "This item is currently disabled, because the color Blue is selected in the color list" is quite different from "Because color Blue". We have made the same scoring as for the point number 5, here 0/10 means a very bad quality (rough and machine-like) in the generated help and 10/10 is just the contrary, a very good quality (natural language). See Figure 3.2.

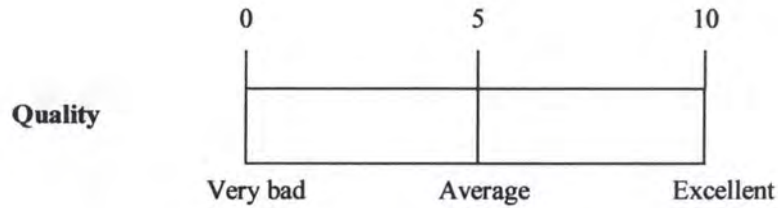


Figure 3. 2: Quality of the generated help

7. The **platform independence** is very interesting. The help-page generator does not depend on the platform (UNIX, Windows NT, Mac...).
8. The difficulty (particularly time-consuming) of **refining**, by the technical writer, the generated help. Here too, a scale is existing (Figure 3.3).

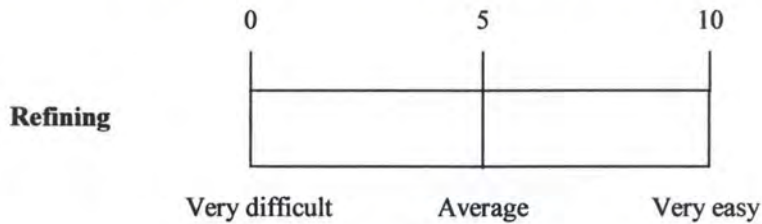


Figure 3. 3: Difficulty of refining the generated help

9. The existence of **navigability buttons**, it is much easier to navigate through the help if some buttons exist.
10. Is it the **same interface** to create the help as the end-user will use to view it? Like this there should be no surprise down the road as far as how it will look when finished.
11. The **consistency checking**. The ability to check the consistency of the help with the application being documented.
12. The difficulty (particularly time-consuming) of **updating** the generated help. It means that if the application changes, is it easy to update the associated help. A scale is existing (Figure 3.4).

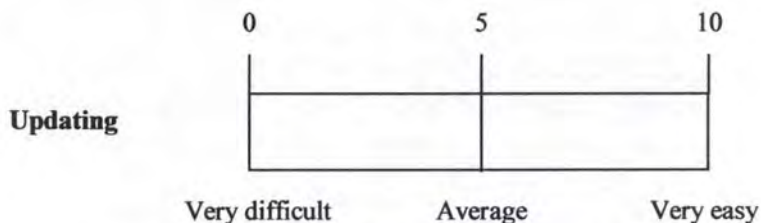


Figure 3. 4: Difficulty of updating the generated help

We now explain some tools and analyze them according to these criteria's. These tools are presented in the chronological order, from the oldest one to the newest one. The softwares are explained in details, excepting Cartoonist and H3, which are just overviewed.

3.2 Cartoonist: coupling a UI framework with automatic generation of context-sensitive animated help

The authors of this project are:

- *Piyawadee "Noi" Sukaviriya and James D. Foley*
Dept. of Electrical Engineering and Computer Science
The George Washington University - Washington

[Sukaviriya 90]

3.2.1 Help messages

Animated help is intended for help questions equivalent to the kind beginning with, "Show me how to...". A question posed to Cartoonist can be interpreted in two ways - as a question about one of the actions in the knowledge base or as a question about an activity or a task consisting of more than one action in the knowledge base. Cartoonist does not endeavor to interpret natural language questions.

Cartoonist demonstrated how to perform an action by showing a mouse icon moving on the screen onto objects with which the user must interact with, gesturing what needed to be done with the object such as which button must be pressed, and simulating the action by generating events to the underlying user interface handler. Typing was also displayed using a keyboard icon showing characters typed in while actual character codes are simulated to the underlying interface handler. An animated help scenario consisted of a series of mouse movements and keyboard animation to demonstrate how an action must be performed. Cartoonist only silently animated.

3.2.2 Architecture

Cartoonist was the first system, which generated animated help within the user interface runtime context. Cartoonist has been implemented in Smaltalk-80 running on both Macintosh II and Sun workstations; its emphasis was automatic generation of animated help from procedural knowledge. Cartoonist distinguished application and interface representations and kept the help generation mechanism independent of application-specific procedures. Cartoonist consisted of a planner which was used to fill action context with appropriate parameters, and to derive a series of action which

would satisfy the pre-conditions⁶ of an action on which help is requested. The animation algorithm has been re-implemented in C++ for HelpTalk⁷.

Currently, they have not worked on a robust interface to help yet. The interface to help is therefore still simplistic. The user clicks on a help icon, which then brings up a dialogue box where a command name or a task name can be entered.

3.2.3 Improvements

One can at least ask the question whether animation really empowers a user when carrying out a task. Until nowadays, it doesn't exist a scientific experience which has proved the value of animation.

Although we hope that animation can be valuable, merely using animation in help does not deliver a perfect help system. Minimal textual explanations must be presented with the animation to help a user generalize concepts.

It would be more exciting and useful if different animation styles such as mini-screen animation, different styles of animation story-telling, animation special effects, etc., could be supported, tested for effectiveness, and left as options for the application designer to choose among.

3.2.4 Comparison

After this overview of Cartoonist, we can compare it to Isolde following the criteria's defined in 3.1 (Figure 3.5).

Criteria	Cartoonist	Isolde
1. Type	Task-oriented	Task-oriented
2. Style	Animated	Textual
3. Hypertext	No	Yes
4. Context-sensitive, dynamic	Yes	Yes
5. Quantity of work (1)	5/10	5/10
6. Quality of the generated help (2)	3/10	6/10
7. Platform independence	No	Yes
8. Refining the generated help (3)	4/10	4/10
9. Navigability buttons	No	No
10. Same interface to create the help as the end-user will use to view it	No	No
11. Consistency checking	No	No
12. Updating (4)	6/10	6/10

Figure 3. 5: Comparison between Cartoonist and Isolde

⁶ Pre- and post- conditions in Cartoonist are represented following the first-order predicate calculus convention

⁷ Cf. infra 3.4

Comments:

- (1) The quantity of work needed to automatically generate help is average because the construction of the specifications is necessarily.
- (2) Cartoonist receives 3/10 for the quality of the generated help because this help is only animated. And we find that this help is quite poor comparatively with Isolde, moreover remember what we have written just upper concerning the impact of animated help on the users.
- (3) The refining in both softwares is quite the same because nothing has really been done to improve the possible refining.
- (4) The updating in Isolde or in Cartoonist are quite the same because we must respectively come back to the specifications or to the task model, modify these last ones and re-generate the help.

3.3 Contextual help for free with formal dialogue design

The authors of this project are:

- *Ph. A. Palanque* and *R. Bastide*
L.I.S., Université Toulouse I
- *L. Dourte*
D.I.R.O., Université de Montréal

[Palanque 93]**3.3.1 Introduction**

The benefit of this method is the ability to automatically generate an important part of the contextual help system. The contextual help should ideally answer three basic questions that the user may be interested in: "What?", "Why not?" and "How?".

The question "What?" corresponds to the interrogation "What can I do from now on?"

The question "Why not?" naturally comes to the user's mind as soon as he wishes to trigger an action whose triggering widget is currently grayed out.

The question "How?" stands for "How can I make that action available again?". The answer to this question should naturally complement that of the question "Why not?" by providing the sequence of commands to trigger in order to enable the desired command.

3.3.2 User access to contextual help

The Select-Cut-Copy-Paste functions of a word processing application will be modeled to demonstrate how users may access the contextual help.

Novice users may be puzzled if they find several inactivated menu items when they open the "Edit" menu. For example, when there is no selection on the screen, the "Cut" and "Copy" functions are not available.

In the help system, the tentative triggering of an inactivated widget is interpreted as a request for contextual help on that item. This request results in the opening of a help window (Figure 3.6).

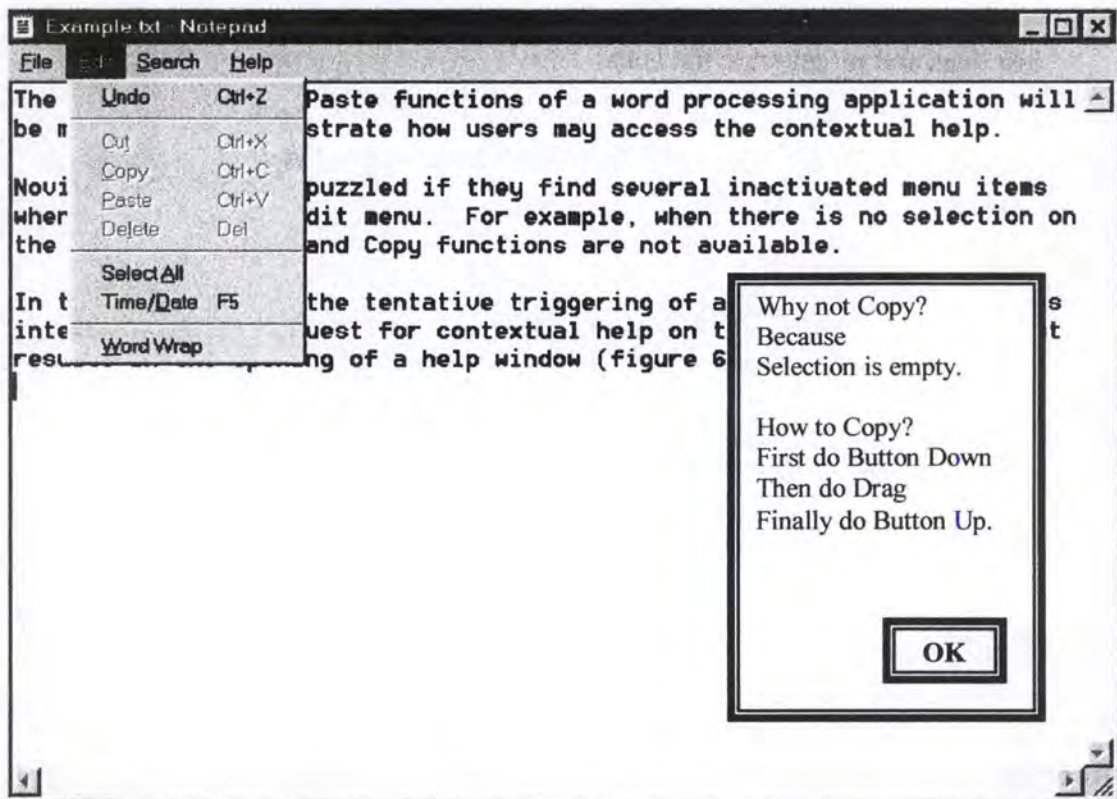


Figure 3. 6: An example of a contextual help window within a word processing application

3.3.3 Formal description of the dialogue

The high-level Petri net (HLPN) depicted in Figure 3.7 models the formal description of the "Select-Cut-Copy-Paste" functions. In the framework of the method they use a dialect of HLPNs, called Petri Nets with Objects (PNOs), which is particularly well suited for the design of user-driven interfaces because of its ability to handle objects (in the object-oriented sense) instead of simple tokens (as in regular Petri nets) in the reachability of places.

A net models the potential evolution of the dialogue in the following way: any user action is associated with one or several transitions in the net (the name of the action is inscribed inside each transition). An action may be triggered if at least one of its associated transitions is enabled in the net (i.e. each of the transition's input places holds at least one token). If the action may not be triggered (none of its associated transitions is enabled), the widget or menu item triggering this action must be grayed out. With this semantics, the question "What?" is automatically answered by the set of all enabled transitions in the net.

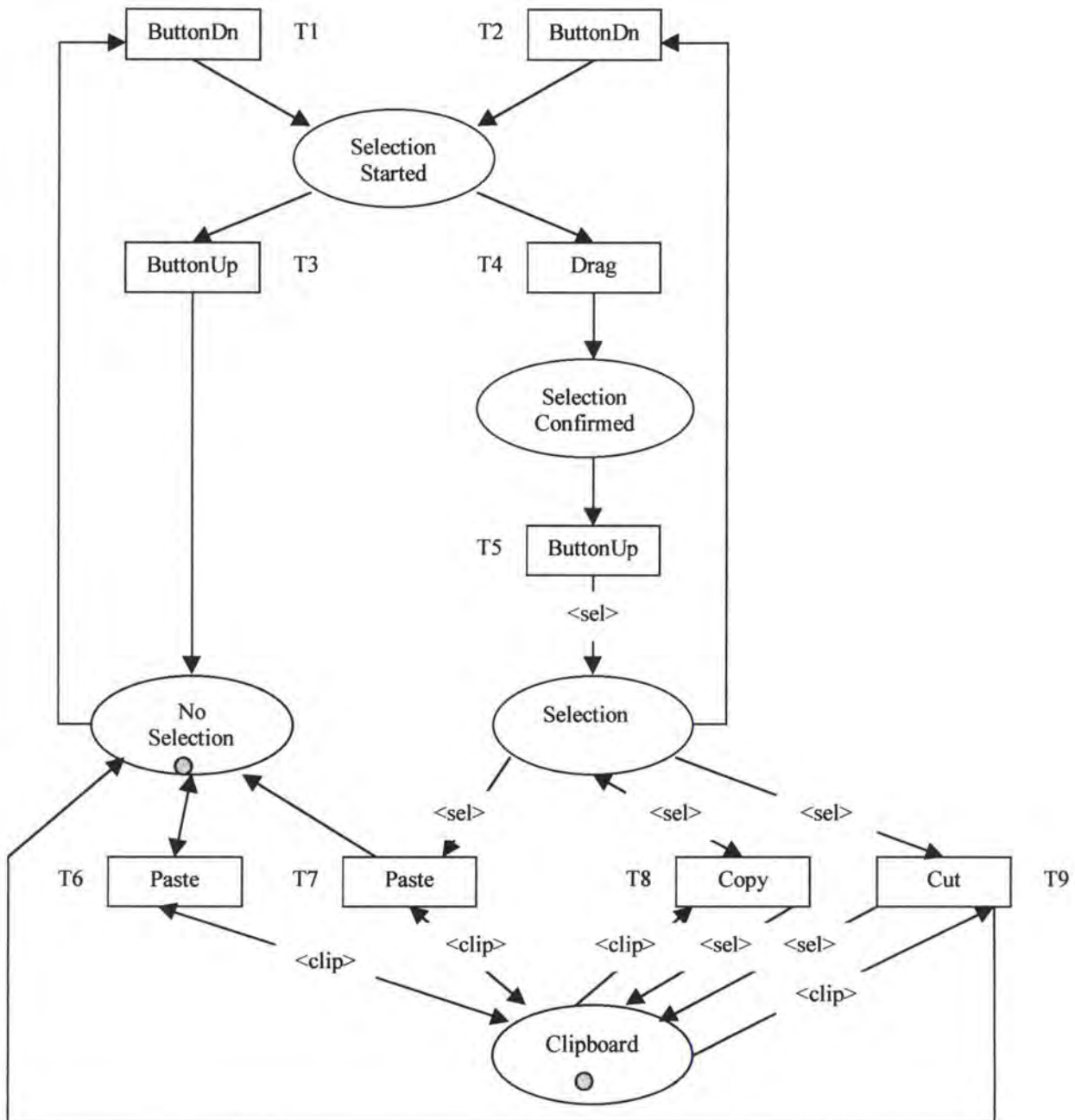


Figure 3. 7: A high-level Petri net modeling the "Select-Cut-Copy-Paste" functions

A marking m is an assignment of tokens to the places of a Petri net. The number and position of tokens may change during the execution of the Petri net. The initial marking (i.e. before the start of the execution) models the state of the interaction when the application is launched. In that state, only places *Clipboard* and *No Selection* hold a token: the clipboard is supposed to contain the results of previous interactions, and the selection is initially empty. From that state, the user may (for instance) try to select something by pushing the mouse button (transition *ButtonDn*), then dragging the mouse (transition *Drag*) and finally releasing the button (transition *ButtonUp*).

When a selection is done (i.e. there is a token in place *Selection*, and no token in place *No Selection*), the functions "Copy" and "Cut" are activated.

3.3.4 Automatic help generation

The question "Why not?" may be answered by examining the net's marking (as may be done for the question "What?"): an action is unavailable only if the current net marking enables none of its associated transitions. The question may be answered by listing all the places that lack a token in order for one of its associated transition to be enabled. For example, from the initial marking (Figure 3.7), the function "Copy" is not available because the place *Selection* holds no token. The answer to the question "Why not Copy?" is therefore "Because Selection is empty".

The answer to the question "How?" requires the use of formal techniques from Petri net theory, namely the construction of the net's **reachability graph**, which may be done automatically. In a reachability graph, each node represents a reachable marking of the net, and an arc flowing from node $n1$ to node $n2$ corresponds to the transition whose occurrence transforms marking $n1$ into marking $n2$. A reachability graph is, in finite cases, a finite states automaton, or an augmented transition network if there is an infinite set of nodes.

The reachability graph corresponding to the net in Figure 3.7 is depicted in Figure 3.8. The markings must be read in the order (*Selection Started*, *Selection Confirmed*, *No Selection*, *Selection*, and *Clipboard*). This graph is a finite state automaton whose initial state is the marking (0,0,1,0,1). From this initial state, for example, there is no arc labeled with Copy, which means that the action copy is not available.

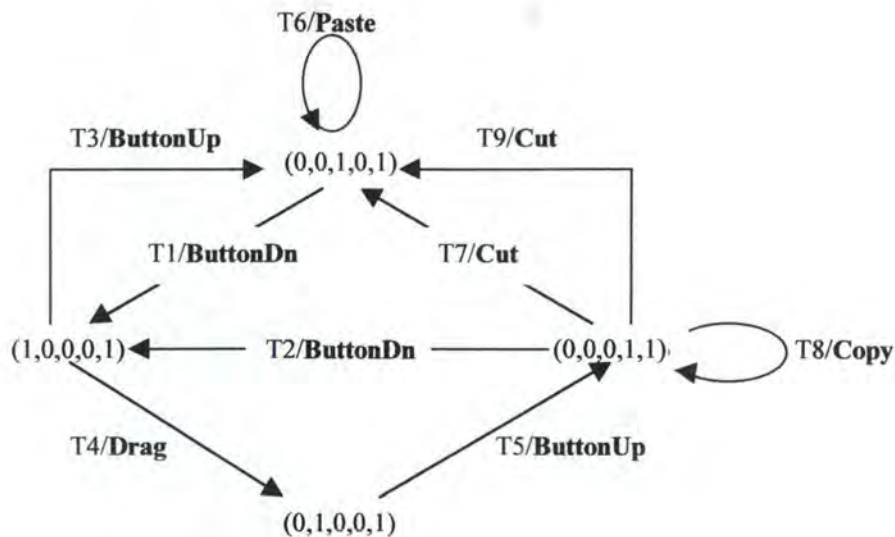


Figure 3. 8: Reachability graph of the Petri net modeling the "Select-Cut-copy-Paste" functions

To answer the question "How?", the reachability graph must be used in the following way: starting from the current state, we must proceed in a breadth first search in the graph, looking for a state featuring an output arc labeled with the desired action. When the path is found, the answer consists in listing the sequence of commands labeling the arcs on the path. The breadth first search ensures that this sequence is the shortest one.

For example, starting from the initial state outlined, the "Copy" command is not available. The search on the reachability graph provides the following path to activate it: ButtonDn, Drag, ButtonUp; the answer to question "How to Copy?" would therefore be:

First do Button Down
Then do Drag
Finally do Button Up.

Obviously, a little more work would be necessary to generate more correct English output. This could be achieved by associating a natural language help messages with each transition of the Petri net.

3.3.5 Comparison

After this explanation of "Contextual help for free with formal dialogue design", we can compare it to Isolde following the criteria's defined in 3.1 (Figure 3.9).

Criteria	Contextual help for free with formal dialogue design	Isolde
1. Type	Task-oriented	Task-oriented
2. Style	Textual	Textual
3. Hypertext	No	Yes
4. Context-sensitive, dynamic	Yes	Yes
5. Quantity of work (1)	5/10	5/10
6. Quality of the generated help (2)	4/10	6/10
7. Platform independence	No	Yes
8. Refining the generated help (3)	4/10	4/10
9. Navigability buttons	No	No
10. Same interface to create the help as the end-user will use to view it	No	No
11. Consistency checking	No	No
12. Updating (4)	6/10	6/10

Figure 3. 9: Comparison between "Contextual help for free with formal dialogue design" and Isolde

Comments:

- (1) The quantity of work needed to automatically generate help is average because the construction of a high-level Petri net representing the formal dialogue design is necessary.
- (2) A little more work would be necessary to generate more correct English output (i.e. first do Button Down, then do Drag, finally do Button Up).
- (3) The refining in both softwares is quite the same because nothing has really been done to improve the possible refining.
- (4) The updating in Isolde or in "Contextual help for free with formal dialogue design" are quite the same because we must respectively come back to the high-level Petri net or to the task model, modify these last ones and re-generate the help.

3.4 HelpTalk: automatic generation of textual, audio, and animated help in the User Interface Design Environment

The authors of this project are:

- *Piyawadee "Noi" Sukaviriya and Jeyakumar Muthukumarasamy*
Graphics, Visualisation, and Usability Centre
Georgia Institute of Technology - Atlanta

- *Anton Spaans and Hans J.J. de Graaff*
Delft University of Technology

[Sukaviriya 94]

3.4.1 Introduction

Help needs to catch up with the user interface technology. It needs to utilize the current media technology to be effective in conveying information to users. With users and interfaces becoming increasingly sophisticated, and eventually adaptive, traditional help may not suffice to help users with their specific problems.

The help research is part of broader research project called UIDE⁸. The overall objective of UIDE is to empower user interface development environments with knowledge about application semantics. The knowledge is captured through a task-oriented, high-level specification. Once captured, the knowledge is used to partially automate the user interface design process and to provide automatic runtime support such as generation of help and user task event logging. We are going to present HelpTalk, the help generation part of the project.

Text, audio, and animation are used as media to deliver automatically generated help. Currently, the automatic help generation algorithm works well but the quality of the help generated is still rough and machine-like, especially the sound bites.

3.4.2 What can HelpTalk generate?

Currently, HelpTalk generates answers to only two types of questions: "**Why** is this widget disabled?" and "**How** can one invoke this widget?" Responses to the first type of questions are currently presented as text strings, while responses to the second type of questions are presented as audio and animation. The users may not necessarily type these questions in. They could be embedded as part of an application interface.

3.4.2.1 Textual Why help

Let's take the example of a computerized reservation system for the German InterCity Express train (ICE) for German audience. The user has not chosen a city for an origin or a destination yet. Attempting to select the button labeled "Weg OK" (confirm the route selection) which is currently disabled causes the help dialogue box to pop up. HelpTalk detects two unsatisfied pre-conditions for the corresponding action, *AcceptRoute*, which states that cities must be chosen for origin and destination of the route. From these two pre-conditions, the explanation "A station has not been selected for ORIGIN and a station has not been selected for DESTINATION" is generated (Figure 3.10).

⁸ User Interface Design Environment

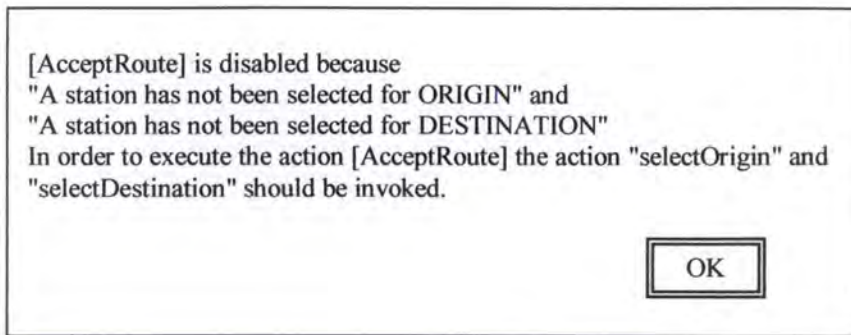


Figure 3. 10: HelpTalk's response to the user's attempt to select the disabled "Weg OK" button in an InterCity Express (ICE) train reservation system

The same help generation mechanism can be used for two different help access mechanisms. Help can be explicitly requested or the information can be voluntary when the user attempts to select a disabled button.

3.4.2.2 Audio/animated HOW help

Clicking on the right mouse button on the disabled "Weg OK" button is a signal to HelpTalk to animate what needs to be done to make the object enabled. The planner must look for actions, which will create conditions satisfying the pre-conditions of the *AcceptRoute* action. Once the planner is done, HelpTalk animates the plan by selecting an origin with the left mouse button, selecting a destination with the right mouse button, and selecting the "Weg Ok" button with the left mouse button.

Extracting information from the knowledge base, which corresponds to different parts of the animation, generates the audio. The audio states "The AcceptRoute cannot be done in this context. To perform the AcceptRoute action, we must perform SelectOrigin, SelectDestination first. To SelectOrigin, select this object using the left mouse button. To SelectDestination, select this object using the right mouse button. To perform AcceptRoute, select the "Weg OK" button using the left mouse button".

3.4.3 Help Knowledge source

The help messages are possible because procedural knowledge is captured in UIDE's knowledge base. UIDE separates its specification, or its knowledge, of an application as two separate models - the application and the interface. The application model contains action and object descriptions specific to an application domain for which an interface is designed. The interface model contains interface actions and objects, which are generic and can be used in various application domains. An interface model of a particular application consists of those interface actions and interface objects, which are chosen for the interface of this application. Actions and objects are related to each other through action-parameter relationships (Figure 3.11).

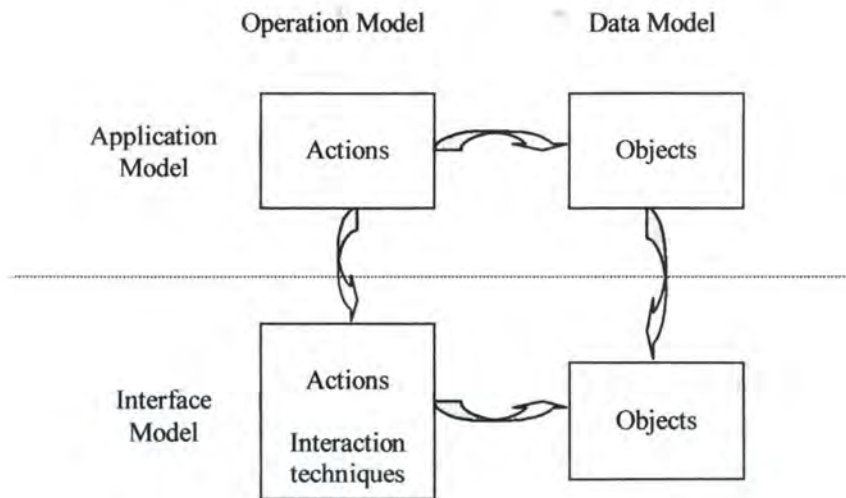


Figure 3.11: UIDE's knowledge base diagram

Since UIDE is a model-based user interface environment, a designer specifies an application by defining objects and listing actions which users can perform without an application. Notice that pre- and post-conditions are expressed in the predicate notation (Figure 3.12).

```

Action Rotate
{
    Parameters:    (gate: GATE)
                  (angle: INTEGER)
    Pre-conditions: exist (x, GATE)
    Post-conditions: angle (gate, angle)
}

```

Figure 3.12: Example of an action representation

The designer then specifies how actions in the application model maps to actions at the interface level.

3.4.4 Architecture

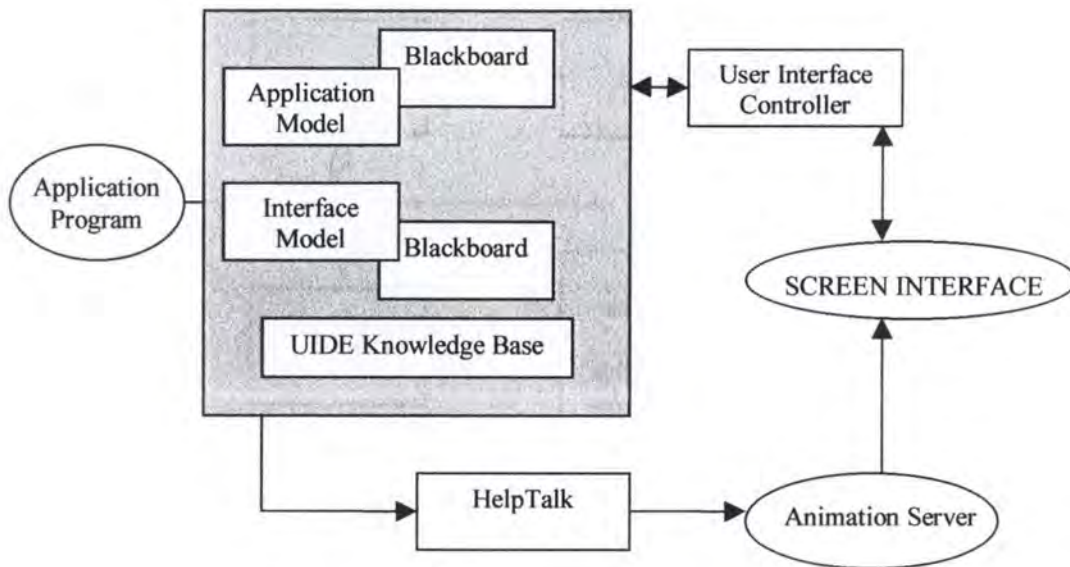


Figure 3.13: UIDE's runtime architecture with HelpTalk

Figure 3.13 shows the UIDE's runtime architecture. At the heart of this architecture is the knowledge base which is created from parsing designer inputs of application actions and how they connect to interface tasks and objects; the former is stored in the application model, the latter in the interface model. At runtime, the blackboards associated with the application and the interface models hold declarative status of the application and its interface, respectively.

The User Interface Controller (UIC) uses the application knowledge and the interface specification in the knowledge base as its source to drive the dialogue sequencing. When the user interacts with the screen interface, UIC determines which application action is invoked, processes the information, and sequences the dialogue accordingly.

Much of the information to answer how-questions is constructed from the application and the interface models combined. The blackboard contents are facts, which altogether represent the current context, allowing HelpTalk to complete context-sensitive help messages or scenarios.

Once HelpTalk is ready to show the user how to perform an action, it animates by first figuring out the animation scenario such as which objects to interact with and in which nature should the interaction be. It then sends out low-level scripts to the Animation Server, which plays out the scripts. The Animation Server runs as a separate process. It responds to low-level commands in scripts such as "move the mouse to position (20,20)" or play-audio "Select this object using the left mouse button." The Animation Server draws the mouse on the screen, moves it around, and sends X events to the application interface, which is controlled by UIC. It also sends audio information to its partner, the Audio Server.

3.4.5 Generation algorithms

A response to a **why** question is generated based on unsatisfied pre-conditions associated with an action. A response to a **how** question is generated based on traversing the UIDE knowledge model to derive at procedural descriptions.

In the textual why help, the text string could be sent to a speech synthesizer. A why explanation is generated in two parts. The first part is the reason part. The second part is the generation of what needs to be done. It could be possible to use animation as part of showing the steps.

3.4.6 Implementation

Both UIDE and HelpTalk are implemented in C++ running on Sun SPARC stations. UIDE's knowledge representation is implemented as C++ classes. At runtime, UIDE, HelpTalk, and the animation Server run as separate processes.

3.4.7 Comparison

After this explanation of HelpTalk, we can compare it to Isolde following the criteria's defined in 3.1 (Figure 3.14).

Criteria	HelpTalk	Isolde
1. Type	Task-oriented	Task-oriented
2. Style	Textual, audio and animated	Textual
3. Hypertext	No	Yes
4. Context-sensitive, dynamic	Yes	Yes
5. Quantity of work (1)	8/10	5/10
6. Quality of the generated help (2)	6/10	6/10
7. Platform independence	No	Yes
8. Refining the generated help (3)	4/10	4/10
9. Navigability buttons	No	No
10. Same interface to create the help as the end-user will use to view it	No	No
11. Consistency checking	No	No
12. Updating (4)	6/10	6/10

Figure 3. 14: Comparison between HelpTalk and Isolde

Comments:

- (1) The quantity of work needed to automatically generate help in HelpTalk is very low because the knowledge is captured through a task-oriented, high-level specification used to partially automate the user interface design process.

- (2) Text, audio, and animation are used as media to deliver automatically generated help. But the quality of the help generated is still rough and machine-like, especially the sound bites.
- (3) The refining in both softwares is quite the same because nothing has really been done to improve the possible refining.
- (4) The updating in Isolde or in HelpTalk are quite the same because we must respectively come back to the specifications or to the task model, modify these last ones and re-generate the help.

3.5 H3: automatic generation of help from interface design models

H3 stands for Humanoid Hyper help.

The authors of this project are:

- *Roberto Moriyon*
Instituto de Ingenieria del Conocimiento
Universidad Autonoma de Madrid
- *Pedro Szekely and Robert Neches*
USC/ISI California

[Moriyon 94]

3.5.1 Help messages

H3 can produce basically four kinds of help messages. The first kind is a summary message describing the item that the user selects for help. The other three kinds provide answers to the question "What commands are available?", "What is displayed here?", and "Where can I click, and what will happen?". H3 does not currently provide task-oriented help to answer questions like "How do I delete a file?" Producing such help is beyond the scope of rule systems like theirs. It requires a planner that can compute the sequence of commands needed to perform a given task.

The help information that H3 gives users goes beyond canned texts attached to the static portions of the display. H3 help messages consist of concatenations of pieces of canned texts with embedded links that show where information mentioned in the help text is displayed, and can also be used to access related information (like in hypertext systems). In addition, the messages are context sensitive so that asking for help on an item produces different messages depending on context and application state (e.g. asking for help on a dimmed item produces a message explaining why the item is dimmed, where as asking for help when the item is enabled, will tell the user what object will be affected by the corresponding command).

In the future the sophistication of the help system will be enhanced. They are interested in incorporating ideas from Cartoonist⁹ to be able to show animations, and to produce task-oriented help.

3.5.2 Architecture

If a user want to ask for help, he points with the mouse to an object on an application display and presses the HELP key on the keyboard. H3 pops up a window with the data messages about the smallest graphical or textual element under the mouse cursor.

H3 constructs a default help system for an application automatically by using the specifications used to construct the application's user interface. Developers can refine the automatically generated help system at different levels. The simplest level is to replace the default pieces of canned text by more appropriate ones. Developers can also add links to messages, and add messages to display elements that do not produce messages by default. Advanced developers can change the behavior of the help system itself by defining new rules in the H3 rule system that computes the help messages. Humans are much more skilled than computers at writing prose, and since the quality of the text is critical to the success of the help system, H3 allows humans to write the text.

In H3 the help messages are specified as rules of the following form:

When *Conditions* **then** *Message-Descriptions*

The conditions identify the context when a particular message is appropriate, and the message descriptions are templates that specify the text and links of messages to be generated.

3.5.3 Comparison

After this overview of H3, we can compare it to Isolde following the criteria's defined in 3.1 (Figure 3.15).

⁹ Cf. supra 3.2

Criteria	H3	Isolde
1. Type	Widget-oriented	Task-oriented
2. Style	Textual	Textual
3. Hypertext	Yes	Yes
4. Context-sensitive, dynamic	Yes	Yes
5. Quantity of work (1)	8/10	5/10
6. Quality of the generated help (2)	4/10	6/10
7. Platform independence	No	Yes
8. Refining the generated help (3)	8/10	4/10
9. Navigability buttons	No	No
10. Same interface to create the help as the end-user will use to view it	No	No
11. Consistency checking	No	No
12. Updating (4)	6/10	6/10

Figure 3. 15: Comparison between H3 and Isolde

Comments:

- (1) The quantity of work needed to automatically generate help in H3 is very low because it uses the specifications used to construct the application's user interface and nothing else.
- (2) The quality of the generated help is not so good in H3 but the users can refine the help to improve its quality. So finally the help will be of good quality, after the refining of the users but no initially!
- (3) The possibility to refine the generated help is very good in H3. Three levels of refining are possible. See 3.5.2 for more details.
- (4) The updating in Isolde or in H3 are quite the same because we must respectively come back to the specifications or to the task model, modify these last ones and re-generate the help.

3.6 CogentHelp: a tool for authoring dynamically generated help for Java GUIs

CogentHelp is a prototype tool for authoring dynamically generated on-line help for applications whose Graphical User Interfaces (GUIs) are built with the Java Abstract Windowing Toolkit (AWT).

The authors of this project are:

- *David E. Caldwell and Mickael White*
CoGenTex, Inc.

[Caldwell 97] and [CogentHelp]

3.6.1 Design goals

The first goal was to effectively assist authors in creating high-quality on-line help documents.

It means:

- Consistency - the general writing style should be consistent throughout the help system ;
- Navigability - the use of grouping and formatting should make it easier to find information about a particular GUI component in the help system ;
- Completeness - all GUI components should be documented ;
- Relevance ;
- Conciseness - redundancy should be avoided ;
- Coherence - information about GUI components should be presented in a logical and contextually appropriate fashion.

The second goal was to support the maintenance of help documents as the documented application evolves.

It means:

- Fidelity - the help author should be assisted in producing complete and up-to-date descriptions of GUI components ;
- Reuse - the help author should not have to write the same text twice.

And the final goal was to facilitate use of the technology deployed to achieve the first two goals. The benefits of the system must be made available at a reasonable cost in terms of the understanding and effort required of the help author.

3.6.2 System

The help authors write the reference-oriented part of an application's help system in small pieces (or "snippets"), indexed to the GUI components themselves. CogentHelp then dynamically assembles the snippets into a set of well-structured help pages for the end-user to browse.

This general approach (storing many small document components in a structured database, and using them to generate complex and/or varied documents) has two main

advantages over a "manual" approach to document creation. First it makes easier to maintain documents. Secondly, having a document generator frees the author to concentrate on writing accurate content for the snippets, rather than the drudgery of applying consistent formatting and managing a complex hypertext network.

Consistency is maintained through the use of utilities, which check for missing or surplus snippets.

Elements of this approach have been used in many systems for software-related documentation but CogentHelp is the first tool to combine the advantages of automatic consistency checking and automatic document generation in a tool for authoring documentation directed at end-users.

3.6.3 Technically

CogentHelp's help-page generator is implemented using Java and HTML for platform independence, and with a client-server architecture in order to allow dynamic generation of help pages which reflect the current state of a GUI. Help topics are delivered by an HTTP server via the Java Servlet API¹⁰; for display in a Web browser. The expandable table of contents and thumbnail hypergraphics are also implemented in Java, as applets hosted by the browser.

3.6.4 Concretely

When we click on Help button, this brings up the help system in a new window (Figure 3.16). This window serves a dual purpose: it is help window that the end-user of the applet will see, but it also hosts CogentHelp's authoring interface. There should be no surprises down the road as far as how it will look when finished!

¹⁰ Application Program Interface: a set of libraries

3.6.4.1 The help window

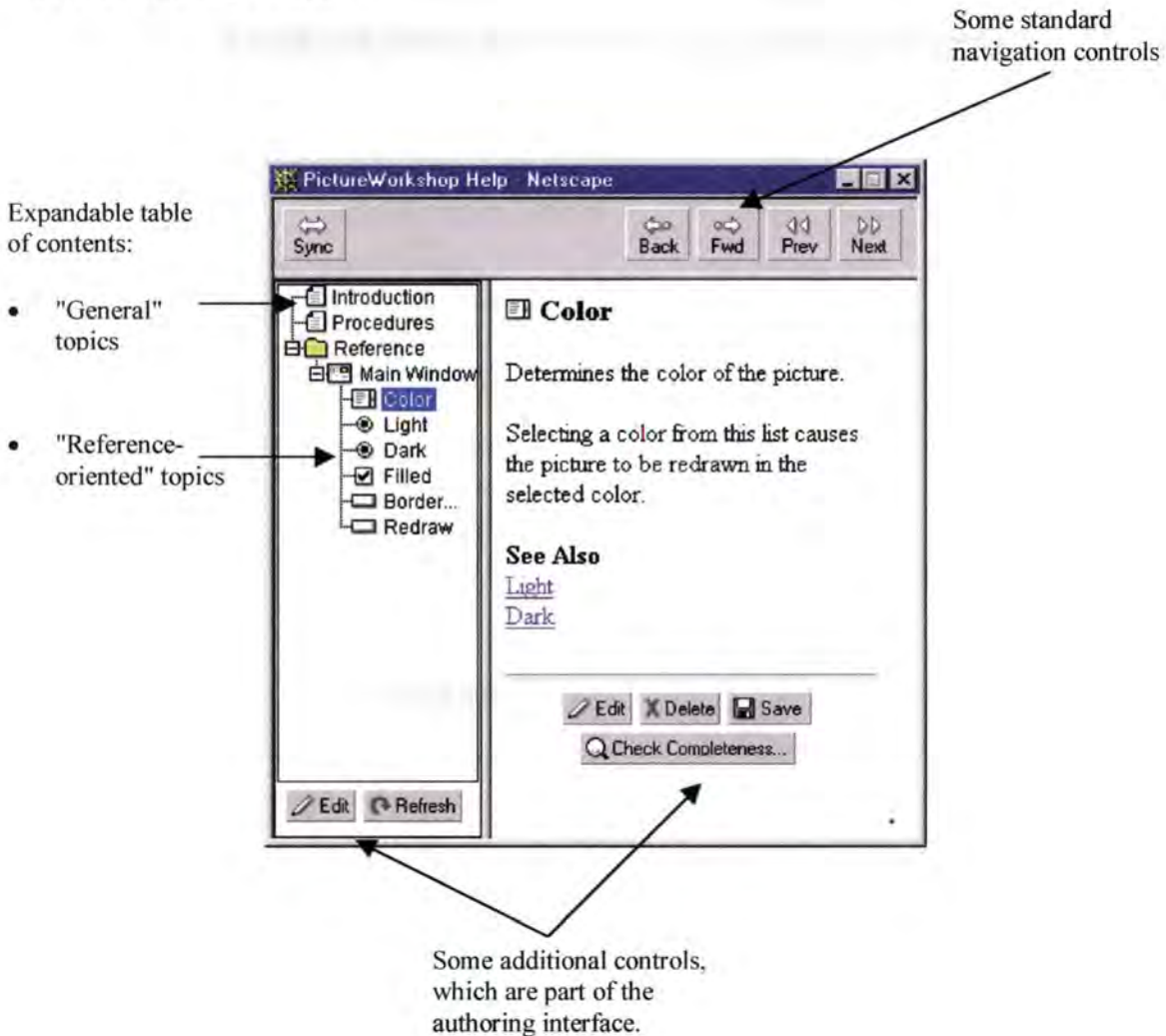


Figure 3. 16: The "Help" window

When we have finished authoring the help system, we will switch it to end-user mode, and the additional controls will no longer appear.

"General" topics give high-level information about the application, task-oriented help, etc.; they are written by hand, using a HTML editing tool. And "reference-oriented" topics describe the functions of individual windows and widgets, they are generated automatically by CogentHelp using text that is provided in the form of "help snippets".

3.6.4.2 Help snippets

If we click on the Edit button shown in the "Color" topic, this brings up a window (Figure 3.17) in which we can edit the help snippets.

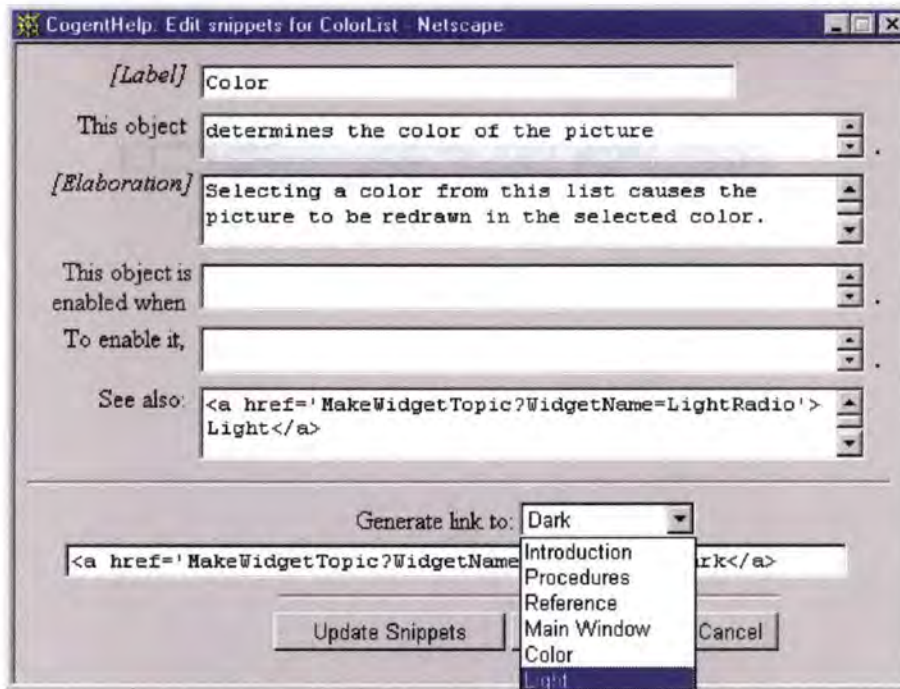


Figure 3. 17: The "Edit snippets" window

Each of the snippets contains a different type of message concerning a widget. The first one is simply a label to be used to refer to the widget in the help system. The second snippet is simply a brief description of the widget's function. The third is an optional elaboration, which may be a paragraph or so of further explanation. The fourth and fifth describe conditions under which the widget is enabled, where this is appropriate, and how to enable it if it is disabled. Finally, the last snippet simply contains a list of hypertext links to other relevant help topics.

The snippets are written in HTML, this lets us enter anything from plain text, to bold, italics, and hypertext links; to audio, video, and whatever else is expressible using HTML.

The window also contains a handy item, which can be used to generate a hypertext link to any topic in the table of contents.

3.6.4.3 Consistency checking

The most important feature is the ability to check the consistency of the reference-oriented help with the application being documented. If we click on the "Check Completeness" button in the help topic window, this displays a report that lets us know how we are doing (Figure 3.18).

CogentHelp tells us about any "obsolete" widgets in the help system - ones for which we have created help topics, but which no longer exist in the application. CogentHelp also lets us know about new widgets for which help topics have been created, but for which we have not actually entered any help snippets yet.



Figure 3. 18: The "Consistency check" window

3.6.4.4 Editing the table of contents



Figure 3. 19: The "Edit table of contents" window

This window (Figure 3.19) lets us move the nodes in the table of contents around as we see fit. It is also possible to use this window to specify how hand-written "general" topics fit into the table of contents, by creating new nodes and specifying the URLs and images associated with them. So CogentHelp also gives us lots of flexibility to integrate reference-oriented help with our hand-written topics.

3.6.4.5 Dynamic help topics

The kind of help topic, whose contents change according to the current state of the application, is possible because one of the bits of information that is

passed to the help server about each widget when a help request is made is a string representing its current "state" (Figure 3.20).



Figure 3. 20: Dynamic help topics

3.6.5 Conclusion

- It concentrates the contents of help topics into one unique location, so that users are sure to get the last update as soon as available.
- Developers of the help system are no longer forced to duplicate recent update of a help system since the update is available on the WWW¹¹.
- It automates much of the "drudge" work involved in creating reference-oriented help, letting us concentrate on getting the *content* right.
- It helps us to maintain consistency between the help and the application we are documenting.
- It gives us flexibility in how we incorporate hand-written materials into the help system.
- It can be customized in various ways, including the way in which it generates dynamic help messages according to the current state of the application.

¹¹ World Wide Web

3.6.6 Comparison

After this explanation of CogentHelp, we can compare it to Isolde following the criteria's defined in 3.1 (Figure 3.21).

Criteria	CogentHelp	Isolde
1. Type	Widget-oriented	Task-oriented
2. Style	Textual	Textual
3. Hypertext	Yes	Yes
4. Context-sensitive, dynamic	Yes	Yes
5. Quantity of work (1)	3/10	5/10
6. Quality of the generated help (2)	8/10	6/10
7. Platform independence	Yes	Yes
8. Refining the generated help (3)	7/10	4/10
9. Navigability buttons	Yes	No
10. Same interface to create the help as the end-user will use to view it	Yes	No
11. Consistency checking (4)	Yes	No
12. Updating (5)	6/10	6/10

Figure 3. 21: Comparison between CogentHelp and Isolde

Comments:

- (1) It takes more time to fill ALL the snippets for ALL the widgets than to make a task model for the application.
- (2) The quality of the generated help for the CogentHelp project is much better than the one from Isolde. That is quite normal because the sentences of the help are the ones coming from the snippets and thus, we determine ourselves the quality of the help.
- (3) To refine the help with CogentHelp, it is very easy: we just change the right snippet. What we put in it will be what we see in our help.
- (4) The Isolde project does not have the consistency checking, that is a disadvantage comparatively to CogentHelp but it is not absolutely necessary. It is just like an option for a car; we have it, quite well, otherwise do not worry ... we still have the car.
- (5) For the help's updating in CogentHelp, it is very easy. Just change the snippets if the specifications of the widgets have changed. Moreover we have the consistency checking which keep the coherence and the consistency of the help.

CHAPTER 4

The task model

Resume:

In chapter 3, we have compared Isolde with the other help generation systems.

This chapter introduces the formalism used for Isolde, describing the different diagrams in UML and their relation with task models.

The next chapter will describe and examine the tools used to develop our task model editor : Java and the GUI builder *VisualAge*.

After a short introduction (section 4.1) explaining why the task model was the optic chosen, section 4.2 introduces Rational Rose, the case tool used in the Isolde project, and some of the diagrams generated by Rational Rose that are useful to describe task models.

Finally, in section 4.3, we describe Diane+H, the formalism we use to represent a task model.

4.1 Introduction

Knowledge representation is one of the most important issue of a tool aimed at producing on-line help. The Object Oriented approach to software development has become very popular. Yet, object oriented development is mostly system centered and not adapted to user usage and interaction patterns [Lu 98a].

On the other hand, task analysis provides a user centered view and is designed to describe users' task.

Popular in interactive software development, task models can help prototyping of the user interface or can be used as a communication tool between the software engineers, the HCI specialists, the end-users and the technical writers.

Task analysis and modeling provide OO with its much needed strength. However, in order to integrate OO and task modeling, a proper mechanism has to be in place. The challenge was to integrate task modeling into the industrial OO software development process.

The case tool we will use is Rational ROSE, in conjunction with a TAsk MOdeling Tool (TAMOT). That tool is the Task model editor we have developed during our training period at CSIRO.

4.2 A case tool : Rationale Rose

In order to produce a model, we have two options : the first one is to start from scratch. It is difficult, time consuming and a model cannot be reused [Lu 98a].

The other possibility is to use some information already available and transform it into a formal graphical notation in order to be easily understood and modified later, using the TAMOT.

The first step is to extract a primal sketch of a particular model and then add information manually to turn it in a more complex model. Software developers use CASE tools to specify programs. An idea would be to reuse information from a CASE tool and make a task model from there.

In the Isolde project, we use Rationale Rose, an Object Oriented CASE tool frequently used in software development.

TAMOT is integrated with ROSE through two transformers : BTT (system Behavior model To Task model) and TTB (Task model To system Behavior model) which exploit the semantic common ground between system behavior models and task models. BTT, built in ROSE script, automatically constructs corresponding task models from system behavior models whereas the TTB does the opposite task. This gives a way to feed the result of task modeling into ROSE for subsequent design.

System behavior models in an OO CASE tool like Rationale Rose typically include use cases, use case diagrams, interaction diagrams and state transition diagrams. We will focus on use case and use case diagrams (section 4.2.1), interaction diagrams (section 4.2.3), state diagrams (section 4.2.5) and see respectively in section 4.2.2, 4.2.4 and 4.2.6 the connection to task model [UML 97].

4.2.1 Use cases and use case diagrams

A use case diagram is a graph of actors, a set of use cases enclosed by a system boundary, communication (participation) associations between the actors and the use cases, and generalizations among the use cases.

A use case is shown as an ellipse containing the name of the use case.

The following relationships are meaningful within a use case diagram :

- **Communicates** : The participation of an actor in a use case is shown by connecting the actor symbol to the use case symbol (solid path) ;
- **Extends** : ‘Extends’ is a relationship between use cases and is shown by a generalization arrow from the use case providing the extension to the base use case. A relation from use case A to use case B means that an instance of use case B may include the behavior specified by A ;
- **Uses** : ‘Uses’ is also a relationship between use cases. A relation from use case A to use case B means that an instance of use case A will also include the behavior as specified by B.

In Figure 4.1, we can observe a use case relationship where an instance of “Place Order” includes “Arrange payment” and where an instance of “Place Order” may include “Request Catalog with Order”

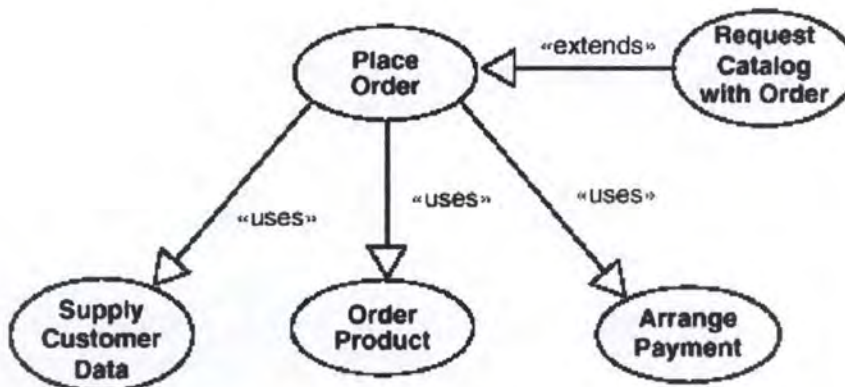


Figure 4. 1: A use case relationship

Figure 4.2 is an example of a use case diagram where there is a communication association between the Customer and the Salesperson for the use case “Place Order”

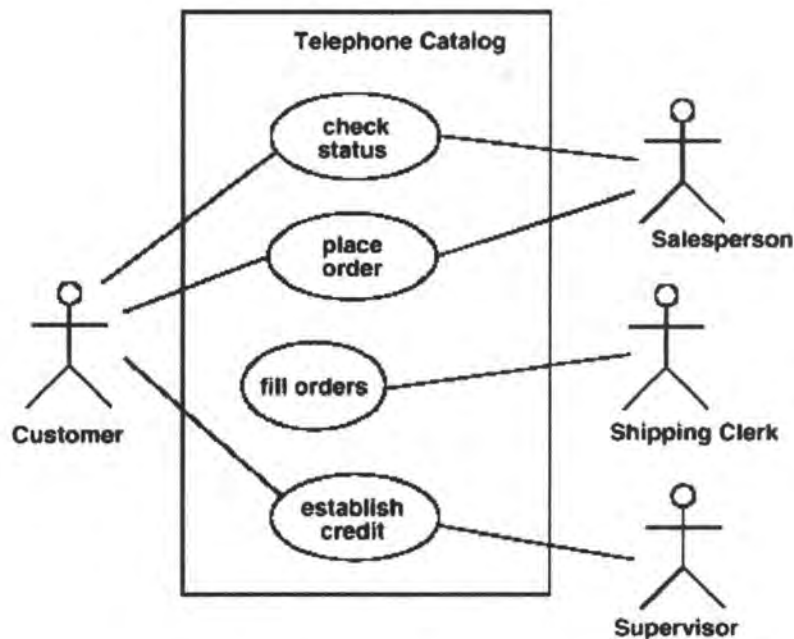


Figure 4. 2: A use case diagram

4.2.2 Use cases and Task models

There is a strong similarity between use cases and task models [Artim 97]. Use cases and use case diagram are semantically similar to task models [Lu 98b].

Use cases capture user requirement for a system by describing how a system will be used and to what ends. We can compare use cases and task models from the following points of views:

- A use case can be seen as equivalent to a composite task in a task model ;
- Some tasks attributes, like task name and comment, can be obtained directly from use cases. For example, a task name can be seen as its corresponding use case ;
- Some information like task precondition and feedback, needed for task model and contained in use cases, are more difficult to extract automatically. Since it is part of a free text, to single it out would require sophisticated language processing.

Besides the characteristics of use cases, use case diagrams have the following features:

- All use cases inside a use case diagram can be seen as sub-tasks of the composite task corresponding to the use case diagram ;
- The **extends** relationship between use cases is equivalent to a sequence with a precondition ;
- The **uses** relationship between use cases is equivalent to a sequence ;
- Unrelated use cases are equivalent to tasks parallel to each other.

4.2.3 Interaction diagrams

A pattern of interaction among objects is shown on an interaction diagram. Interaction diagrams come in two forms based on the same underlying information but each emphasizing a particular aspect of it: **sequence** diagrams and **collaboration** diagrams.

Sequence diagrams :

A *sequence diagram* shows an interaction arranged in time sequence. In particular, it shows the objects participating in the interaction by their “lifelines” and the messages that they exchanged arranged in time sequence. It does not show the associations among the objects.

A sequence diagram has two dimensions: the vertical dimension represents time, the horizontal dimension represents different objects. Normally time proceeds down the page. (The dimensions may be reversed if desired.)

Usually, only time sequences are important, but in real-time applications, the time axis could be an actual metric. There is no significance to the horizontal ordering of the objects. Objects can be grouped into “swimlanes” on a diagram.

In Figure 4.3, we can see a simple sequence diagram with concurrent objects.

From the time the caller lifts the receiver (time a), meaning that he wants to make a call, it takes less than a second for the system to send a dial tone to the caller. Then, the caller must start dialing the phone number within 10 seconds, etc.

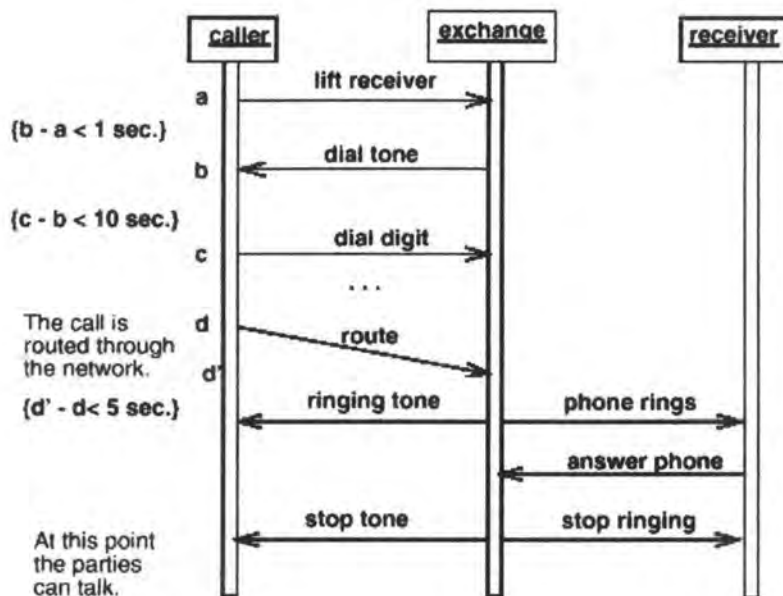


Figure 4.3: A sequence diagram

Collaboration diagrams :

A collaboration diagram shows an interaction organized around the objects in the interaction and their links to each other. Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects. On the other hand, a collaboration diagram does not show time as a separate dimension, so the sequence of messages and the concurrent threads must be determined using sequence numbers.

A collaboration is a modeling unit that describes a set of interactions among types. A collaboration involves two kinds of model constructs: a description of the static structure of the affected objects, including their relevant relationships, attributes, and operations; and a description of the sequences of messages exchanged among the objects to perform work.

A collaboration diagram is a graph of objects and links with message flows attached to its links. The context of the diagram shows the objects relevant to the performance of an operation, including objects indirectly affected or accessed during the operation.

The context for an operation includes its arguments and local variables created during its execution as well as ordinary associations. Objects created during the execution may be designated as «new»; objects destroyed during the execution maybe designated as «destroyed»; objects created during the execution and then destroyed may be designated as «transient».

The invoker of an interaction may be shown on a collaboration diagram as an actor symbol. The internal messages that implement an operation are numbers starting with number 1. For a procedural flow of control the subsequent message numbers are nested in accordance with call nesting. For a nonprocedural sequence of messages exchanged among concurrent objects, all the sequence numbers are at the same level (that is, they are not nested).

In Figure 4.4, we can see a collaboration diagram

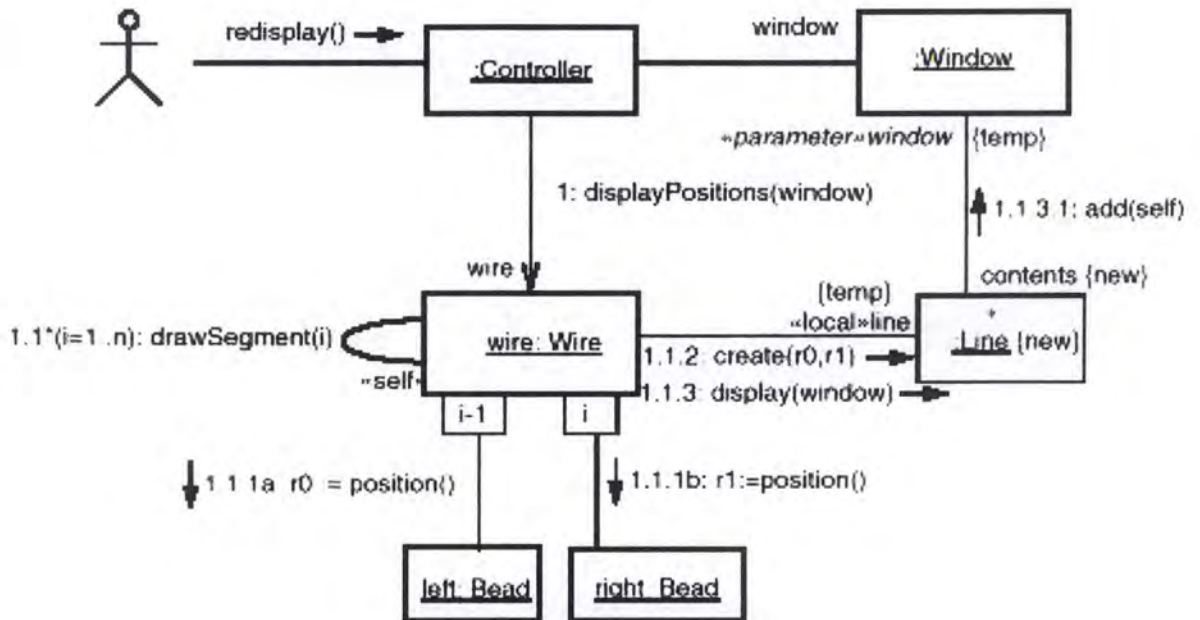


Figure 4. 4: A collaboration diagram

4.2.4 Interaction diagrams and Task models

Since sequence diagrams and collaboration diagrams can be automatically generated from each other, either can be used. Arbitrarily, sequence diagrams are used in our studies.

The question is : “How much information contained in sequence diagram could be reused to generate task models ?”

Based on analysis, we can conclude than task models in part resembles sequence diagrams. A task model describes only users’ and system’s behavior while sequence diagrams represent system’s internal behavior as well.

In general, we have the following observations :

- A sequence diagram can be seen as equivalent to a composite task in a task model ;
- A message between objects is equivalent to an elementary task ;
- All messages inside a sequence diagram can be seen as sub-tasks of the composite task corresponding to the sequence diagram ;
- A lot of information in a sequence diagram is irrelevant to its corresponding task-model, such as most interactions between invisible objects ;
- Some task attributes (comments, ...) can be obtained automatically from sequence diagrams. They can be extracted from the corresponding message ;
- Some task attributes (Interactive or manual, terminal event or not) can be implied from sequence diagrams ;
- Some information contained in sequence diagrams and needed in task models may be difficult to extract, such as sequence preconditions, task precondition, and feedback.

With the rules listed above, algorithms can be designed to automatically construct task models from sequence diagrams.

4.2.5 State (transition) diagrams

A state diagram shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions.

In the example of Figure 4.5, if we start from state “DialTone” and the caller dials an invalid digit, the system generates a message. But if the dialing is valid, the system connects the caller to the receiver and the action “Ring” starts.

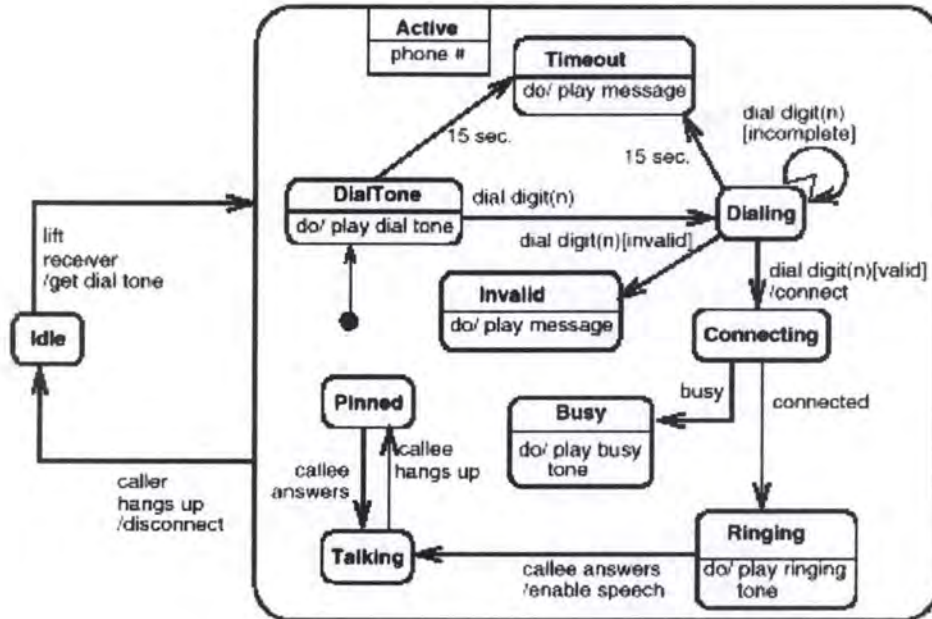


Figure 4. 5: A state diagram

4.2.6 State (transition) diagrams and Task models

State diagrams for a single object is of limited use because an end-user task is normally delivered through interaction between multiple objects.

State diagrams for the system seem to be more interesting for task models. Unfortunately, this kind of state diagram seems to be very hard to build.

That is why we will not go further with state diagrams for now, but they might be used in the future.

4.3 The Diane+H formalism

DIANE+ [Tarby 96] extends the DIANE method to make possible the automatic generation and the automatic management of the user interface.

DIANE+H is a sub-set of DIANE+, used within the Isolde project, in the context of automatic on-line help generation. DIANE+'s goal is to automatically manage the interaction during the execution, while DIANE+H is to automatically generate on-line help.

In order to have control over the man-machine dialogue, we need to define a few concepts : these concepts are the *operation* and the *precedence*. A precedence is a sequencing link between operations. An operation is either a process which can be

performed (e.g. print the screen), a set of operations (called sub-operations), sets of sub-operations, and so on.












The questions we consider are :

- **Who triggers an operation ?** The triggering is *optional* when it is the user, and *automatic* when it is the computer. In the first case, the user can trigger the operation and decides when to trigger it. In the second case, the user cannot decide to trigger the operation ;
- **Who performs an operation ?** The operation is *manual* if it is the user, (e.g., sign a document), *automatic* if it is the computer (e.g., disconnect), and *interactive* if it is both (e.g., enter a name) ;
- **Who checks the performing of an operation ?** The operation is *optional* when the user checks, and *required* when the computer checks. Example: for the "Record a client" aim, the "Enter the name of client" operation is required whereas the "Print a client" operation is optional. All operations of consultation, printing, etc., are generally optional.

Another kind of operation exists in DIANE+ : the *constrained* operation. A constrained operation results of the splitting of an operation into sub-operations, whereas a constraint is associated in order to define how many sub-operations must be performed.

Another feature implemented is the boolean connector. It allows multiple branching from a task to another. The available boolean connectors are AND, OR and XOR.

We illustrate some graphical notations in Figure 4.6

Task Attribute	Graphical form	Explanation
Interactive		User interactive with system
Manual		Performed by user
Automatic		Performed by System
Elementary		Task box without shade
Composite		Task box with shade
Decomposition		Task refinement
Feed back		Describe feedback provided by the application
Mandatory		Task box in solid lines.
Optional		Task box in dotted lines
Parallelism		Tasks can be performed in parallel
Task sequence		Order of task to be performed




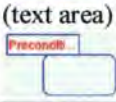
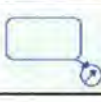
Sequence precondition.		Indicates the condition under which the link will be followed.
Constraint		The task will be able to be executed a minimum of I and maximum of j times.
Name		Task name
Comment	(text area)	Task Comment
Task precondition		The task will be blocked until the precondition becomes true.
Terminal		The normal end of the task

Figure 4. 6: Some graphical notations

CHAPTER 5

Development softwares

Resume:

In chapter 4, we have seen the formalism used in Isolde to describe a model in Rationale Rose and how some diagrams could be used to make a task model.

In this chapter, we analyze the development softwares we have used during our training period for the implementation of Isolde: Java and a builder from IBM, VisualAge. The advantages and the disadvantages of both softwares are presented.

The next chapter will describe the architecture of our system and the different classes and methods in Java.

Section 5.1 gives us an overview of the Java language.

In section 5.2, we analyze the builder we have used to develop our interface: VisualAge.

We finish (section 5.3) with a remark concerning our experience with both softwares.

5.1 Java

Java is a high-level programming language similar to C, C++ and Pascal. That is a simple, object-oriented, compiled, platform independent, multi-threaded, robust and extensible language.

5.1.1 The origins

Java has been created and implemented by a small team headed by James Gosling¹² at Sun Microsystems in California. This team was working on the development of an application. After a while, they found that programming languages like C or C++ were not appropriate. The programs written in C++ must be compiled for a particular environment. Each time we are changing of environment, a new compilation has to be made. Moreover the C++ programs must be recompiled when the libraries used are changing.

In 1990, James Gosling began the creation of a new programming language that could not have the disadvantages of the traditional languages like C or C++. Java is the result.

5.1.2 Why this name: Java

At the beginning, James Gosling called this language "Oak" (from the tree situated opposite the window of his office). After, the development team found that this name was already existing, it was the name of a programming language. That is the reason why another name had to be found. After discussions inside the team, Java was chosen. Java does not stand for Just Another Vague Acronym!

5.1.3 Advantages

5.1.3.1 Java is simple

Most of the programmers are using C and those working with the object-oriented programming are using C++. Java is very similar to C++, he has less functions (those which are rarely used or not absolutely needed) because of simplifications. The most differences between Java and C++ are:

- There is no pointer because there are a lot of bugs in the C++ programs. They are replaced with objects and with tables of objects. The complex manipulation of pointers is so replaced with access to tables via indexes ;
- The allocation and the deallocation are automatically made by the way of a garbage collector. The system decides by himself which are the non referenced objects and frees them ;

¹² He is also responsible for the development of the Emacs (UNIX) and NeWS windows system.

- Every method is linked to a class, there are no global functions like in C. Moreover these classes can be loaded dynamically. The virtual machine only uses the needed classes. As soon as a new class is needed, this last one is memorized;
- The Java language has got exceptions that are represented with classes. They allow responding to a non-foreseeable situation for the program without forcing the stop of the program.

One of the purposes of the Java language is to allow the construction of applications that can be run on small machines. The size of the minimal interpreter and its classes is 40 kbytes, if we are adding the standard libraries, the size will be 175 kbytes.

Personally, we find that this advantage is particularly true. This language is very easy to understand and to work with. After a few days, it is possible to have a very good idea of how to develop applications. It is much easier than C++; for example, there is no pointer and happily it is possible to do at least the same things as with C++. But also with the automatic memory (de)allocation, the dynamic loading and the exception classes. The fact that applications developed in Java can be run on small machines is also very interesting in our case, because one of the purposes of Isolde is the possibility to be run on every machine.

5.1.3.2 Java is object-oriented

Java is an object-oriented programming language. To be considered as object-oriented, a language must have at least these four characteristics:

1. Encapsulation: regrouping the methods and its variables inside a class. With this system, it is impossible to access directly the variables of an object ;
2. Heritage: relation between classes allowing a sub-class to have its attributes and to share methods defined in the upper class ;
3. Polymorphism: processes allow having multiple implementation of a method identified by its name. The polymorphism is implemented with the dynamic binding ;
4. Dynamic binding: technique allowing to link the calls of the procedure (methods) to the corresponding code at the time of the execution and not at the time of the compilation.

The Java language has got the characteristics below. For example, it is divided into classes. Each class is containing a set of methods, which is defining the comportment of this object. A particular class can inherit comportments of other classes. The root of the classes hierarchy is already the *Object* class.

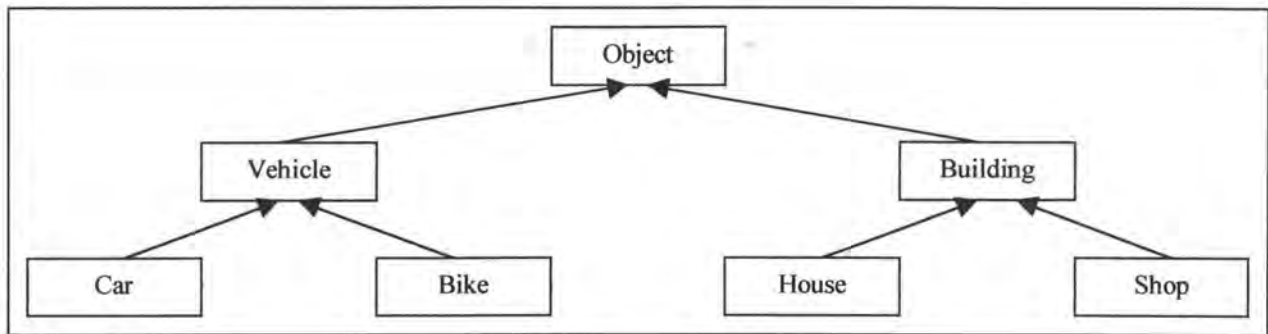


Figure 5. 1: Classes hierarchy

For example in Figure 5.1, we could define a class *Car* that is inheriting properties of the class *Vehicle*, this last one is inheriting properties from the class *Object*.

Java has only the simple heritage. It means that each class can inherit methods from only one other class. Some language have multiple heritage, this can introduce some confusion and make the language too complicate.

During the declaration of a class in Java, we must indicate the allowed access types to the variables and to the methods of the class. The classes can be declared:

- ✓ *public* - the methods and the variables are available for all the other classes ;
- ✓ *protected* - the methods and the variables are only available for the sub-classes of this class and nowhere else ;
- ✓ *private* - the methods and the variables are only available inside the class in which they have been declared, they are not available for other classes, even for the sub-classes.

The object-oriented philosophy of Java is quite powerful, we can re-use objects from previous applications or from libraries made by others.

This last point has been an advantage in the development of our application because we have re-used objects from others. The WWW is a good source to find objects that can be re-used in applications.

5.1.3.4 Java is compiled

Before the execution of a Java program, we must transform it to byte-code. Byte-code is very similar to machine instructions, it allows the programs written in Java to be very effective. Nevertheless byte-code is not specific to a particular machine. So it can be run on a very large range of different computers without recompiling the start program.

The fact that Java is compiled has already the same advantage for our application. We mean that our task modeling tool has the possibility to effectively be run on every machine.

5.1.3.5 Java is platform independent

Seeing that Java programs are compiled in byte-code, they can be run on every platform accepting Java. There is no need to recompile a Java program to run it on another computer.

Java language is identical for every computer type. Surprisingly it is not the case for modern programming language like C or C++. Each compiler and each development environment is lightly different; it makes the portability problem very hard to solve.

Java system gives also an exhaustive classes library, which gives us access to the operating system. Here are some of the major ones:

- Java.util: useful classes like vectors, enumeration, properties, etc ;
- Java.io: classes managing the input/output flow. They give us access to the file system ;
- Java.awt: (Abstract Window Toolkit) – tools for creating graphical user interfaces.

Personally, we have made the task modeling tool on a Windows platform and this application was correctly running on a UNIX platform, excepting that ALL the interactive objects were something like 25 pixels upper than with the Windows version. So after having changed this, it was correctly running.

5.1.3.6 Java is multi-threaded

The multi-threading allows to program applications that are realizing several parallel operations. Most of the modern computer systems like Unix and Windows95 allow the multi-task. Java also offers this possibility.

A Java program can have more than one task to run in parallel.

It is often very difficult to program with this type of environment because a lot of events can occur at the same time or in a non-foreseeable order. Java gives synchronization tips that are facilitating the programming by the way of the *Thread*¹³ class. This class also contains a set of methods allowing to run or to stop the execution of a thread and also to verify its status.

We have not worked with the multi-threading in our application, so it was not really an advantage for us. But in some application, it can be very interesting.

¹³ This class belongs to the *java.lang* library of the Java language

5.1.3.7 Java is extensible

It is also possible to use Java programs inside other programs written in other languages. Seeing that the data structures of Java are likely to the C ones, it is relatively easy. The biggest problem is that most programs are not multi-threaded.

This seems very interesting but we did not have to use it during our training period.

5.1.3.8 Java is robust

Java has been imagined to develop applications that have to be robust. The accent is put on the detection of errors as fast as possible. So the Java compiler is using a big part of its time to detect syntax errors, before that the program is distributed.

One of the advantages of typed language (like C++) is that some errors can be detected during the compilation. Java imposes the programmer the explicit declarations of variables, methods and their types, contrarily to the C in which we can make implicit declarations.

Even the Java program can contain bugs. If an unexpected situation occurs, the program does not stop, it is generating an exception. The Java interpreter will find the corresponding exceptions and will manage them.

This advantage is very interesting, everybody want an application which is robust. It demands a good way of programming because we had to manage some exceptions. Unfortunately we find that this advantage is not so obvious, we have met a lot of bugs in the development of our application. This disagreement will be explained in the next section¹⁴.

5.1.4 Disadvantages

Unfortunately, Java has also disadvantages. The biggest one is that Java has a lot of bugs. Sometimes it is possible to work with by doing differently than what was forecasted. But other times it is really impossible to work properly with these bugs. For example, it was impossible to remove an object from a container if the mouse was still on this object. With the WWW, it was possible to send our potential bugs to SUN, they sent back if it was really a bug and tips to pass through if it was possible. For example, it was impossible to print a string; after having sent a mail to SUN, the response was that this bug has already being reported and that it was possible to pass through this bug by setting the font every time before printing a string.

Another disadvantage is that there is no debugger environment for the JDK, it means that for debugging we had to write the content of the variables on the screen. It takes a lot of time, too much time.

¹⁴ Cf infra 5.1.4

5.2 VisualAge

VisualAge is quite a very good builder, except from the bugs. It manages the classes and the methods very properly and it is very easy to find what we are looking for. The debugger session is powerful and very easy to use. The creation of a STATIC user interface is going very fast, without major problems. The only disadvantage is that there are some interactive objects that you cannot create, for example a thumbnail groove or a predefined message box (warning box...). But the problem is that these objects do not exist as natural object in Java. However, we can imagine using external libraries (e.g. Tea Set Widgets).

VisualAge is a very good builder for the debugging. We could not use it at the end of our training period because there were too many bugs and some features were not implemented in version 1.0.

5.3 Remark

In conclusion, we have taken a lot of time to try to work with these bugs (first Java bugs and after VisualAge bugs). Especially at the beginning, we did not know that it was possible to have so many bugs in a compiler or in a commercial software, we thought it was our fault! We think that the reason of this is the youth of Java and naturally of VisualAge. With the further versions, it will much better and Java will really be the language of the future.

CHAPTER 6

Internal management

Resume:

In chapter 5, we have described the software environment used to implement the task model editor: Java and VisualAge.

This chapter is aimed at describing the system from the technical point of view whereas chapter 7 will take care of the graphical interface and the Human Computer Interaction.

We first introduce in section 6.1 the architectural analysis of our research. Then, in section 6.2, we describe the semantics of the language.

Section 6.3 shows the functional analysis. It describes the objects and their hierarchy.

In section 6.4, we develop the functional analysis and describe the internal structure.

In section 6.5, we mention the problem of memorizing a complete model by keeping in memory a minimum set of information (the root of the tree).

Section 6.6 explains the conversion of a file written in Rose Script into our internal structure.

Section 6.7 talks about the ability to modify graphically a model by the technical writers.

In section 6.8, we describe the user file: a 'work' file containing a final or non-final form of a particular model.

Finally, in section 6.9, we explain how to convert the internal structure into a LISP file, used later for language generation.

6.1 Architectural analysis (schema of the model)

Figure 6.1, describes the complete process of the task model editor from the acquisition of the Rose file to the generation of the LISP file.

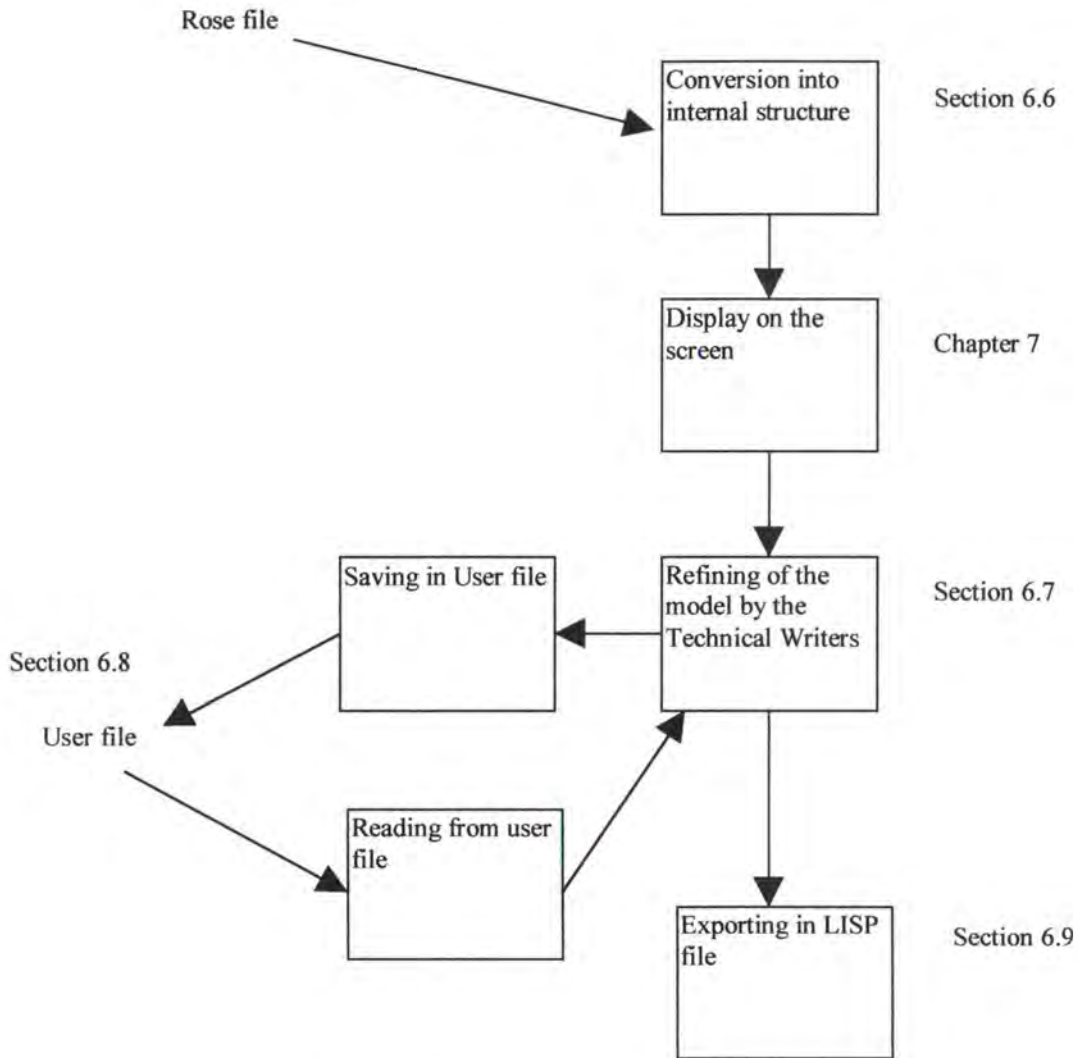


Figure 6. 1: Schema of the model

6.2 Semantics.

There are many ways to describe a model. The one chosen for the Isolde project is a Task model, as seen in chapter 5. For the internal representation, we use objects and their attributes. The internal representation is described by sequences of different objects: Tasks and Boolean connectors.

A Task can be **interactive**, **automatic** or **manual**. It is **mandatory** or **optional** and can have a **precondition** and/or a **feedback**. A task is either **terminal** or not. It can also have a description.

Some Tasks are executed more than once : that is why each Task has a connectivity field that contains the minimum and the maximum of times it will be executed. A Task can then have the connectivity (0-n) or (1-5) or (1-1), etc.

A Boolean Connector can be OR, AND, XOR or have no name.

A Sequence is a link between a Task and a Boolean Connector or a link between two Tasks. A sequence can have a sequence pre-condition, which is a condition for using the sequence that is considered.

In Figure 6.2, we can consider a simple model where each task is elementary.

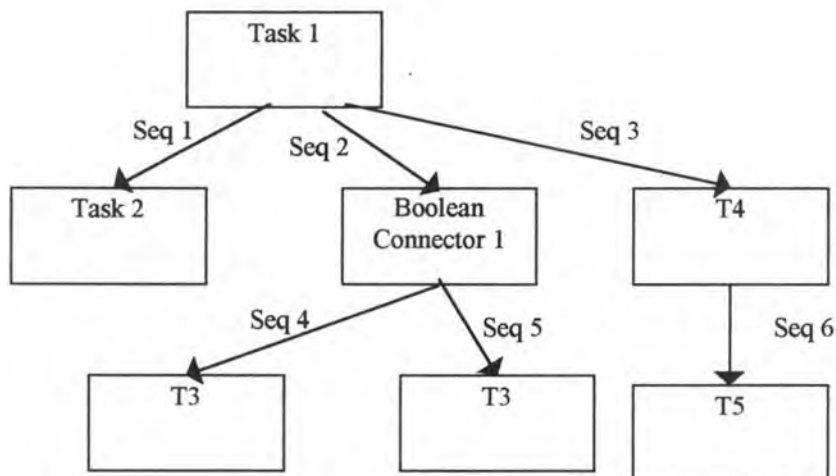


Figure 6. 2: A simple model with elementary Tasks

Another feature of the language is that a Task can be described as a sequence of sub-tasks. Each of the sub-tasks can also be described as a sequence of sub-sub-tasks, etc.

This means that, in order to describe that in our internal representation, we start from a very high level (very complex task) and refine each task again and again until we reach a very low level task analysis. In the next example, we can see the decomposition of a complex task in elementary tasks using successive decompositions. This is illustrated in Figure 6.3.

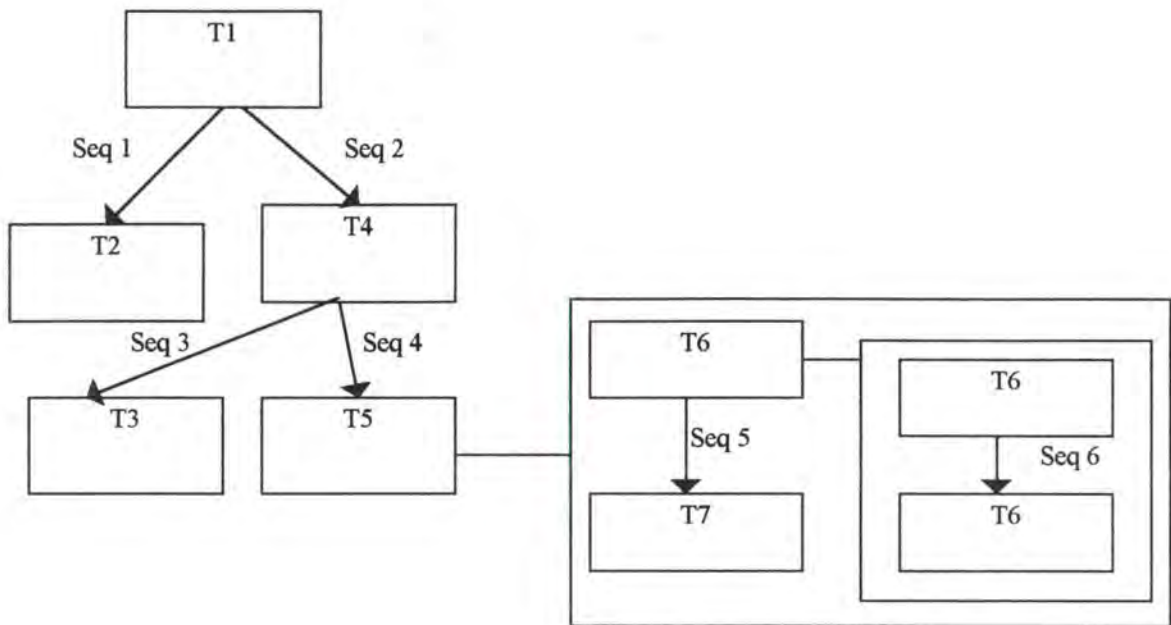


Figure 6. 3: A model with composite Tasks

6.3 Functional analysis (OO approach : description of the objets and the hierarchy)

Java is a object-oriented language. The first step was to find a correct hierarchy for the different objects, in order to be able to describe the model to use the potential of an object-oriented language.

We use the following decomposition and consider a primitive class: a **Widget**.

A **Widget** is a simple object identified by its position on the screen (X and Y integer coordinates).

It is also characterized by I children ($I \geq 0$), J parents ($J \geq 0$) and an expansion box (that can be either null, not null and empty, not null and not empty).

Let's now define two sub-classes that have all the characteristics of a Widget plus some more: The **Sequence**, the **Task** and the **Boolean connector**.

A **Sequence** is a Widget with a pre-condition and a Boolean flag (= TRUE if the sequence has a precondition).

A **Task** is a Widget with some more characteristics specific of the task: It has a task name, a description, a feedback, a precondition, a style, a choice, a complexity, flags, and a connectivity (min, max).

A **Boolean Connector** is a **Widget** that has also a Boolean type (OR, AND, XOR, none). We can see in Figure 6.4 the relation between the different objects.

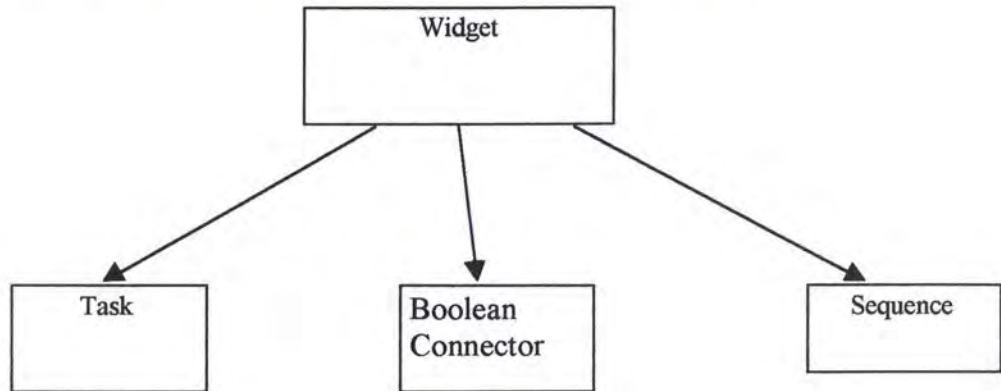


Figure 6. 4: The hierarchy of the objects

We can represent of a model in a graph. In order to represent the links between every **Widget**, we use a Java class that implements the linked list feature: the `Java.Util.Vector` class.

In Figure 6.5, we can see how the information of a model is represented in the internal structure.

The \downarrow arrow represents a sequence of objects.

The \rightarrow arrow represents a number of objects linked to the same element.

The \Rightarrow arrow represents the 'how to' relation, i.e. an expansion box.

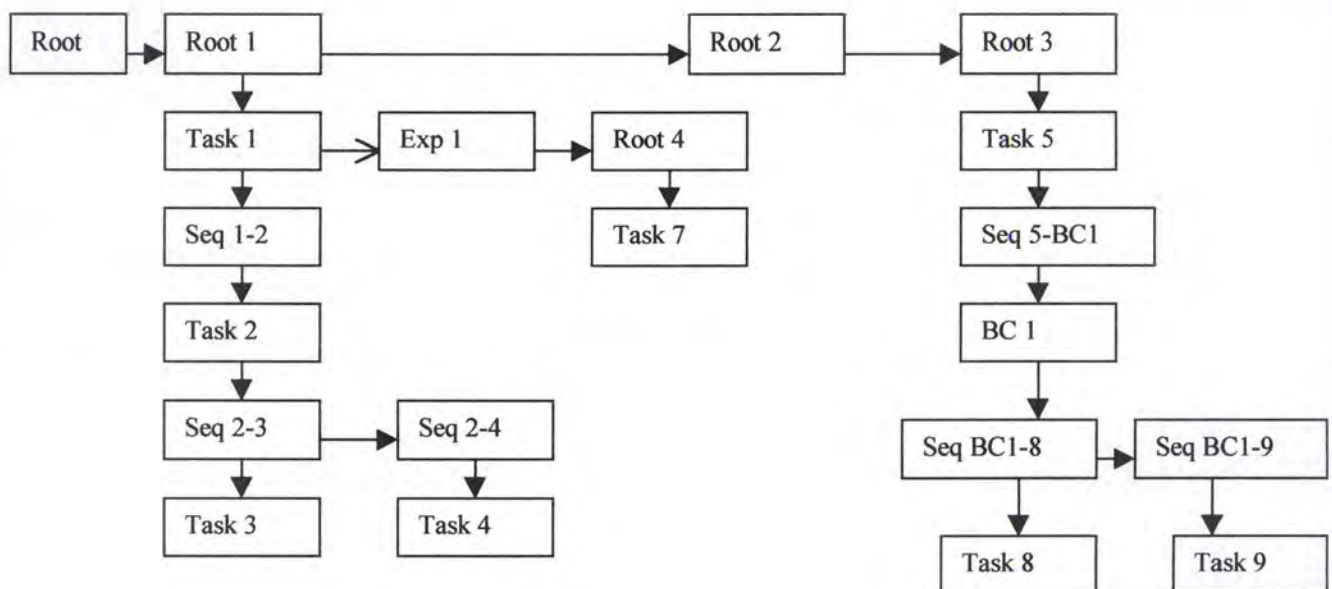


Figure 6. 5: The internal representation

Let us remark that, in order to memorize the complete graph, we must only memorize the first root *Vector* and all its connections. From there, knowing the connections of the different objects we know, we can induce all the other objects. In the next section, we will see how to manage the root.

6.4 The root and its properties

In order to manipulate a graph, it can be useful to consider a limited number of nodes that allows us to access all the other nodes, using the successors of the known nodes.

First of all, we will consider an oriented graph G , composed of a set of nodes S and a set of links $A(A \subseteq S \times S)$. $G = (S, A)$. We assume we have an anti-reflexive relation.

If we consider a node s , its successors are nodes s' such as there exists a link $(s, s') \in A$. A node s_n is accessible from another node s_0 if there exists a set of links $((s_0, s_1), (s_1, s_2), \dots, (s_{k-1}, s_k), (s_k, s_n))$

Let us consider the function $\text{succ} : s \times G \rightarrow S$ that associates to a node s and a graph G the set of nodes that are accessible by one transition in graph G , i.e. using only one link.

$$\text{succ}(s, G) = \{s' \mid (s, s') \in A\}$$

As for succ , $\text{succ}^* : s \times G \rightarrow S$ associates to a node s and a graph G the set of nodes accessible from s .

$$\text{succ}^*(s, G) = \{s' \mid \exists s_0, s_1, \dots, s_n \in S : s_0 = s \wedge \forall i : 0 \leq i < n, (s_i, s_{i+1}) \in A \wedge s_n = s'\}$$

Let us also define the ancêtres function. $\text{ancêtres} : s \times G \rightarrow S$ associates to a node s and a graph G all the nodes s_i ($i=1, \dots, n$) from where we can access s

$$\text{ancêtres}(s, G) = \{s' \mid \exists s_0, s_1, \dots, s_n \in S : s_0 = s' \wedge \forall i : 0 < i \leq n, (s_i, s_{i-1}) \in A \wedge s_n = s\}$$

Now is the time to define the $R \subseteq S$. R is the minimal cardinality subset such as every node $s \in S$ is accessible from an element of R . Formally,

$R =$ set of nodes :

$$\triangleright \forall s \in S : \exists r_1 \in R : \text{succ}^*(r_1, G) \ni s$$

$$\triangleright \neg \exists R_2 : (\#R_2 < \#R) \wedge \forall s \in S : \exists r_2 \in R_2 : \text{succ}^*(r_2, G) \ni s$$

In a non cyclic graph, we can prove that R is unique [Anciaux 98].

6.5 Description of the procedures

6.5.1 Class IO

This class has one purpose: taking care of the inputs and outputs.

Method **imp**

Parameters:

this: not used here

{The IO is not really an object from the data structure point of view}

dirName, fileName: String

{In order to use the imp method, we need to know which file to look in, in which directory.}

The method:

This method reads a 'Rose' file (a file containing no coordinates).

In this file, the name of a Widget is its identifier.

The Boolean Connectors, however, are identified by an integer I (I>999999).

After reading the file, the structure contains all the elements but no adequate coordinates. The task coordinates have the -1 value, and the sequence coordinates have the 0 value.

The next step is to add a weight to every Widget in the structure by calling the addweight method.

When every Widget has a weight, we can call the addcoord method to add coordinates to every Task and Boolean Connector.

The final step is to call the addcoordseq method to add coordinates to the Sequences.

Method **readfile**

Parameters:

this: not used here

dirName, fileName: String

{In order to use the readfile method, we need to know which file to look in, in which directory.}

The method:

This method reads a 'user' file (a file containing coordinates)

Remember that every task that has the same name has the same expansion box.

Method **savefile**

Parameters:

this: not used here

dirName, fileName: String

{In order to use the readfile method, we need to know which file to look in, in which directory.}

The method:

This method saves a 'user' file (a file containing coordinates)

It actually opens the file, calls the Widget.wsave method, and closes the file.

Method export*Parameters:*

this: not used here

dirName, fileName: String

{In order to use the readfile method, we need to know which file to look in, in which directory.}

The method:

This method exports a file in a 'LISP' format

It actually opens the file, calls the Widget.wexport method, and closes the file.

6.5.2 Class Widget**Method wsave***Parameters:*

this: the root

{We always use the 'top' root as a calling parameter for the wsave method}

dataOutputStream: DataOutputStream

{Class used for files}

dirName: String,

fileName: String

{Directory name and file name}

ss int

{The number of times a file has been saved, or exported, since last new, open or import.}

The method:

This recursive method goes once to every Widget in the structure, increments its *saven* variable then calls the corresponding method, for every Widget:

Task.save

BooleanConnector.save

Sequence.save

Widget.saveexp

Method wexport

Parameters:

this: the root

dataOutputStream: DataOutputStream
{Class used for files}

dirName: String,
fileName: String
{Directory name and file name}

ss int
{Parameter containing the number of times a file has been saved.}

The method:

This recursive method goes once to every Widget in the structure, increments `saven` then redirects to every particular format, for every Widget except from the Sequence:

Task.export

BooleanConnector.export

Widget.exportexp

Method saveexp

Parameters:

this: the task that is expanded

{In order to write the information about an expansion box, we need to know from which task it is expanded}

dataOutputStream: DataOutputStream
{Class used for files}

dirName, fileName: String
{In order to use the saveexp method, we need to know in which file to write, in which directory.}

The method:

This method writes to a user file the information about an expansion box, that is

- The name of the expanded task
- The first nodes of the expansion box
- The coordinates of the expansion box

Method addweight

Parameters:

this: the root

flag: boolean

{This recursive method must be first called with flag initialized to true, in order to reinitialize the check vector, then recursively called with flag initialized to false}

The method:

This method takes a complete structure from a Rose file, but with no coordinates or weight in it, and returns the same structure with still no coordinates but with a weight for every Task and Boolean Connector.

The purpose of this method is to add a coordinate to every Task and Boolean Connector. The way to do it is to consider that the more parents and children a Task (or BC) has, the bigger weight it has.

The weight of a Widget that is not in the root is (# of parents + # of children)
As a Widget that is in the root intuitively needs more 'space', the weight of a Widget which is in the root is $(11 + 10 * (\# \text{ of children}))$

This definition is, of course, completely arbitrary, but seems to work quite well.

Method addcoord

Parameters:

this: the root

flag: boolean

{This recursive method must be first called with the flag initialized to true, in order to reinitialize the check vector, then recursively called with flag initialized to false}

startx: int

{This variable tells the method at which x coordinate the next Widget can be drawn}

starty: int

{This variable tells the method at which y coordinate the next Widget can be drawn}

spacex: int

{This variable tells the method the x space available for the next Widget}

The method:

This recursive method takes a complete structure from a Rose file, with no coordinates. Every Task and Boolean Connector have just a weight. The method returns the same structure with coordinates for every Task and Boolean Connector.

To decide which coordinate to give to every object, we proceed like this:

There is a ratio variable: k that is initialized at 1.8 but can be increased if we want more space between the Widgets, decreased if we want less space between them.

The y coordinates are incremented by 90 for each level

For the x coordinates, according to the weight of every Widget, and the number of brothers it has, a 'vital space' is proportionally given to every Widget.

Because this method is only called by `IO.import`, the only type of file we will deal with are simple Rose files. That means that, in order to compute the coordinates of a task in an expansion box, we just need to draw them on top of each others, without using any kind of weight.

So far, the implementation suits simple models but does not support models with composite tasks.

The approach we have used is very intuitive and simple, but more elaborate work in this field can be seen in [Henry 91] and [Ryall 97].

Method `addcoordseq`

Parameters:

`this`: the root

`flag`: boolean

{This recursive method must be first called with `flag` initialized to true, in order to reinitialize the check vector, then recursively called with `flag` initialized to false}

The method:

This recursive method takes a structure with coordinates only for Tasks and Boolean Connectors. It goes to every Sequence, finds its coordinates according to which *parent* and *child* it has, and returns a structure with coordinates assigned to every object.

Method `find`

Parameters:

`this`

{This parameter is the Widget we start from when performing a search}

`name`: int

The id of the Task or Boolean Connector we are looking for

`found`: boolean

{This recursive method must be first called with `found` initialized to true, in order to reinitialize the check vector, then recursively called with `found` initialized to false}

The method:

This recursive method finds an id and returns the object (BC or Task) that is identified by that id. The boolean variable 'found' returns true if the objet was found, false otherwise. The method must be first called with `found` initialized to true, then recursively with `found` initialized to false, in order to re-initialize the check counter before each search.

Remark that this method does not look in the expansion boxes.

Method find2

This method works exactly the same way as find, except that it takes a task name (a String) as a parameter, instead of a task id.

It is used with the Rose file, where the task name is identifier.

Method superfind

Parameters:

this

{This parameter can be any Widget we want to start from when performing a search}

name: Task

The Task that has a name we are looking for

found: boolean

{This recursive method must be first called with found initialized to true, in order to reinitialize the check vector, then recursively called with found initialized to false}

The method:

This method looks for a task. It returns a Widget if a task with the same name as 'name' is found, null otherwise.

The method has to be called with the boolean variable 'found' initialized to 'true' in order to clear the check list (to be able to deal with nested tasks)

Remark that this method looks in the expansion boxes as well.

Method superexpanded

Parameters:

Idem superfind

The method:

This method looks for a task name. It returns a Widget if a task with the same name as 'name' is found **and expanded**, null otherwise. The method has to be called with the boolean variable 'found' initialized to 'true' in order to clear the check list (to be able to deal with nested tasks)

Method superfindall

Parameters:

this

{This parameter can be any Widget we want to start from when performing a search}

name: String
 {The name of the Task we are looking for}

found: boolean
 {This recursive method must be first called with found initialized to true, in order to reinitialize the check vector, then recursively called with found initialized to false}

exp: Widget
 {The expansion box we want to add to every task that has the same name as 'name'}

The method:

This method adds an expansion box exp to every task that has the task name 'name'

Method superremove

Parameters:

Idem superfind

The method:

This method looks for a task name.
 It returns a Widget if the task is found, null otherwise.
 The method has to be called with the boolean variable 'found' initialized to 'true' in order to clear the check list (to be able to deal with nested tasks)

Method closew

Parameters:

expbox: Widget
 found: boolean

The method:

This method goes to every Widget and closes all the expansion boxes

Method wcons

Parameters: none

The method:

This is the constructor of a Widget. It initializes *saven* to *savenum*. The method is used when the user creates himself an expansion box, or performs the 'new' command.

Method wcons2

Parameters: none

The method:

This is the constructor of a Widget. It initializes *saven* to 0. The method is used when opening or importing a file.

Method makeexp

Parameters:

expbox: Widget
found: boolean

The method:

This method draws all the descendants of a root in the same expansion box

Method makeAll

Parameters:

Idem makeexp

The method:

This method goes to every Widget and draws it

6.5.3 Class Sequence

Method cons

Parameters:

this: the sequence we want to create
x1,y1,x2,y2: int
{Coordinates of the sequence we want to create}

pre: String
{Precondition of the sequence we want to create}

The method:

Taking all the parameters of a Sequence, it returns a Sequence fully loaded
This method is used when the user wants to create a Sequence

Method consopen

Parameters:

this: the sequence we want to create
 x1,y1,x2,y2: int
 {Coordinates of the sequence we want to create}

pre: String
 {Precondition of the sequence we want to create}

The method:

Taking all the parameters of a Sequence, it returns a Sequence fully loaded
 This method is used when the user opens a file

Method create*Parameters:*

this: the sequence we want to create
 x1,y1,x2,y2: int
 {Coordinates of the sequence we want to create}

w1, w2: Widget
 {The widgets we want to link}

w: Widget
 {The root: we use it to detect the nested graphs by calling the find method}

The method:

This method creates a new sequence between w1 and w2, i.e. w1 has a new child (the sequence), the sequence has a new child (w2). The method also expands the 'root' of the tree as mentioned in section 6.4

Method createopen*Parameters:*

this: the sequence we want to create
 x1,y1,x2,y2: int
 {Coordinates of the sequence we want to create}

w1, w2: Widget
 {The widgets we want to link}
 pre: String
 {Precondition of the sequence}

The method:

Creates a new sequence between w1 and w2 when the user opens a file

Method createRose

Parameters:

Idem createopen

The method:

Creates a new sequence between w1 and w2 when the user imports a Rose file

Method delete

Parameters:

this: the sequence we want to delete

rr: Widget
{The root}

The method:

This method deletes a sequence from a model, and takes care of removing or adding elements to the root if needed

Method save

this: the sequence we want to save

dataOutputStream: DataOutputStream
{Class used for files}

dirName, fileName: String

{In order to use the saveexp method, we need to know in which file to write, in which directory.}

The method:

This method writes to a user file the information about a sequence, that is

- The id of the 'parent' Widget
- The id of the 'child' Widget
- The precondition, blank line otherwise
- The coordinates of the sequence, negative if in an expansion box that is not expanded

6.5.4 Class Task

Method cons

Parameters:

this: the Task we want to create

x1,y1,x2,y2: int
{Coordinates of the Task we want to create}

taskname: String
 {Name of the task: not identifier of the Task}

description: String
 {Description of the Task}

precondition: String
 {Precondition of the Task}

st: String
 {Style of the Task}

chce: String
 {Choice of the Task}

comp: String
 {Complexity of the Task}

tpre, feed, tevent: boolean
 {Flags, respectively for precondition, feedback and terminal event}

feedback: String
 {Feedback of a Task}

min, max: String
 {Connectivity of the Task}

rr: Widget
 {The root we want to add the Task to}

The method:

Taking all the parameters of a Task, it returns a Task fully loaded.
 This method is used when the user wants to create a Task.
saven is initialized to *savenum*.

Method consimport

Parameters:

Idem cons except that here, there is no min or max parameter. The default parameter for a Task is (1-n)

The method:

Taking all the parameters of a Task, it returns a Task fully loaded.
 This method is used when the user imports a Rose file.

Method consopen

Parameters:

Idem cons

The method:

Taking all the parameters of a Task, it returns a Task fully loaded.
This method is used when the user opens a file.
saven is initialized to 0.

Method create

Parameters:

Idem cons

The method:

This method creates a new Task, i.e. adds it to the root. We must be aware that for adding a task to an expansion box, the root we use is the expansion box itself.

Method createimport

Same as create except that this method is used for importing a Rose file

Method createopen

Same as create except that this method is used for opening a user file

Method delete

Parameters:

this: the Task we want to delete
rr: Widget
{The root}

The method:

This method deletes a Task from a model, and takes care of removing or adding elements to the root if needed. It can handle any nested loop between the different elements.

Method export

this: the Task we want to export
dataOutputStream: DataOutputStream
{Class used for files}

dirName, fileName: String
{In order to use the saveexp method, we need to know in which file to write, in which directory.}

The method:

This method exports the different parameters of a Task, which are:

- The Task id
- The id of its expansion box
- The semantics of the Task
- The different links (children)

This is also the place where we check whether a Task is a Terminal Event and we display the information if needed.

Method save

this: the Task we want to save
 dataOutputStream: DataOutputStream
 {Class used for files}

dirName, fileName: String
 {In order to use the saveexp method, we need to know in which file to write, in which directory.}

The method:

This method writes to a user file the information about a Task, that is

- The id of the Task
- The task name
- The minimum
- The maximum
- Yes if it is a Terminal Event, No otherwise
- The coordinates of the Task
- The task precondition, blank line otherwise
- The feedback, blank line otherwise
- The choice
- The style
- The complexity
- The description
- A blank line

6.5.5 Class Boolean Connector

Method cons

Parameters:

this: the Boolean Connector we want to create
 x1,y1,x2,y2: int
 {Coordinates of the Boolean Connector we want to create}

boolname: String

{Name of the Boolean Connector we want to create}

rr: Widget

{The root where to add the Boolean Connector}

The method:

Taking all the parameters of a Boolean Connector, it returns a Boolean Connector fully loaded.

This method is used when the user wants to create a Boolean Connector

Method consimport

Parameters:

Idem cons except that here, there is no need for a root as a parameter: we always create a new root in an import operation.

The method:

Taking all the parameters of a Task, it returns a Task fully loaded.

This method is used when the user imports a Rose file.

Method consopen

Idem consimport except that here, there is one more parameter: the Boolean Connector ID.

This method is used when the user opens a user file.

Method create

Parameters:

Same parameters as cons.

The method:

This method creates a new Boolean Connector, i.e. adds it to the root. We must be aware that for adding a Boolean Connector to an expansion box, the root we use is the expansion box itself.

Method createimport

Same as create, but for importing Rose files only.

Method createopen

Same as create, but for opening users files only.

Same parameters as consopen.

Method delete

Parameters:

this: the Boolean Connector we want to delete
 rr: Widget
 {The root}

The method:

This method deletes a Boolean Connector from a model, and takes care of removing or adding elements to the root if needed.

Method export

this: the Boolean Connector we want to export
 dataOutputStream: DataOutputStream
 {Class used for files}

dirName, fileName: String
 {In order to use the saveexp method, we need to know in which file to write, in which directory.}

The method:

This method exports the different parameters of a Boolean Connector, which are:
 -The Boolean Connector id
 -The type of Boolean Connector
 -The different links (children)

Method save

this: the Boolean Connector we want to save
 dataOutputStream: DataOutputStream
 {Class used for files}

dirName, fileName: String
 {In order to use the saveexp method, we need to know in which file to write, in which directory.}

rr: Widget
 {The root}

The method:

This method writes to a user file the information about a Boolean Connector, that is
 -The id of the Boolean Connector
 -The Boolean name
 -The coordinates of the Boolean Connector

6.6 How to convert files from Rose into the internal structure

6.6.1 Description of the file generated by Rationale Rose

The file contains some information about the task model. The information describes the relation between an object and another object.

The file also contains some information about the domain model, which is described in chapter 5.

6.6.2 The reading of the file and the transformation into internal structure

We first fill our structure with all the elements we read in the Rose file. To do so, we must consider the different objects present in the Rose file:

- The Tasks
- The Boolean Connectors
- The Sequences
- The expansion boxes

We read the file in three passes: The first pass only takes care of the Tasks and Boolean Connectors. After this is done, we have a flat basic structure.

In the second pass, we read the Sequences, and make the connections between Tasks and Boolean Connectors. After this is done, we have a sequence of objects, but every object is elementary (not expressed as a sub-sequence of objects).

The final stage is to read the expansion boxes, so we can deal with complex hierarchy of tasks as well. The coordinates are lacking.

The import command reads a Rose file, converts it to an internal structure which has to be visualized on the screen.

A major problem is to convert a file structure with no coordinates to an internal structure with coordinates and a visual representation.

In Figure 6.6, we see the conversion from a Rose file into internal representation with no coordinates.

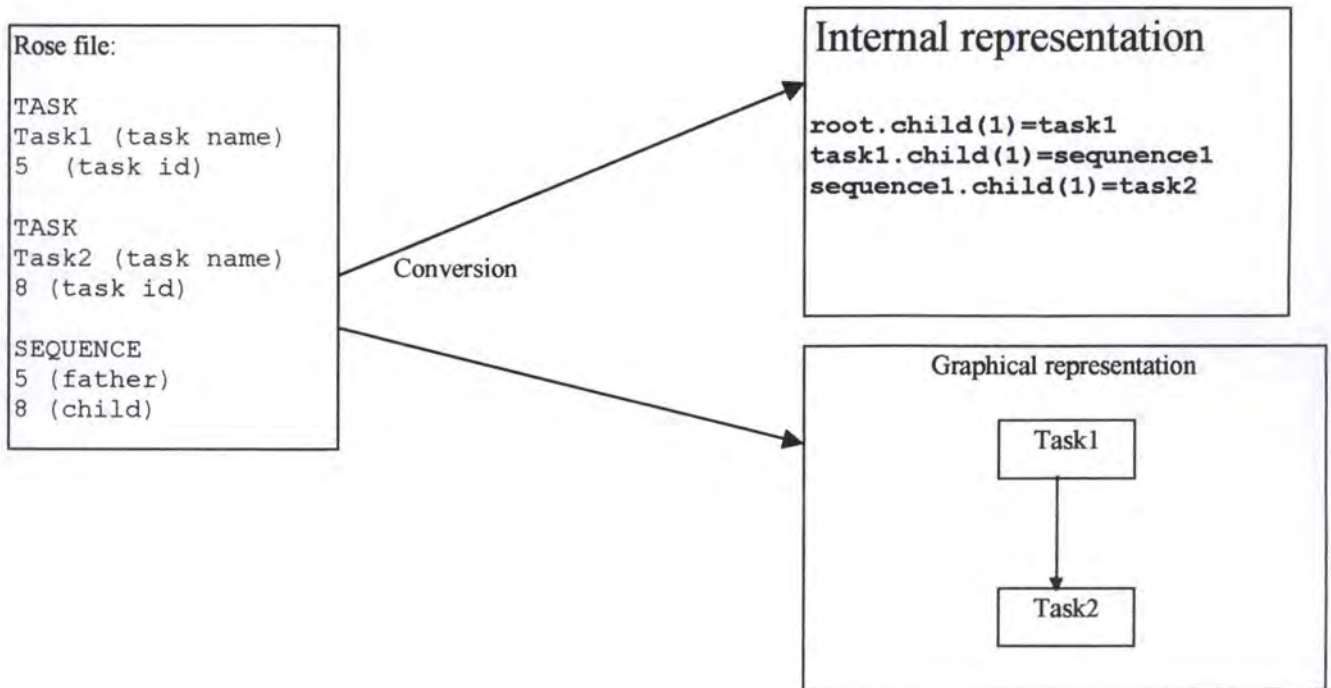


Figure 6. 6: The conversion into internal structure with no coordinates

6.6.3 The coordinates problem

The first step is to return the same structure with still no coordinates but with a weight for every Task and Boolean Connector. The way to do it is to consider that the more parents and children a Task (or BC) has, the bigger weight it has.

Practically, we should go through the structure, examine every Task and Boolean Connector, and assign a weight to each of them. If the object we consider is in the root, its weight will be higher (for readability reasons).

In Figure 6.7, we see the addition of a *weight* parameter for every Task and Boolean Connector.

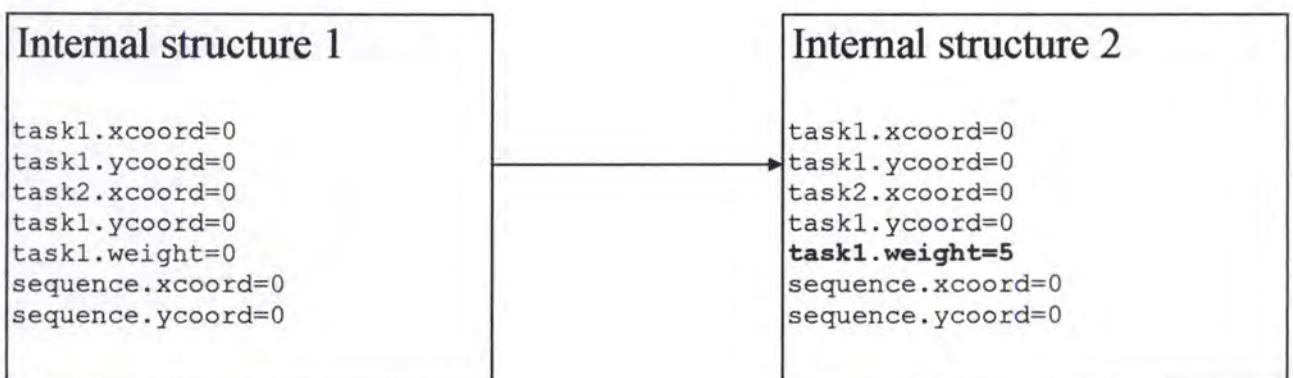


Figure 6. 7: Addition of a *weight* parameter

Going through the structure again, we can add coordinates to every task and every Boolean connector. We use three main parameters:

- The vital space for x coordinates
- The vital space for y coordinates

In Figure 6.8, we add coordinates to every Task and Boolean Connector

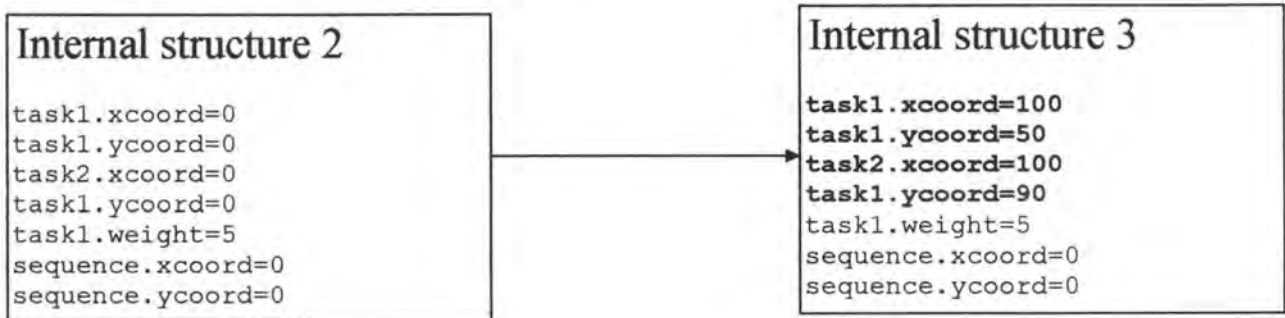


Figure 6.8: Addition of coordinates

The next step is to go through the structure again to add coordinates to the sequences, knowing the coordinates of the father and son of each sequence (Figure 6.9).

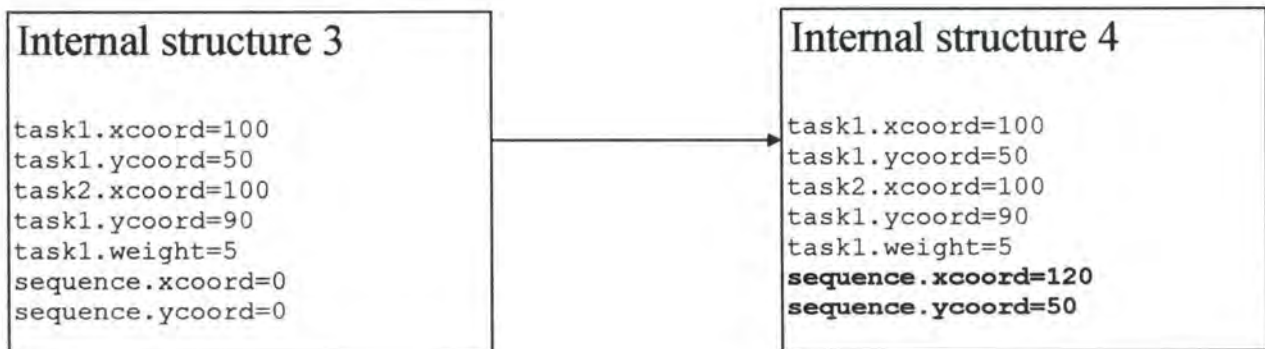


Figure 6.9: Addition of coordinates to the sequences

Now that the structure is complete and that we know the coordinates of each object, we can go through the structure one more time and draw every object on the screen.

6.7 Modifications of the structure by the technical writers

Now that we have a primal sketch of a task model, we want to allow the user (the technical writer) to modify the task model, by adding, deleting, moving the different objects, and decompose any task in sub-tasks.

When the user performs an operation, the graph must be updated. Besides creating or deleting a particular object, we must take care of having at all times, a consistent set of roots. That's why, after some operations, the root must be updated (ANCI AUX, 1998)

6.8 Saving and re-opening a particular model

6.8.1 Second type of file (with coordinates, etc.) : the user file

The technical writers can save the model into a user file and re-open it exactly the way it was before. Let's mention that whereas the Rose file doesn't have any coordinates, the user file must provide coordinates.

The user file is very similar to the Rose file, except that it provides coordinates for every element (Task, Boolean Connector, Sequence and Expansion Box).

Tasks

These are the information written in the file:

- The id of the Task
- The task name
- The minimum
- The maximum
- Yes if it is a Terminal Event, No otherwise
- The coordinates of the Task
- The task precondition, blank line otherwise
- The feedback, blank line otherwise
- The choice
- The style
- The complexity
- The description
- A blank line

Boolean Connectors

These are the information written in the file:

- The id of the Boolean Connector
- The Boolean name
- The coordinates of the Boolean Connector

6.9 How to convert the internal structure into a predefined (LISP) format

6.9.1. Third type of file: the LISP (or export) file (with no coordinates, etc.)

When the Technical Writer is satisfied with its final task model, he can export it to a LISP file.

We must dissociate the different objects to export:

Tasks

The different parameters to be exported are:

- The Task id
- The id of its expansion box
- The semantics of the Task
- The different links (children)

If a Task is a Terminal Event , we display the information as well

Boolean Connectors

The different parameters to be exported are:

- The Boolean Connector id
- The type of Boolean Connector
- The different links (children)

Sequences

The different parameters to be exported are:

- The id of the 'parent' Widget
- The id of the 'child' Widget
- The precondition, blank line otherwise
- The coordinates of the sequence, negative if in an expansion box that is not expanded

Expansion Boxes

The different parameters to be exported are:

- The name of the expanded task
- The first nodes of the expansion box
- The coordinates of the expansion box

CHAPTER 7

The Graphical User Interface

Resume:

In chapter 6, we have covered the technical analysis of the editor. This chapter is interface-oriented and describes the graphical part of the task-modeling tool.

The next chapter will contain the last part of the Isolde project : the language generation itself.

After a short introduction (section 7.1), we illustrate the different functions of the Graphical User Interface (GUI) in section 7.2.

Section 7.3 suggests some improvements for the interface.

7.1 Introduction

The user interface of our task modeling tool was designed based on a user requirement analysis [Balbo 97].

It is important to notice that only the functions currently implemented will be presented here. In the further versions of the application, the others functions will be operational.

7.2 The functions of the interface

In the next pages, when we will be spoken about widgets, we will mean boolean connector, sequence and task.

7.2.1 Buttons and checkboxes of the GUI

- Clicking on the "Select" button puts us on the "Select" mode.
- Clicking on the "Create Task" button puts us on the "Create Task" mode.
- Clicking on the "Create Boolean Connector" button puts us on the "Create Boolean Connector" mode.
- Clicking on the "Sequence tasks" button puts us on the "Sequence tasks" mode.
- If we are in the "Create Task" mode, we can choose to create:
 - An "Interactive", an "Automatic" or a "Manual" task;
 - A "Mandatory" or an "Optional" task;
 - An "Expanded" or an "Elementary" task

With:

- A "Task precondition";
- A "Feedback";
- A "Terminal event".

By clicking where we want to put it after have clicked on the corresponding checkboxes (choice of the kind of task we want).

- If we are in the "Sequence tasks" mode, we can create a new sequence by pressing the left button of the mouse on the origin of the sequence (boolean connector or task) and release it on the extreme of the sequence (boolean connector or task). It is impossible to create a sequence between 2 boolean connectors, between 2 times the same widget (task or boolean connector) or between 2 widget (task or boolean connector) from different levels (for

example an expansion box and another expansion box). It is possible to add a sequence precondition to this sequence by clicking on the "Sequence precondition" checkbox before the actual creation.

- If we are in the "Create Boolean Connector" mode, we can create a new boolean connector by clicking where we want to put it.

Figure 7.1 shows the buttons and the checkboxes of the GUI.

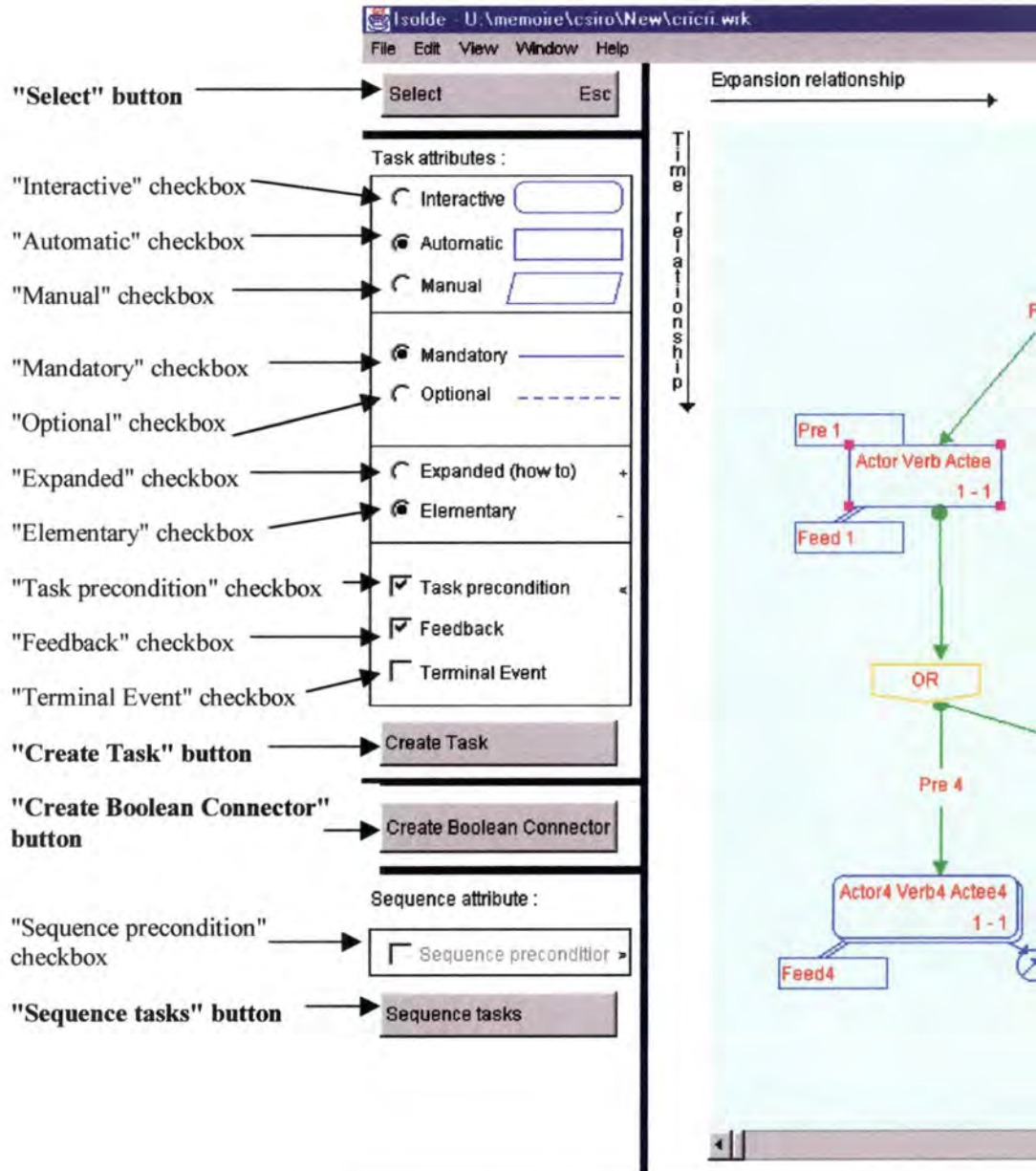


Figure 7. 1: Buttons and checkboxes of the Graphical User Interface

7.2.2 The menu item "File"

- It is possible to make a new task model by clicking on the "New" menu item (File -> New).
- It is possible to open a file previously saved from the Graphical User Interface (GUI), by clicking on the "Open" menu item (File -> Open).
- It is possible to open a file generated from Rose by clicking on the "Import" menu item (File -> Import).
- It is possible to save as a GUI format by clicking on the "Save" menu item (File -> Save).
- It is possible to export toward the language generation (LISP) by clicking on the "Export" menu item (File -> Export).
- It is possible to print the task model by clicking on the "Print" menu item (File -> Print).
- It is possible to quit the application by clicking on the "Quit" menu item (File -> Quit).

Figure 7.2 shows the menu item "File".

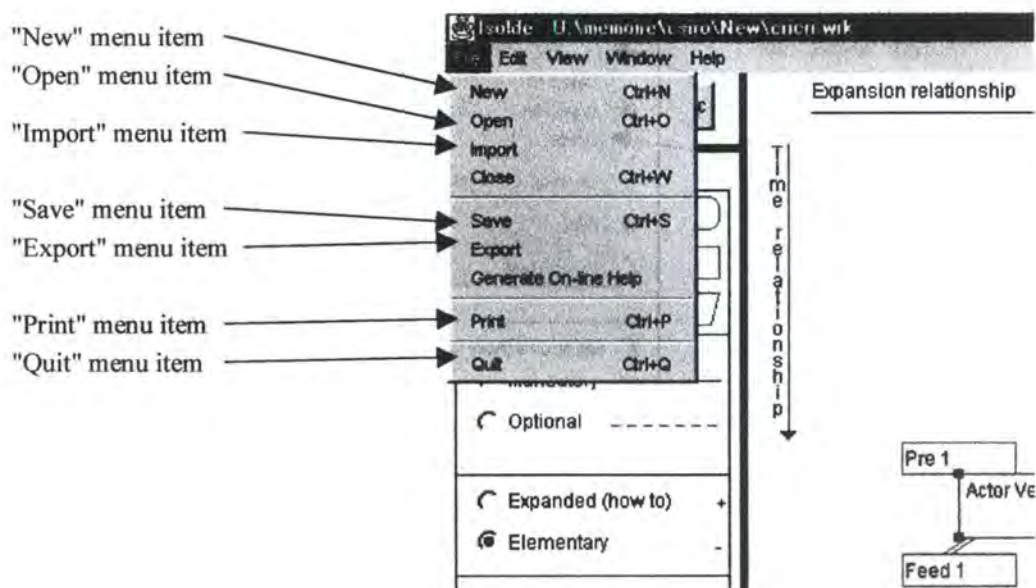


Figure 7. 2: The menu item "File"

7.2.3 The menu item "Edit"

- If a boolean connector or a task is selected, it is possible to cut this selected widget (copy it to the clipboard and delete it from the task model) by clicking on the "Cut" menu item (Edit -> Cut).
- If a boolean connector or a task is selected, it is possible to copy this selected widget (copy it to the clipboard) by clicking on the "Copy" menu item (Edit -> Copy).
- If a boolean connector or a task has been copied to the clipboard (copy or cut), it is possible to paste it by clicking where we want to put it after have clicked on the "Paste" menu item (Edit -> Paste).
- When a widget is selected, the deleting of this last is made by pressing the "Delete" key or clicking on the "Delete" menu item (Edit -> Delete). If this widget was a boolean connector or a task which had got links (sequences), the links are deleted at the same time.

Figure 7.3 shows the menu item "Edit".

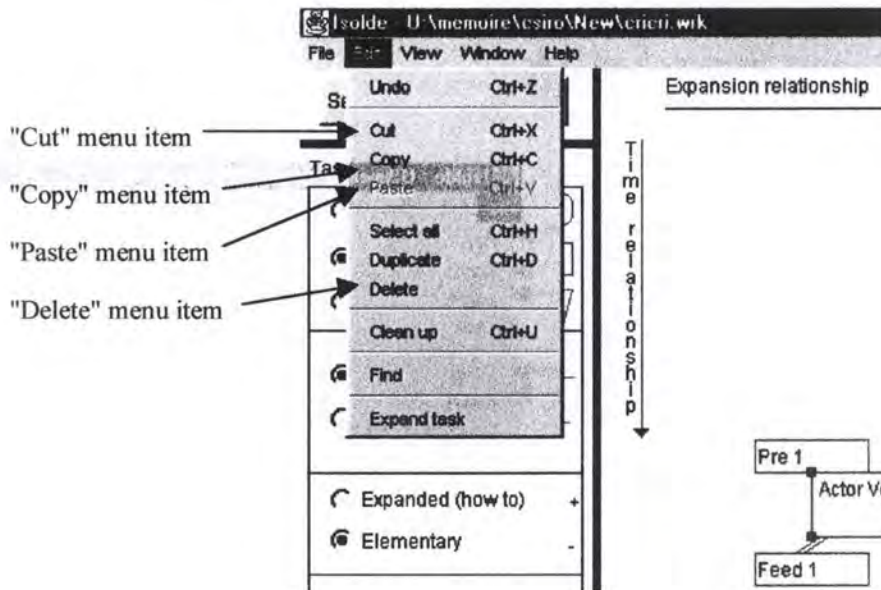


Figure 7.3: The menu item "Edit"

7.2.4 The menu item "View"

- For the legibility of the task model, it is possible to show or to hide some attributes:

- Task preconditions: clicking on the menu item (View -> Task preconditions).
- Sequence preconditions: clicking on the menu item (View -> Sequence preconditions).
- Feedbacks: clicking on the menu item (View -> Feedbacks).
- Constraints: clicking on the menu item (View -> Constraints).

Figure 7.4 shows the menu item "View".

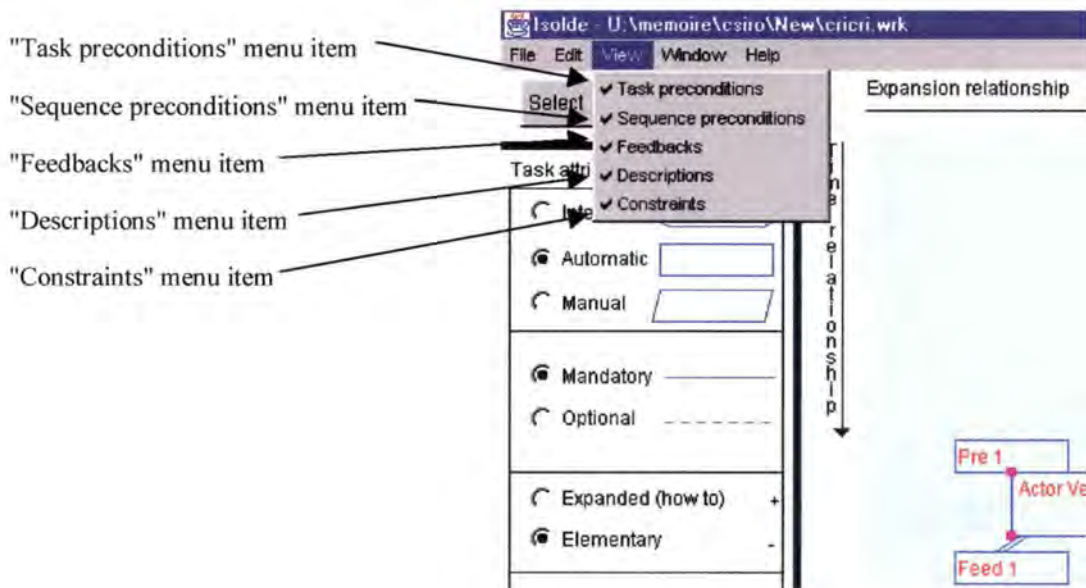


Figure 7. 4: The menu item "View"

7.2.5 The expansion box

- It is possible to expand the expandable tasks by double clicking on this last one and choosing the "Expand" button on the dialogue appearing. To close this expansion box, just click on the top right button of the expansion box.

Figure 7.5 shows how to expand an expandable task and Figure 7.6 shows how to close an expansion box.

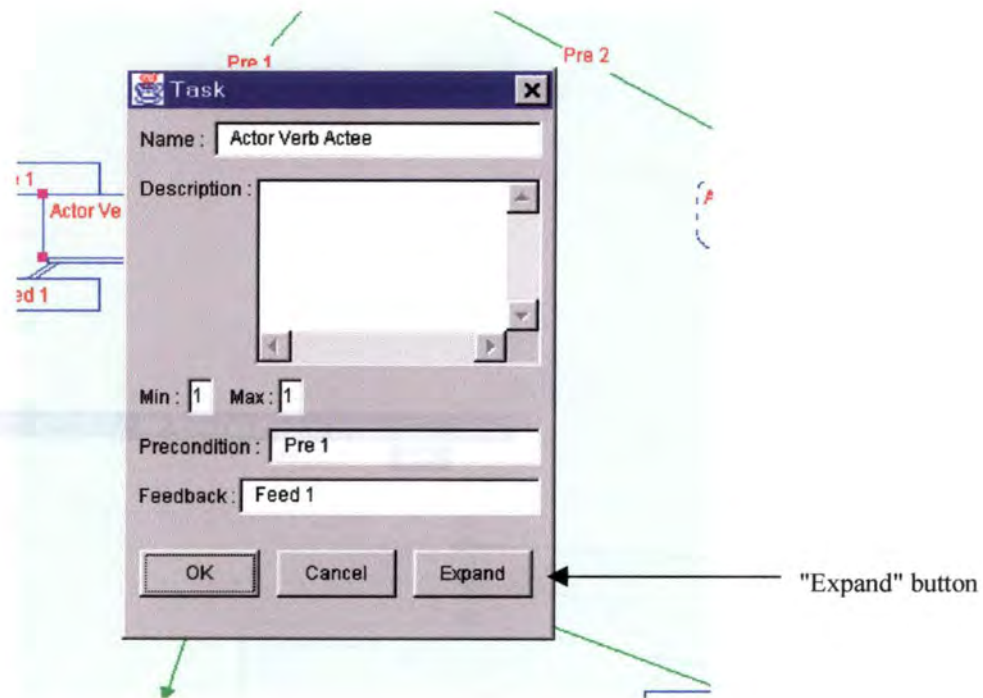


Figure 7.5: How to expand an expandable task?

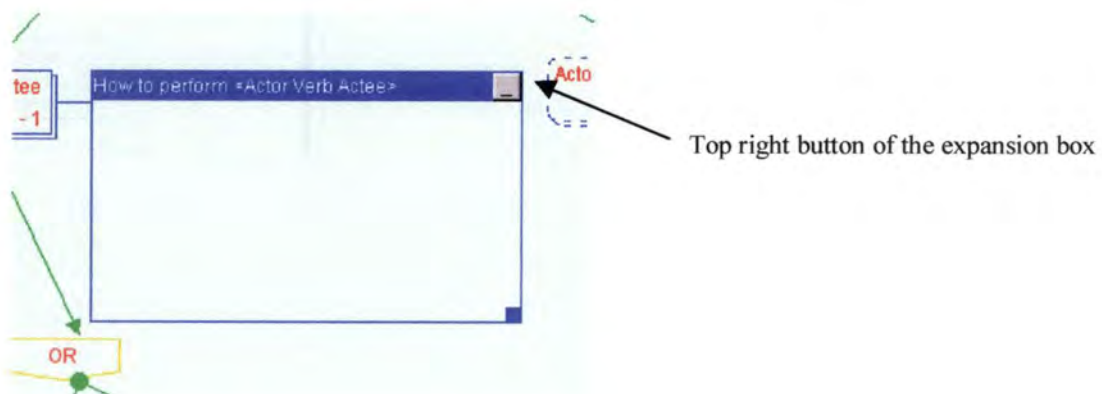


Figure 7.6: How to close an expansion box?

7.2.6 Various

- If we are in the "Select" mode, we can select a widget by clicking on this one; if another widget was selected, it becomes deselected. Only one widget can be selected at one time.

- To move a widget (boolean connector or task), press the left button of the mouse on the widget and drag it where we want to see it. The links (sequences) to this widget are also moving.

Figure 7.7 shows the Graphical User Interface.

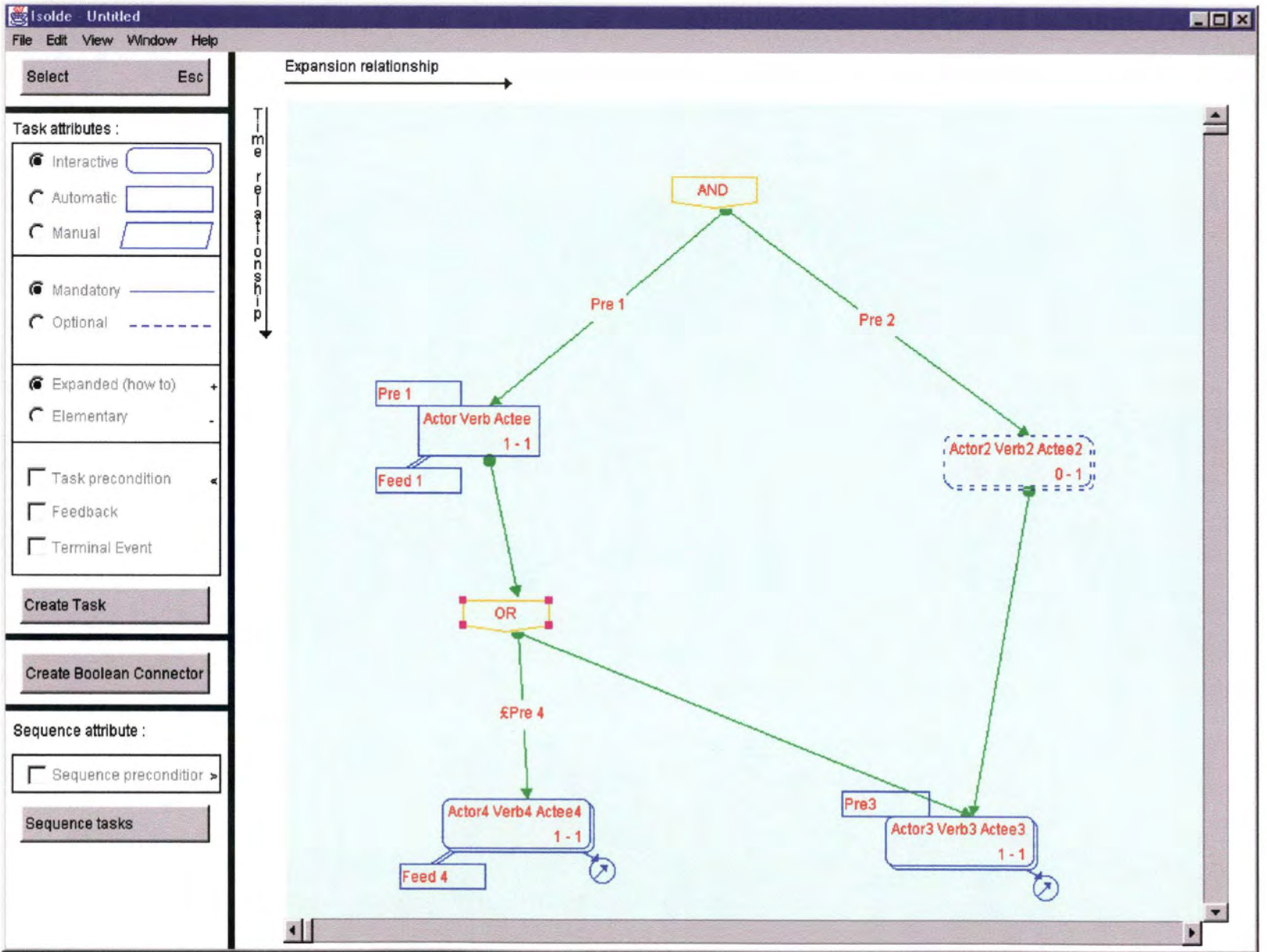


Figure 7.7: The Graphical User Interface

7.3 Suggestions for the improving of the interface

We think that a "Sticky" checkbox would be a good idea. What does it mean? At the present time, after the creation of a widget, we automatically come back in the "Select" mode. So if we want to create a task model with 10 tasks, we had to click 10 times on the "Create Task" button and it is very boring. Now with a "Sticky" checkbox, we can choose the present behavior, click each time on the "Create ..." button or choose the "Sticky" mode, it means that we only click one time and we create as much widgets as we want. Just deselect it when we have finished with the creation of widgets.

To respect the standards, the menu items that are calling another window when we are clicking on it must contain "..." at the end. For example, "Open" -> "Open...", "Import" -> "Import...", "Save" -> "Save...", "Export" -> "Export...".

It would be better if we could find in the menu bar, menu items with the same behaviors than the buttons "Select", "Create task", "Create boolean connector", "Sequence tasks".

A tool bar would be a good idea. We could find there buttons like "New", "Open", "Save", etc.

CHAPTER 8

The natural language generation

Resume:

In chapter 7, we have described the Graphical User Interface.

After all the graphical modifications are done, we can move to the last part of the Isolde project : the language generation itself. This is covered by this chapter.

Chapter 9 will illustrate the complete process with an example.

In section 8.1, we mention what kind of help we actually produce.

In section 8.2, we describe the transition between a the internal representation and the LISP file produced.

Section 8.3 is a remark about the knowledge representation.

8.1 Introduction

Let us remember that the purpose of the Isolde project is to produce end-user documentation. The information usually produced for end-users includes procedural help which can be seen as an answer to the question “how to”.

Procedural help describes the different steps to perform a particular action and is greatly linked to the programmed behavior of the system itself.

From the technical writers’ perspective, procedural help represents the most routine part of their work. They have to check themselves all the possible commands to perform a particular task. It is a very long and tedious task to write that manually and needs not much creativity. That is why we try to do it automatically.

The language used to represent the different steps to perform a user goal must be simple enough to be used by non-experts. As seen in chapter 5, we use the Diane+H formalism.

Nowadays, user’s documentation is very often delivered in hypertext form, on-line, as an integrated component of the software itself. It makes the job of technical writers’ job even more laborious because they must include and maintain the hypertext links.

8.2 The text generator

The text generator is written in LISP. It takes the domain and task models as input and makes texts and sentences. It generates hypertext instructions in English.

8.2.1 Illustration

Let us consider a very simple example illustrated by the following Diane+ model (Figure 8.1):

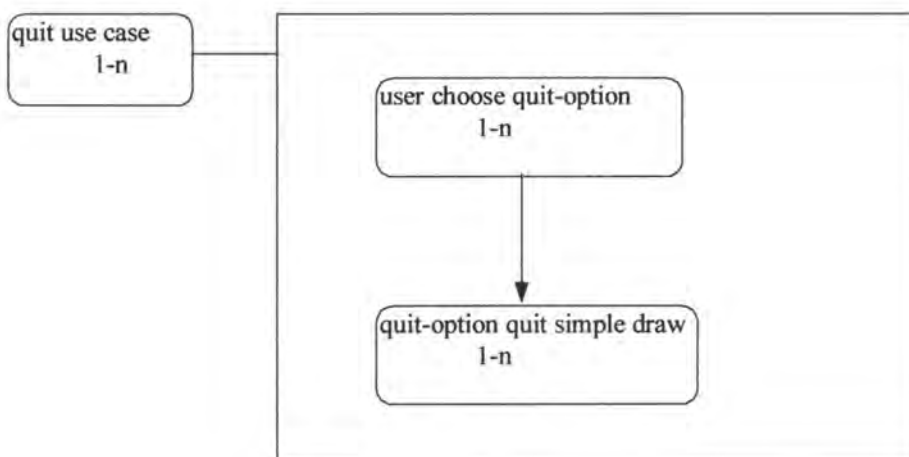


Figure 8. 1: A simple example of a composite task

The first part of the input of the text generator is the task model. The first thing to do is to define every object :

```
;; -----
;; The task model stuff
;; -----
(def-diane-expansion Main
  :start (action-t0)
  )
```

Here, we have defined the Main module

```
(def-diane-expansion expl
  :start (action-t1)
  )
```

The next step was to define the expansion box containing the two elementary tasks. The first task of the expansion box is t1.

```
(def-diane-action action-t1
  :semantics user-choose-quit-option-2
  )
```

Now, we have defined the first task : t1, and have assigned a semantics to it.

```
(def-diane-link link-1
  :domain action-t1
  :range action-t2
  )
```

This is a link between the first and the second task. The domain field corresponds to the task “from”, the range is the task “to”.

```
(def-diane-terminal-event term-2
  :type NORMAL
  )
```

This is to declare a terminal event, i.e. a task that is not linked to a further task. When a terminal event is reached, one must perform the action following the closest expanded task.

```
(def-diane-action action-t2
  :semantics quit-option-quit-simplesdraw-3
  )
```

The second task is defined.

```
(def-diane-link link-2
  :domain action-t2
  :range term-2
  )
```

The second task is terminal, so we add a link from that task to a terminal event.

```
(def-diane-action action-t0
  :expansion expl
  :semantics quit-use-case-1
  )
```

Finally, we define the composite task that is decomposed in an expansion box (expl).

```
(in-package :dm)
```

The second part of the file is the domain model part. It helps the system to situate the objects in the environment.

```

;; -----
;; The domain model stuff
;; -----
(def-dm-instance user
  :dm-concept dm-Object
  )
(def-dm-instance simpledraw
  :dm-concept program-concept
  :lexical-root "simpledraw"
  )
(def4-dm-instance file-menu
  :dm-concept menu-concept
  :lexical-root "file menu"
  :dm-relations ((dm-part-of simpledraw))
  )
(def-dm-instance quit-option
  :dm-concept menu-item-concept
  :lexical-root "quit option"
  :dm-relations ((dm-part-of file-menu))
  )
(def-dm-instance quit-use-case-1
  :dm-concept dm-action
  )
(def-dm-instance user-choose-quit-option-2
  :dm-concept dm-action
  :lexical-root "user choose quit-option"
  :dm-relations ((dm-actor user)
                 (dm-actee quit-option))
  )
(def-dm-instance quit-option-quit-simpledraw-3
  :dm-concept dm-action
  :lexical-root "quit-option quit simpledraw"
  :dm-relations ((dm-actor quit-option)
                 (dm-actee simpledraw))
  )

```

The text generator generates an user's documentation in hypertext form. Here is in Figure 8.2. what the simple example should produce:

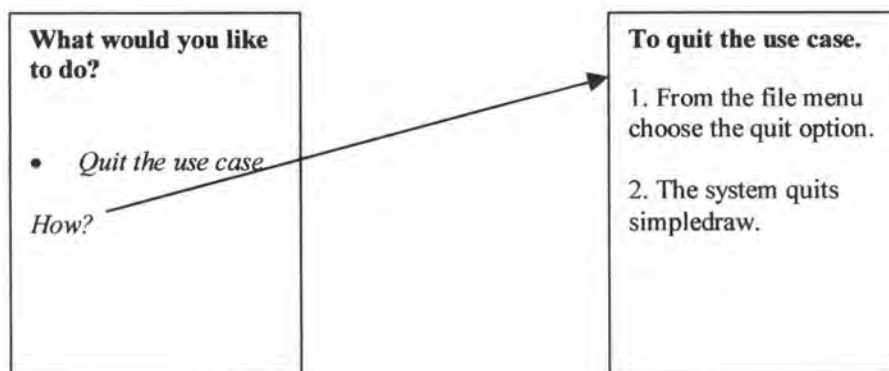


Figure 8. 2: An hypertext example

The 'How' in italic has an hypertext link to the other page.

8.3 A remark concerning the hypertext semantics

Hypertext semantics is very rich and allows any kind of semantic link from a file to another. Any word contained in a file can be associated to a particular location in any other file. The relation between two locations can be of any type.

The Diane+ formalism describes a task model and is limited semantically. As seen in chapter 4, the only relations between two objects can be either a sequence relation, a parallel execution of tasks or an 'expand' relation when a task is expanded into sub-tasks. The boolean connectors also allow multiple sequences from a particular task.

The first relation is represented by a numbered list in the hypertext file; the second one is described as several actions available in the same hypertext file; the third one is represented by a link between the file corresponding to the composite task considered and the file containing its sub-tasks. Boolean connectors also describe parallel tasks and the notion of choice in the execution. In the case of sequence preconditions, the precondition is also added in the hypertext file.

Of course, it would be very interesting to explore more of the hypertext power; but, in generation, the problem is the *knowledge representation*. That knowledge must be represented in an explicit manner to be usable in language generation and the Diane+ formalism, aimed at describing task models, does not allow more relations between objects.

In the future, it would be nice to create new links to other texts completely written manually. It would also be interesting to add different kinds of semantic links from one file to another.

Another interesting work would be to consider a sequence precondition as a logical proposition automatically parsed into elementary objects and analyzed more fully to provide the end-user with more help power and efficiency.

CHAPTER 9

An example

Resume:

After describing the Isolde project and analyzing the task editor in details, this chapter illustrates this thesis with a concrete example.

A demo is also provided on the disk included. The demo is a video sequence shortly describing the steps required to produce hypertext files plus an example of the actual help generated.

Considering a Simple Text Editor (STE), we follow all the steps required for on-line help generation, starting from the Rose file, until the hypertext files containing the end-user help.

First of all, section 9.1 starts with a UML model describing the STE.

In section 9.2, we consider the import file generated by Rose.

In section 9.3, we can see the actual representation of the model.

Section 9.4 covers the topic of the export file.

In section 9.5, we display some of the final hypertext files produced.

9.1 UML model

Figure 9.1 describes a UML diagram related to the STE (Simple Text Editor) containing information useful for text generation.

A converter extracts useful information to generate the import file containing information about domain and task model.

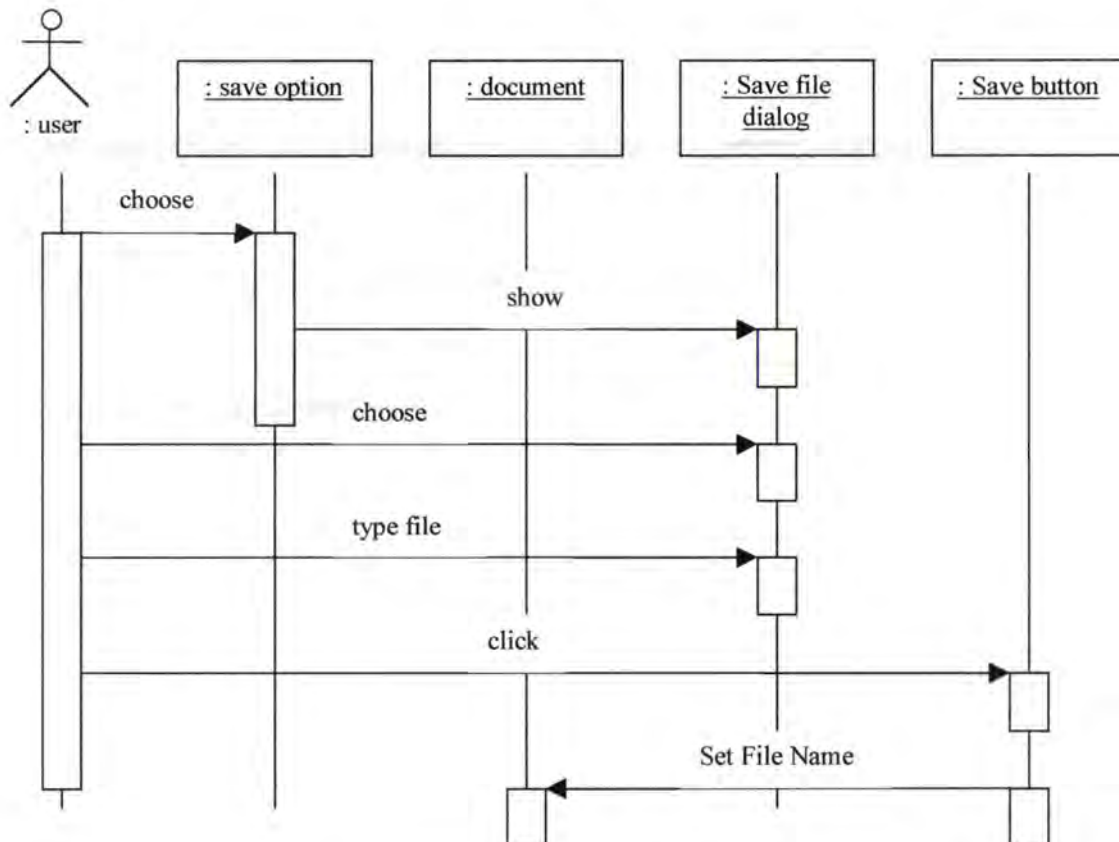


Figure 9. 1: UML diagram related to STE

9.2 Import

In Appendix A, we can see a file generated by Rosascript that describes the STE.

This file is read by the task model editor in order to be modified by the technical writers.

9.3 Display

In Figure 9.2, we illustrate the Diane+ representation of the STE model on the screen. At this moment, the technical writer can modify graphically the model (move a widget, delete a widget, expand a task, etc.).

Until the model is not completely ready for language generation, the technical writer can save it and re-open it as many times as he wants.

When the technical writer agrees with a model, he can export it using the ‘export’ command.

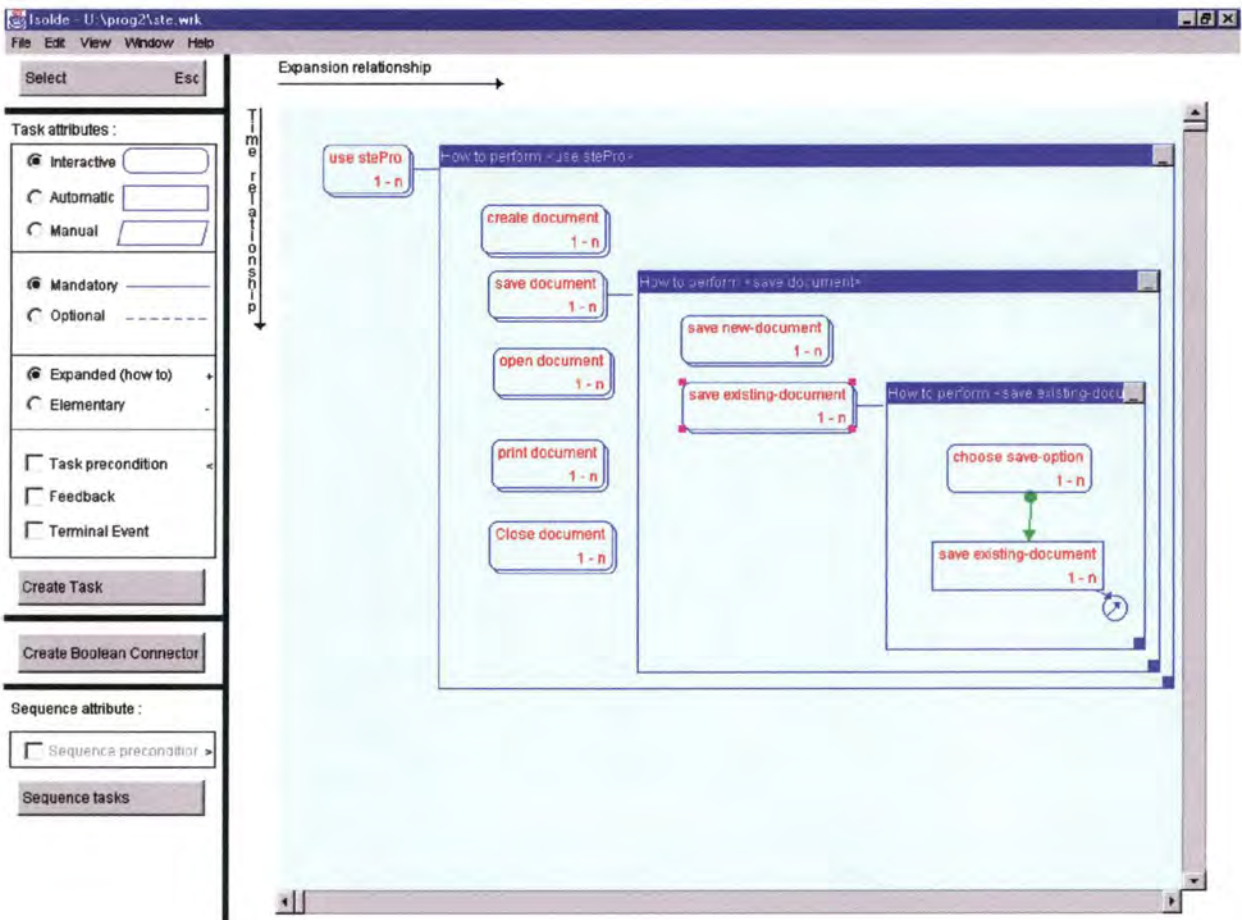


Figure 9. 2: Diane+ representation of the STE model

9.4 Export

When the model is ready for generation, the technical writer can export the model into a LISP file used for language generation. The corresponding file for the stePro (STE program) can be found in Appendix B.

9.5 Language generation

Here is the actual hypertext files generated from the STE model (Figure 9.3).

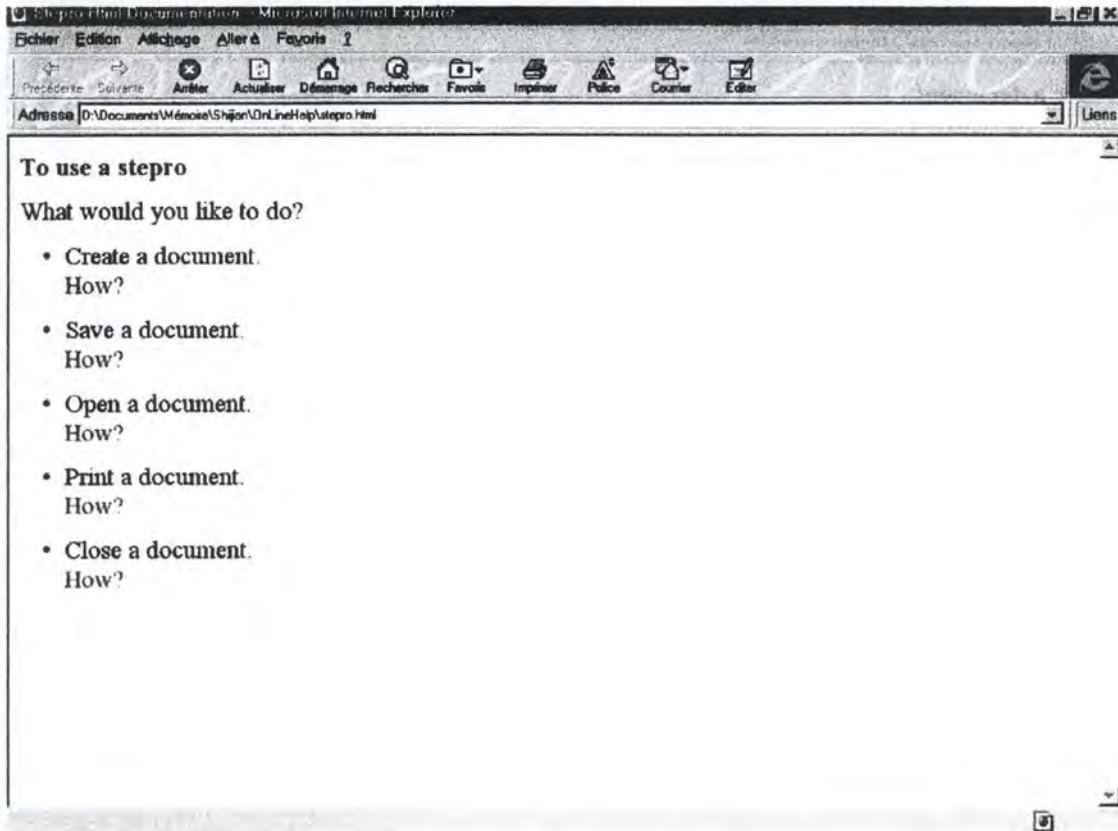


Figure 9. 3: Hypertext file to use a STE program

If the end-user clicks on “How” following “Save a document”, this is the file produced (Figure 9.4)

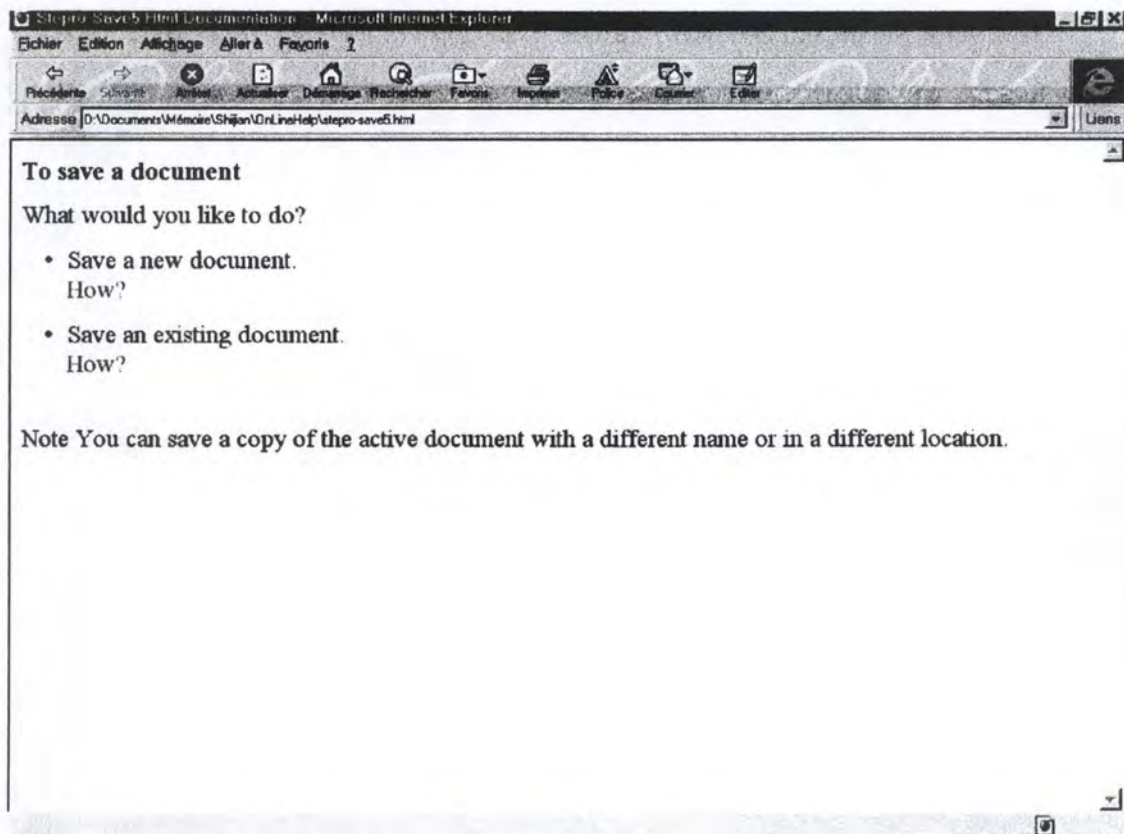


Figure 9. 4: Hypertext file to save a document

Finally, if the user clicks on the “How” following “Save a new document”, here is the hypertext that appears (Figure 9.5)

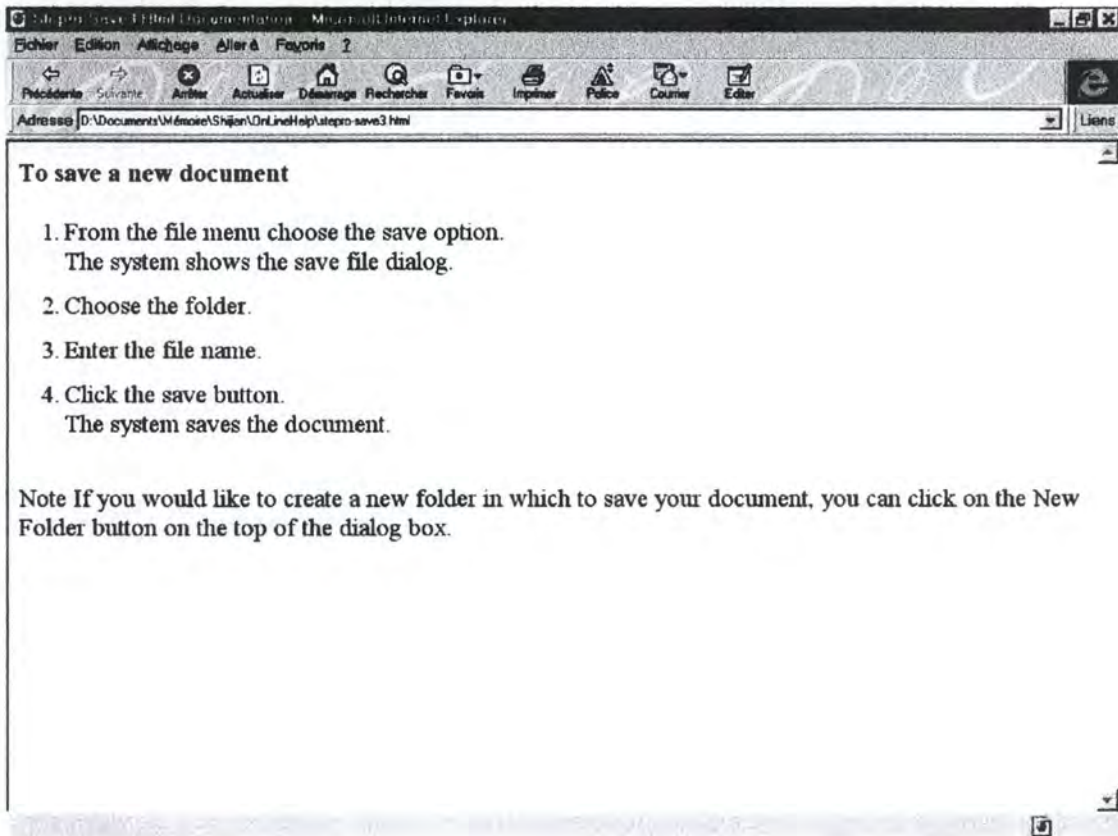


Figure 9. 5: Hypertext file to save a new document

CHAPTER 10

Conclusion

Nowadays, softwares tend to be more and more complex and powerful. Besides, users expect to be assisted by a user-friendly guide any time they ask for it.

Help systems are undoubtedly essential for any elaborate software, and on-line help is a crucial aspect of any software system.

The role of technical writers is currently changing, as the support of on-line help has migrated to hypertext form and that it is desirable to automate production of on-line help.

Help system generators become usable for concrete utilization in commercial softwares and Isolde has a place among them.

Our contribution in the Isolde project was the implementation of the task model editor. The editor includes the reading of an import file generated by Rosascript, the import file describing a particular model. The next step is the transformation of the model into an internal structure and the display of the model on the screen, allowing the technical writers to make change graphically and to save/open a particular model. Finally, the editor exports the model into a LISP file used for language generation.

Some interesting problems we had to deal with were the conversion from the import file into graphical representation, the realization of the graphical interface, and the managing of the structure and the graphical objects after each modification by the technical writers. We have also suggested some improvement for the current interface.

We have learned a lot during our training period at CSIRO working on the Isolde project. It was a great work experience as well as a wonderful opportunity to work with very interesting people.

The Isolde project is not finished and the implementation surely needs some improvements, but we are glad it already produces some good results and is on its way to become a very promising tool for automatic help generation.

Bibliography

[Anciaux 98]

V. Anciaux, "Travail réalisé dans le cadre de la chaire Francqui", FUNDP, Namur, 1998

[Artim 97]

J.M. Artim, "Integrating user interface design and object-oriented development through task analysis and use cases", *CHI'97 workshop on Object Oriented User Interfaces*, 1997

Available at <http://www.cutsys.com/ooi/>

[Balbo 97]

S. Balbo, N. Ozkan, and C. Paris, "Novel uses of task models: two case studies", *Proceedings NATO/ONR workshop on cognitive tasks*, 1997

[Balbo 98]

S. Balbo, N. Ozkan, and C. Paris, "Understanding a Task Model: An Experiment", *Human Computer Interaction '98*, 1998

[Burns 93]

L.M. Burns and A. Malhotra, G. Sockut, K.-Y. Whang, "AERIAL: ad-hoc entity-relationship investigation and learning", *Int. J. Man-Machine Studies*, 38, 1993, pages 607 - 623

[Caldwell 97]

D. E. Caldwell, and M. White, "CogentHelp: a tool for authoring Dynamically Generated Help for Java GUIs", 1997, pages 17 - 22

[Charney 86]

D. H. Charney, and L. M. Reder, "Designing Interactive Tutorials for Computer Users", *Human-computer interaction*, Volume 2, 1986, pages 297 - 317

[CogentHelp]

"CogentHelp Walk-through"

Available at <http://www.cogentex.com/systems/cogenthelp/walkthrough.html>

[Dale]

R. Dale, H. Moisl, and H. Somers, "A Handbook of Natural Language Processing - Techniques and Applications for the Processing of Languages as Text"

Available at <http://www.mri.mq.edu.au/ltg/nlphandbook/contents1.html>

[Deaton 96]

M. Deaton, and C. L. Zubak, *Designing Windows 65 Help: A Guide to Creating Online documents*, Que, 1996

[Elkerton 90]

J. Elkerton, S. J. Goldstein, and S. L. Palmiter, "Designing a Help System Using a GOMS Model: A Preliminary Method Execution Analysis", *Proceedings of the Human Factors Society 34th Annual Meeting*, Volume 1, 1990, pages 250 - 263

Available at

http://www.tu-graz.ac.at/0x811b0205_0x000cdb7b:internal&action=body.action

[Ergoval]

"Thèse, Chapitre 3, La méthode Ergoval"

Available at <http://www.lis.univ-tlse1.fr/~FARENC/these/chapitre3.htm#ch3-4>

[Frank]

M. R. Frank, J.J. "Hans" de Graaff, D. F. Gieskens, and James D. Foley, "Building user interfaces interactively using pre- and postconditions"

[Gwei 90]

G. M. Gwei, and E. Foxley, "Towards a Consultative On-Line Help System", *International Journal of Man-Machine Studies*, Volume 32, Number 4, April 1990, pages 363 - 383

[Green 93]

T. R. G. Green, "European Association for Cognitive Ergonomics, news, reviews and reports", *International Journal of Man-Machine Studies*, Volume 39, 1993, pages 521-528

[Grimm 88]

S. Grimm, J. Malicki, and S. Obermeyer, "A user needs approach to context-sensitive help", *SIGCHI Bulletin*, Volume 19, Number 3, January 1988, pages 65 - 67

[Henry 91]

T. R. Henry, and S. E. Hudson, "Interactive Graph Layout", *Proceedings of the 4th ACM Symposium on User Interface Software and Technology*, South Carolina, 1991, pages 55 - 64

[ISOLDE]

"Isolde: Integrated Software and On-line Documentation Environment, Cecile Paris, CMIS: Current Research Projects"
Available at <http://www.syd.dit.csiro.au/staff/cecile/isolde.html>

[Knabe 95]

K. Knabe, "Apple Guide: A Case Study in User-Aided Design of Online Help", *CHI '95 Mosaic of creativity*, May 7-11 1995, pages 286-287

[Lalioti 94]

V. Lalioti and P. Loucopoulos, "Visualisation of conceptual specifications", *Informations Systems*, Volume 19, Number 3, 1994, pages 291 - 309

[Lu 98a]

S. Lu, C. Paris, and K. Vander Linden, "Towards the Automatic Construction of Task Models from Object-Oriented Diagrams", 1998

[Lu 98b]

S. Lu, C. Paris, and K. Vander Linden, "Integrating Task Modeling into the Object Oriented Design Process: A Pragmatic Approach", 1998

[Mittal 93]

V. O. Mittal, and C. L. Paris, "Intelligent Help Facilities: Generating Natural Language Description with Examples", *Proceedings of the Fifth International Conference on Human-Computer Interaction*, Volume 2, 1993, pages 379 - 384

[Moriyon 94]

R. Moriyon, P. Szekely, and R. Neches, "Automatic Generation of Help from Interface Design Models", *Proceedings of ACM CHI '94 Conference on Human Factors in Computing Systems*, Volume 2, 1994, pages 225-231

[Newman 97]

B. Newman, and R. W. Riner, "Designing Accessible Online Help", *Training & Development*, March 1997, pages 41-44

[Nichols 96]

M. C. Nichols, and R. R. Berry, "Design Principles for Multi-window Online Information Systems: Conclusions from Research, Applications, and Experience", *STC technical communication, Journal of the Society for Technical Communication*, Volume 43, number 3, Third quarter, August 1996

[Palanque]

Ph. Palanque, D. Salber, and R. Bastide, "Gestion automatique de l'aide contextuelle multimédia d'une IHM par exécution de sa spécification formelle"

[Palanque 93]

Ph. A. Palanque, R. Bastide, and L. Dourte, "Contextual help for free with formal dialogue design", *HCI International 93 Conference - Orlando, Florida, USA (August 8 - 13 1993)*, 1993, pages 159 - 164

[Paris 96a]

C. Paris, and K. Vander Linden, "An Interactive Support Tool for Writing Multilingual Manuals", *IEEE Computer, Special Issue on Interactive Natural Language Processing*, 29 (7), July 1996, pages 49 - 56

[Paris 96b]

C. Paris, and K. Vander Linden, "Isolde, Integrated Software and On-Line Documentation Environment, An authoring Tool for On-Line Help, An overview of On-Line Documentation and CASE tools", Report on Tasks: Task 1 and 3, July 1996

[Paris 96c]

C. Paris, K. Vander Linden, "An overview of on-line documentation and CASE tool: Isolde", *Report on task 1 and 3. Technical report ITRI-95-16*, Brighton, 1996

[Paris 98]

C. Paris, K. Vander Linden, S. Lu., "A practical approach to the generation of on-line help", 1998

[Pemberton 96]

L. Pemberton, "Isolde, Integrated Software and On-Line Documentation Environment, An authoring Tool for On-Line Help, Requirements from Technical Writing", Report on Task 2, Nov 1996

[Protsko 91]

L. B. Protsko, P. G. Sorenson, J. P. Tremblay, and D. A. Schaefer, "Towards the Automatic Generation of Software Diagrams", *IEEE Transactions on software engineering*, Volume 17, Number 1, January 1991, pages 10 - 21

[Ryall 97]

K. Ryall, J. Marks, and S. Shieber, "An Interactive Constraint-Based System for Drawing Graphs", UIST 97, Canada, 1997

[Sellen 90]

A. Sellen, and A. Nicol, "Building User-centered On-line Help", 1990, pages 143-153

[Sukaviriya 90]

P. "Noi" Sukaviriya and J. D. Foley, "Coupling A UI Framework with Automatic Generation of Context-Sensitive Animated Help", *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology, Snowbird, Utah, USA, October 3-5 1990/sponsored by ACM SIGGRAPH and ACM SIGCHI*, 1990, pages 152-166

[Sukaviriya 94]

P. "Noi" Sukaviriya, J. Muthukumarasamy, A. Spaans, and H. J.J. de Graaff, "Automatic Generation of Textual, Audio, and Animated Help in UIDE: The User Interface Design Environment", 1994, pages 44 - 52

[Tarby 96]

J.-C. Tarby, and M.-F. Barthet, "The Diane+ method", *Proc. Second international workshop on computer-aided design of user interfaces*, Namur, June 1996

[UML 97]

“UML, Notation Guide: Unified Modelling Language version 1.0”, Rational Software corporation, 1997

[Vander Linden 97]

"K. Vander Linden, Research Interests"

Available at <http://www.calvin.edu/~kvlinden/research.html>

Table of figures

Figure 2. 1: Example of procedural help from MSWord97. Part of the screen for "How to open a file from the hard disk"	20
Figure 3. 1: Quantity of work needed to automatically generate help	23
Figure 3. 2: Quality of the generated help	24
Figure 3. 3: Difficulty of refining the generated help	24
Figure 3. 4: Difficulty of updating the generated help	24
Figure 3. 5: Comparison between Cartoonist and Isolde	26
Figure 3. 6: An example of a contextual help window within a word processing application	28
Figure 3. 7: A high-level Petri net modeling the "Select-Cut-Copy-Paste" functions	29
Figure 3. 8: Reachability graph of the Petri net modeling the "Select-Cut-copy-Paste" functions	31
Figure 3. 9: Comparison between "Contextual help for free with formal dialogue design" and Isolde	32
Figure 3. 10: HelpTalk's response to the user's attempt to select the disabled "Weg OK" button in an InterCity Express (ICE) train reservation system	34
Figure 3. 11: UIDE's knowledge base diagram	35
Figure 3. 12: Example of an action representation	35
Figure 3. 13: UIDE's runtime architecture with HelpTalk	36
Figure 3. 14: Comparison between HelpTalk and Isolde	37
Figure 3. 15: Comparison between H3 and Isolde	40
Figure 3. 16: The "Help" window	43
Figure 3. 17: The "Edit snippets" window	44
Figure 3. 18: The "Consistency check" window	45
Figure 3. 19: The "Edit table of contents" window	45
Figure 3. 20: Dynamic help topics	46
Figure 3. 21: Comparison between CogentHelp and Isolde	47
Figure 4. 1: A use case relationship	52
Figure 4. 2: A use case diagram	53
Figure 4. 3: A sequence diagram	55
Figure 4. 4: A collaboration diagram	56
Figure 4. 5: A state diagram	58
Figure 4. 6: Some graphical notations	60
Figure 5. 1: Classes hierarchy	65
Figure 6. 1: Schema of the model	71
Figure 6. 2: A simple model with elementary Tasks	72
Figure 6. 3: A model with composite Tasks	73

Figure 6. 4: The hierarchy of the objects	74
Figure 6. 5: The internal representation.....	74
Figure 6. 6: The conversion into internal structure with no coordinates.....	92
Figure 6. 7: Addition of a <i>weight</i> parameter	92
Figure 6. 8: Addition of coordinates	93
Figure 6. 9: Addition of coordinates to the sequences.....	93
Figure 7. 1: Buttons and checkboxes of the Graphical User Interface.....	100
Figure 7. 2: The menu item "File"	101
Figure 7. 3: The menu item "Edit"	102
Figure 7. 4: The menu item "View"	103
Figure 7. 5: How to expand an expandable task?.....	104
Figure 7. 6: How to close an expansion box?	104
Figure 7. 7: The Graphical User Interface	106
Figure 8. 1: A simple example of a composite task	111
Figure 8. 2: An hypertext example.....	113
Figure 9. 1: UML diagram related to STE.....	117
Figure 9. 2: Diane+ representation of the STE model.....	118
Figure 9. 3: Hypertext file to use a STE program	119
Figure 9. 4: Hypertext file to save a document	120
Figure 9. 5: Hypertext file to save a new document.....	121

APPENDIX **A**

Import file

This appendix is the import file automatically generated by Rosescript for the Simple Text Editor.

This file generated automatically by Rosescript.

(dm-object)
user
(dm-object)
STE
SystemApplet
STE
(dm-object)
FramePeer
FramePeer
(dm-object)
TextComponent
TextComponent
(dm-object)
Toolkit
Toolkit
(dm-object)
FilenameFilter
FilenameFilter
(dm-object)
ArithmeticException
ArithmeticException
(dm-object)
ClassLoader
ClassLoader
(dm-object)
Double
Double
(dm-object)
InterruptedException
InterruptedException
(dm-object)
NoSuchMethodError
NoSuchMethodError
(dm-object)
String
String
(dm-object)
DatagramSocket
DatagramSocket
(dm-object)
PlainSocketImpl
PlainSocketImpl
(dm-object)
Socket
Socket
(dm-object)
SocketInputStream
SocketInputStream
(dm-object)
Print-Option
MenuItem
file-menu
Print Option
(dm-object)
MenuPeer
MenuPeer
(dm-object)
WindowPeer
WindowPeer
(dm-object)
Checkbox
Checkbox
(dm-object)
Dimension
Dimension

(dm-object)
MenuItem
MenuItem
(dm-object)
OutputStream
OutputStream
(dm-object)
PrintStream
PrintStream
(dm-object)
ArrayStoreException
ArrayStoreException
(dm-object)
Boolean
Boolean
(dm-object)
ClassFormatError
ClassFormatError
(dm-object)
LinkageError
LinkageError
(dm-object)
UnknownContentHandler
UnknownContentHandler
(dm-object)
URLConnection
URLConnection
(dm-object)
Vector
Vector
(dm-object)
file-menu
Menu
STE
file menu
(dm-object)
CanvasPeer
CanvasPeer
(dm-object)
MenuComponentPeer
MenuComponentPeer
(dm-object)
AWTError
AWTError
(dm-object)
BorderLayout
BorderLayout
(dm-object)
CheckboxMenuItem
CheckboxMenuItem
(dm-object)
GridBagConstraints
GridBagConstraints
(dm-object)
FocusManager
FocusManager
(dm-object)
FileInputStream
FileInputStream
(dm-object)
FilterInputStream
FilterInputStream
(dm-object)
StringBufferInputStream
StringBufferInputStream
(dm-object)
SecurityManager

SecurityManager
(dm-object)
StackOverflowError
StackOverflowError
(dm-object)
UnknownHostException
UnknownHostException
(dm-object)
Hashtable
Hashtable
(dm-object)
HashtableEntry
HashtableEntry
(dm-object)
Print-Dialog
STE
Print Dialog
(dm-object)
ContainerPeer
ContainerPeer
(dm-object)
MenuComponent
MenuComponent
(dm-object)
FilterOutputStream
FilterOutputStream
(dm-object)
Process
Process
(dm-object)
RuntimeException
RuntimeException
(dm-object)
StringBuffer
StringBuffer
(dm-object)
Random
Random
(dm-object)
document
document
(dm-object)
folder-name
String
document
folder name
(dm-object)
file-name
String
document
file name
(dm-object)
AppletContext
AppletContext
(dm-object)
DialogPeer
DialogPeer
(dm-object)
TextAreaPeer
TextAreaPeer
(dm-object)
MediaEntry
MediaEntry
(dm-object)
DataInput
DataInput
(dm-object)

EOFException
EOFException
(dm-object)
InputStream
InputStream
(dm-object)
Error
Error
(dm-object)
IllegalAccessException
IllegalAccessException
(dm-object)
Integer
Integer
(dm-object)
UnsatisfiedLinkError
UnsatisfiedLinkError
(dm-object)
UnknownServiceException
UnknownServiceException
(dm-object)
BitSet
BitSet
(dm-object)
Stack
Stack
(dm-object)
PrintJob
PrintJob
(dm-object)
FileDialogPeer
FileDialogPeer
(dm-object)
Canvas
Canvas
(dm-object)
Component
Component
(dm-object)
FileDialog
FileDialog
(dm-object)
GridBagLayout
GridBagLayout
(dm-object)
Insets
Insets
(dm-object)
MediaTracker
MediaTracker
(dm-object)
DataOutputStream
DataOutputStream
(dm-object)
IOException
IOException
(dm-object)
RandomAccessFile
RandomAccessFile
(dm-object)
ArrayIndexOutOfBoundsException
ArrayIndexOutOfBoundsException
(dm-object)
Character
Character
(dm-object)
Class

Class
(dm-object)
IllegalAccessError
IllegalAccessError
(dm-object)
IncompatibleClassChangeError
IncompatibleClassChangeError
(dm-object)
ThreadDeath
ThreadDeath
(dm-object)
ContentHandler
ContentHandler
(dm-object)
SocketException
SocketException
(dm-object)
SocketImpl
SocketImpl
(dm-object)
HashtableEnumerator
HashtableEnumerator
(dm-object)
MemoryImageSource
MemoryImageSource
(dm-object)
PanelPeer
PanelPeer
(dm-object)
CardLayout
CardLayout
(dm-object)
Dialog
Dialog
(dm-object)
FontMetrics
FontMetrics
(dm-object)
BufferedOutputStream
BufferedOutputStream
(dm-object)
DataInputStream
DataInputStream
(dm-object)
PipedOutputStream
PipedOutputStream
(dm-object)
ClassNotFoundException
ClassNotFoundException
(dm-object)
OutOfMemoryError
OutOfMemoryError
(dm-object)
VerifyError
VerifyError
(dm-object)
DatagramPacket
DatagramPacket
(dm-object)
InetAddress
InetAddress
(dm-object)
Observable
Observable
(dm-object)
Applet
Applet

(dm-object)
Event
Event
(dm-object)
MenuContainer
MenuContainer
(dm-object)
Scrollbar
Scrollbar
(dm-object)
FileNotFoundException
FileNotFoundException
(dm-object)
ClassCastException
ClassCastException
(dm-object)
Float
Float
(dm-object)
NegativeArraySizeException
NegativeArraySizeException
(dm-object)
Number
Number
(dm-object)
Observer
Observer
(dm-object)
AppletStub
AppletStub
(dm-object)
PixelGrabber
PixelGrabber
(dm-object)
CheckboxPeer
CheckboxPeer
(dm-object)
Container
Container
(dm-object)
GridBagLayoutInfo
GridBagLayoutInfo
(dm-object)
Image
Image
(dm-object)
FileDescriptor
FileDescriptor
(dm-object)
ClassCircularityError
ClassCircularityError
(dm-object)
IllegalThreadStateException
IllegalThreadStateException
(dm-object)
InstantiationException
InstantiationException
(dm-object)
InternalError
InternalError
(dm-object)
Math
Math
(dm-object)
Win32Process
Win32Process
(dm-object)

ContentHandlerFactory
ContentHandlerFactory
(dm-object)
Enumeration
Enumeration
(dm-object)
Properties
Properties
(dm-object)
New-option
MenuItem
file-menu
New option
(dm-object)
Save_button
Button
Save-file-dialog
Save_button
(dm-object)
ImageConsumer
ImageConsumer
(dm-object)
ComponentPeer
ComponentPeer
(dm-object)
ListPeer
ListPeer
(dm-object)
CheckboxGroup
CheckboxGroup
(dm-object)
Rectangle
Rectangle
(dm-object)
TextArea
TextArea
(dm-object)
TextField
TextField
(dm-object)
StreamTokenizer
StreamTokenizer
(dm-object)
InstantiationError
InstantiationError
(dm-object)
Open-option
MenuItem
file-menu
Open option
(dm-object)
DirectColorModel
DirectColorModel
(dm-object)
ChoicePeer
ChoicePeer
(dm-object)
DataOutput
DataOutput
(dm-object)
PushbackInputStream
PushbackInputStream
(dm-object)
Compiler
Compiler
(dm-object)
Thread

Thread
(dm-object)
VectorEnumerator
VectorEnumerator
(dm-object)
LabelPeer
LabelPeer
(dm-object)
AWTException
AWTException
(dm-object)
Frame
Frame
(dm-object)
MenuBar
MenuBar
(dm-object)
File
File
(dm-object)
FileOutputStream
FileOutputStream
(dm-object)
LineNumberInputStream
LineNumberInputStream
(dm-object)
PipedInputStream
PipedInputStream
(dm-object)
SequenceInputStream
SequenceInputStream
(dm-object)
UTFDataFormatException
UTFDataFormatException
(dm-object)
Cloneable
Cloneable
(dm-object)
ExceptionInInitializerError
ExceptionInInitializerError
(dm-object)
Long
Long
(dm-object)
NoSuchMethodException
NoSuchMethodException
(dm-object)
UnknownError
UnknownError
(dm-object)
VirtualMachineError
VirtualMachineError
(dm-object)
URL
URL
(dm-object)
URLStreamHandlerFactory
URLStreamHandlerFactory
(dm-object)
user
user
(dm-object)
OK_button
Button
Print-Dialog
OK_button
(dm-object)

FilteredImageSource
FilteredImageSource
(dm-object)
ButtonPeer
ButtonPeer
(dm-object)
CheckboxMenuItemPeer
CheckboxMenuItemPeer
(dm-object)
TextComponentPeer
TextComponentPeer
(dm-object)
TextFieldPeer
TextFieldPeer
(dm-object)
FlowLayout
FlowLayout
(dm-object)
Point
Point
(dm-object)
ByteArrayInputStream
ByteArrayInputStream
(dm-object)
ByteArrayOutputStream
ByteArrayOutputStream
(dm-object)
IllegalMonitorStateException
IllegalMonitorStateException
(dm-object)
IndexOutOfBoundsException
IndexOutOfBoundsException
(dm-object)
MalformedURLException
MalformedURLException
(dm-object)
SocketOutputStream
SocketOutputStream
(dm-object)
URLEncoder
URLEncoder
(dm-object)
Dictionary
Dictionary
(dm-object)
StringTokenizer
StringTokenizer
1
(dm-object)
Close-option
MenuItem
file-menu
Close option
(dm-object)
Open-File-Dialog
FileDialog
STE
Open File Dialog
(dm-object)
Save-file-dialog
DialogFileDialog
STE
Save file dialog
(dm-object)
file-name
TextField
Save-file-dialog

file name
(dm-object)
folder-list
Panel
Save-file-dialog
folder list
(dm-object)
CropImageFilter
CropImageFilter
(dm-object)
ImageProducer
ImageProducer
(dm-object)
IndexColorModel
IndexColorModel
(dm-object)
MenuBarPeer
MenuBarPeer
(dm-object)
MenuItemPeer
MenuItemPeer
(dm-object)
Font
Font
(dm-object)
List
List
(dm-object)
NoClassDefFoundError
NoClassDefFoundError
(dm-object)
NumberFormatException
NumberFormatException
(dm-object)
Runnable
Runnable
(dm-object)
SecurityException
SecurityException
(dm-object)
StringIndexOutOfBoundsException
StringIndexOutOfBoundsException
(dm-object)
System
System
(dm-object)
ProtocolException
ProtocolException
(dm-object)
EmptyStackException
EmptyStackException
(dm-object)
save-option
MenuItem
file-menu
save option
(dm-object)
ColorModel
ColorModel
(dm-object)
ImageFilter
ImageFilter
(dm-object)
ImageObserver
ImageObserver
(dm-object)
LayoutManager

LayoutManager
(dm-object)
ImageMediaEntry
ImageMediaEntry
(dm-object)
Menu
Menu
(dm-object)
AbstractMethodError
AbstractMethodError
(dm-object)
NullPointerException
NullPointerException
(dm-object)
Object
Object
(dm-object)
Runtime
Runtime
(dm-object)
Date
Date
(dm-object)
ObserverList
ObserverList
(dm-object)
AudioClip
AudioClip
(dm-object)
GridLayout
GridLayout
(dm-object)
Panel
Panel
(dm-object)
BufferedInputStream
BufferedInputStream
(dm-object)
InterruptedException
InterruptedException
(dm-object)
Exception
Exception
(dm-object)
IllegalArgumentException
IllegalArgumentException
(dm-object)
Throwable
Throwable
(dm-object)
ServerSocket
ServerSocket
(dm-object)
URLStreamHandler
URLStreamHandler
(dm-object)
NoSuchElementException
NoSuchElementException
(dm-object)
Open_button
Button
Open-File-Dialog
Open_button
(dm-object)
RGBImageFilter
RGBImageFilter
(dm-object)

ScrollbarPeer
 ScrollbarPeer
 (dm-object)
 Button
 Button
 (dm-object)
 Choice
 Choice
 (dm-object)
 Color
 Color
 (dm-object)
 Graphics
 Graphics
 (dm-object)
 Label
 Label
 (dm-object)
 Polygon
 Polygon
 (dm-object)
 Window
 Window
 (dm-object)
 CloneNotSupportedException
 CloneNotSupportedException
 (dm-object)
 NoSuchFieldError
 NoSuchFieldError
 (dm-object)
 ThreadGroup
 ThreadGroup
 (dm-object)
 SocketImplFactory
 SocketImplFactory
 (Task)
 use-stePro
 use stePro
 No
 Mandatory
 Interactive
 Expanded
 (Expansion)
 use-stePro
 create-document
 save-document
 open-document
 print-document
 Close-document
 (Task)
 create-document
 create document
 No
 Mandatory
 Interactive
 Expanded
 (dm-action)
 create-document
 user
 document
 create
 (Expansion)
 create-document
 user-choose-New-option-1
 (Task)
 user-choose-New-option-1
 choose New-option

No
Mandatory
Interactive
Elementary
(Task)
New-option-addDocument-STE-2
addDocument STE
Yes
Mandatory
Automatic
Elementary
(Sequence)
user-choose-New-option-1
New-option-addDocument-STE-2
(dm-action)
user-choose-New-option-1
user
New-option
choose
(dm-action)
New-option-addDocument-STE-2
system
STE
addDocument
(Task)
save-document
save document
No
Mandatory
Interactive
Expanded
(dm-action)
save-document
user
document
save
(Expansion)
save-document
save-new-document
save-existing-document
(Task)
save-new-document
save new-document
No
Mandatory
Interactive
Expanded
(dm-action)
save-new-document
user
new-document
save
(Expansion)
save-new-document
user-choose-save-option-3
(Task)
user-choose-save-option-3
choose save-option
No
Mandatory
Interactive
Elementary
(Task)
save-option-show-Save-file-dialog-4
show Save-file-dialog
No
Mandatory

Automatic
 Elementary
 (Task)
 user-choose-folder-Save-file-dialog-5
 choose-folder Save-file-dialog
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 user-type-file-name-Save-file-dialog-6
 type-file-name Save-file-dialog
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 user-click-Save_button-7
 click Save_button
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 Save_button-setFileName-document-8
 setFileName document
 Yes
 Mandatory
 Automatic
 Elementary
 (Sequence)
 user-choose-save-option-3
 save-option-show-Save-file-dialog-4
 (Sequence)
 save-option-show-Save-file-dialog-4
 user-choose-folder-Save-file-dialog-5
 (Sequence)
 user-choose-folder-Save-file-dialog-5
 user-type-file-name-Save-file-dialog-6
 (Sequence)
 user-type-file-name-Save-file-dialog-6
 user-click-Save_button-7
 (Sequence)
 user-click-Save_button-7
 Save_button-setFileName-document-8
 (dm-action)
 user-choose-save-option-3
 user
 save-option
 choose
 (dm-action)
 save-option-show-Save-file-dialog-4
 system
 Save-file-dialog
 show
 (dm-action)
 user-choose-folder-Save-file-dialog-5
 user
 Save-file-dialog
 choose-folder
 (dm-action)
 user-type-file-name-Save-file-dialog-6
 user
 Save-file-dialog
 type-file-name
 (dm-action)
 user-click-Save_button-7

user
Save_button
click
(dm-action)
Save_button-setFileName-document-8
system
document
setFileName
(Task)
save-existing-document
save existing-document
No
Mandatory
Interactive
Expanded
(dm-action)
save-existing-document
user
existing-document
save
(Expansion)
save-existing-document
user-choose-save-option-9
(Task)
user-choose-save-option-9
choose save-option
No
Mandatory
Interactive
Elementary
(Task)
save-option-save-document-10
save document
Yes
Mandatory
Automatic
Elementary
(Sequence)
user-choose-save-option-9
save-option-save-document-10
(dm-action)
user-choose-save-option-9
user
save-option
choose
(dm-action)
save-option-save-document-10
system
document
save
(Task)
open-document
open document
No
Mandatory
Interactive
Expanded
(dm-action)
open-document
user
document
open
(Expansion)
open-document
user-Choose-Open-option-11
(Task)
user-Choose-Open-option-11

Choose Open-option
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 STE-Show-Open-File-Dialog-12
 Show Open-File-Dialog
 No
 Mandatory
 Automatic
 Elementary
 (Task)
 user-Choose-directory-Open-File-Dialog-13
 Choose-directory Open-File-Dialog
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 user-Choose-File-Open-File-Dialog-14
 Choose-File Open-File-Dialog
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 user-Click-Open_button-15
 Click Open_button
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 Open_button-getDirectory-Open-File-Dialog-16
 getDirectory Open-File-Dialog
 Yes
 Mandatory
 Automatic
 Elementary
 (Sequence)
 user-Choose-Open-option-11
 STE-Show-Open-File-Dialog-12
 (Sequence)
 STE-Show-Open-File-Dialog-12
 user-Choose-directory-Open-File-Dialog-13
 (Sequence)
 user-Choose-directory-Open-File-Dialog-13
 user-Choose-File-Open-File-Dialog-14
 (Sequence)
 user-Choose-File-Open-File-Dialog-14
 user-Click-Open_button-15
 (Sequence)
 user-Click-Open_button-15
 Open_button-getDirectory-Open-File-Dialog-16
 (dm-action)
 user-Choose-Open-option-11
 user
 Open-option
 Choose
 (dm-action)
 STE-Show-Open-File-Dialog-12
 system
 Open-File-Dialog
 Show
 (dm-action)
 user-Choose-directory-Open-File-Dialog-13

STE
Open-File-Dialog
Choose-directory
(dm-action)
user-Choose-File-Open-File-Dialog-14
user
Open-File-Dialog
Choose-File
(dm-action)
user-Click-Open_button-15
user
Open_button
Click
(dm-action)
Open_button-getDirectory-Open-File-Dialog-16
system
Open-File-Dialog
getDirectory
(Task)
print-document
print document
No
Mandatory
Interactive
Expanded
(dm-action)
print-document
user
document
print
(Expansion)
print-document
user-Choose-Print-Option-17
(Task)
user-Choose-Print-Option-17
Choose Print-Option
No
Mandatory
Interactive
Elementary
(Task)
Toolkit-show-Print-Dialog-18
show Print-Dialog
No
Mandatory
Automatic
Elementary
(Task)
user-Select-printer-Print-Dialog-19
Select-printer Print-Dialog
No
Mandatory
Interactive
Elementary
(Task)
user-set-No-of-copy-Print-Dialog-20
set-No-of-copy Print-Dialog
No
Mandatory
Interactive
Elementary
(Task)
user-set-print-range-Print-Dialog-21
set-print-range Print-Dialog
No
Mandatory
Interactive

Elementary
 (Task)
 user-Click-OK_button-22
 Click OK_button
 No
 Mandatory
 Interactive
 Elementary
 (Task)
 OK_button-getGraphics-PrintJob-23
 getGraphics PrintJob
 Yes
 Mandatory
 Automatic
 Elementary
 (Sequence)
 user-Choose-Print-Option-17
 Toolkit-show-Print-Dialog-18
 (Sequence)
 Toolkit-show-Print-Dialog-18
 user-Select-printer-Print-Dialog-19
 (Sequence)
 user-Select-printer-Print-Dialog-19
 user-set-No-of-copy-Print-Dialog-20
 (Sequence)
 user-set-No-of-copy-Print-Dialog-20
 user-set-print-range-Print-Dialog-21
 (Sequence)
 user-set-print-range-Print-Dialog-21
 user-Click-OK_button-22
 (Sequence)
 user-Click-OK_button-22
 OK_button-getGraphics-PrintJob-23
 (dm-action)
 user-Choose-Print-Option-17
 user
 Print-Option
 Choose
 (dm-action)
 Toolkit-show-Print-Dialog-18
 system
 Print-Dialog
 show
 (dm-action)
 user-Select-printer-Print-Dialog-19
 document
 Print-Dialog
 Select-printer
 (dm-action)
 user-set-No-of-copy-Print-Dialog-20
 Toolkit
 Print-Dialog
 set-No-of-copy
 (dm-action)
 user-set-print-range-Print-Dialog-21
 user
 Print-Dialog
 set-print-range
 (dm-action)
 user-Click-OK_button-22
 user
 OK_button
 Click
 (dm-action)
 OK_button-getGraphics-PrintJob-23
 system
 PrintJob

getGraphics
(Task)
Close-document
Close document
No
Mandatory
Interactive
Expanded
(dm-action)
Close-document
user
document
Close
(Expansion)
Close-document
user-choose-Close-option-24
(Task)
user-choose-Close-option-24
choose Close-option
No
Mandatory
Interactive
Elementary
(Task)
Close-option-quit-STE-25
quit STE
Yes
Mandatory
Automatic
Elementary
(Sequence)
user-choose-Close-option-24
Close-option-quit-STE-25
(dm-action)
user-choose-Close-option-24
user
Close-option
choose
(dm-action)
Close-option-quit-STE-25
system
STE
quit
(dm-action)
use-stePro
user
stePro
use

APPENDIX **B**

Export file

This appendix is the Lisp file produced by the task modeling tool for the Simple Text Editor.

```
(in-package :dm)

;; -----
;; The task model stuff
;; -----

(def-diane-expansion Main
  :start (action-t33)
  )
(def-diane-expansion exp1
  :start (action-t34 action-t37 action-t48 action-t55 action-t63)
  )
(def-diane-expansion exp2
  :start (action-t35)
  )
(def-diane-action action-t35
  :semantics user-choose-New-option-1
  )
(def-diane-link link-1
  :domain action-t35
  :range action-t36
  )
(def-diane-terminal-event term-36
  :type NORMAL
  )
(def-diane-action action-t36
  :semantics system-create-new-document-2
  )
(def-diane-link link-2
  :domain action-t36
  :range term-36
  )
(def-diane-action action-t34
  :expansion exp2
  :semantics create-document
  )
(def-diane-expansion exp3
  :start (action-t38 action-t45)
  )
(def-diane-expansion exp4
  :start (action-t39)
  )
(def-diane-action action-t39
  :semantics user-choose-save-option-3
  )
(def-diane-link link-3
  :domain action-t39
  :range action-t40
  )
(def-diane-action action-t40
  :semantics save-option-show-Save-file-dialog-4
  )
(def-diane-link link-4
  :domain action-t40
  :range action-t41
  )
(def-diane-action action-t41
  :semantics user-choose-folder-7
  )
(def-diane-link link-5
  :domain action-t41
  :range action-t42
  )
(def-diane-action action-t42
  :semantics user-enter-file-name-8
  :documentation "Note You can use long descriptive file names if you want. "
  )
```

```

(def-diane-link link-6
  :domain action-t42
  :range action-t43
)
(def-diane-action action-t43
  :semantics user-click-Save_button-7
)
(def-diane-link link-7
  :domain action-t43
  :range action-t44
)
(def-diane-terminal-event term-44
  :type NORMAL
)
(def-diane-action action-t44
  :semantics system-save-document-10
)
(def-diane-link link-8
  :domain action-t44
  :range term-44
)
(def-diane-action action-t38
  :expansion exp4
  :semantics save-new-document
  :documentation "Note If you would like to create a new folder in which to save your document, you can click
on the New Folder button on the top of the dialog box. "
)
(def-diane-expansion exp5
  :start (action-t46)
)
(def-diane-action action-t46
  :semantics user-choose-save-option-9
)
(def-diane-link link-9
  :domain action-t46
  :range action-t47
)
(def-diane-terminal-event term-47
  :type NORMAL
)
(def-diane-action action-t47
  :semantics system-save-existing-document-13
)
(def-diane-link link-10
  :domain action-t47
  :range term-47
)

(def-diane-action action-t45
  :expansion exp5
  :semantics save-existing-document
  :documentation "Note You can save a copy of the active document with a different name or in a different
location. "
)

(def-diane-action action-t37
  :expansion exp3
  :semantics save-document
  :documentation "Note You can save a copy of the active document with a different name or in a different
location. "
)
(def-diane-expansion exp6
  :start (action-t49)
)
(def-diane-action action-t49
  :semantics user-Choose-Open-option-11
)

```

```
(def-diane-link link-11
  :domain action-t49
  :range action-t50
  )
(def-diane-action action-t50
  :semantics STE-Show-Open-File-Dialog-12
  )
(def-diane-link link-12
  :domain action-t50
  :range action-t51
  )
(def-diane-action action-t51
  :semantics user-Choose-directory-17
  )
(def-diane-link link-13
  :domain action-t51
  :range action-t52
  )
(def-diane-action action-t52
  :semantics user-Choose-File-18
  )
(def-diane-link link-14
  :domain action-t52
  :range action-t53
  )
(def-diane-action action-t53
  :semantics user-Click-Open_button-15
  )
(def-diane-link link-15
  :domain action-t53
  :range action-t54
  )
(def-diane-terminal-event term-54
  :type NORMAL
  )
(def-diane-action action-t54
  :semantics system-Open-document-20
  )
(def-diane-link link-16
  :domain action-t54
  :range term-54
  )
(def-diane-action action-t48
  :expansion exp6
  :semantics open-document
  )
(def-diane-expansion exp7
  :start (action-t56)
  )
(def-diane-action action-t56
  :semantics user-Choose-Print-Option-17
  )
(def-diane-link link-17
  :domain action-t56
  :range action-t57
  )
(def-diane-action action-t57
  :semantics Toolkit-show-Print-Dialog-18
  )
(def-diane-link link-18
  :domain action-t57
  :range action-t58
  )
(def-diane-action action-t58
  :semantics user-Select-printer-24
  )
(def-diane-link link-19
```

```

:domain action-t58
:range action-t59
)
(def-diane-action action-t59
:semantics user-set-No-of-copy-25
)
(def-diane-link link-20
:domain action-t59
:range action-t60
)
(def-diane-action action-t60
:semantics user-set-print-range-26
)
(def-diane-link link-21
:domain action-t60
:range action-t61
)
(def-diane-action action-t61
:semantics user-Click-OK_button-22
)
(def-diane-link link-22
:domain action-t61
:range action-t62
)
(def-diane-terminal-event term-62
:type NORMAL
)
(def-diane-action action-t62
:semantics system-print-document-28
)
(def-diane-link link-23
:domain action-t62
:range term-62
)
(def-diane-action action-t55
:expansion exp7
:semantics print-document
)
(def-diane-expansion exp8
:start (action-t64)
)
(def-diane-action action-t64
:semantics user-choose-Close-option-24
)
(def-diane-link link-24
:domain action-t64
:range action-t65
)
(def-diane-terminal-event term-65
:type NORMAL
)
(def-diane-action action-t65
:semantics Close-option-quit-STE-25
)
(def-diane-link link-25
:domain action-t65
:range term-65
)
(def-diane-action action-t63
:expansion exp8
:semantics Close-document
)
(def-diane-action action-t33
:expansion exp1
:semantics use-stePro
:documentation "STEpro is a simple text editor program. It allows you to do standard manipulation of
document."

```

```
)  
(in-package :dm)  
;;-----  
;; The domain model stuff  
;;-----  
(def-dm-instance user  
 :dm-concept dm-Object  
 )  
(def-dm-instance STE  
 :dm-concept dm-Object  
 :lexical-root "STE"  
 )  
(def-dm-instance FramePeer  
 :dm-concept dm-Object  
 :lexical-root "FramePeer"  
 )  
(def-dm-instance TextComponent  
 :dm-concept dm-Object  
 :lexical-root "TextComponent"  
 )  
(def-dm-instance Toolkit  
 :dm-concept dm-Object  
 :lexical-root "Toolkit"  
 )  
(def-dm-instance FilenameFilter  
 :dm-concept dm-Object  
 :lexical-root "FilenameFilter"  
 )  
(def-dm-instance ArithmeticException  
 :dm-concept dm-Object  
 :lexical-root "ArithmeticException"  
 )  
(def-dm-instance ClassLoader  
 :dm-concept dm-Object  
 :lexical-root "ClassLoader"  
 )  
(def-dm-instance Double  
 :dm-concept dm-Object  
 :lexical-root "Double"  
 )  
(def-dm-instance InterruptedException  
 :dm-concept dm-Object  
 :lexical-root "InterruptedException"  
 )  
(def-dm-instance NoSuchMethodError  
 :dm-concept dm-Object  
 :lexical-root "NoSuchMethodError"  
 )  
(def-dm-instance String  
 :dm-concept dm-Object  
 :lexical-root "String"  
 )  
(def-dm-instance DatagramSocket  
 :dm-concept dm-Object  
 :lexical-root "DatagramSocket"  
 )  
(def-dm-instance PlainSocketImpl  
 :dm-concept dm-Object  
 :lexical-root "PlainSocketImpl"  
 )  
(def-dm-instance Socket  
 :dm-concept dm-Object  
 :lexical-root "Socket"  
 )  
(def-dm-instance SocketInputStream  
 :dm-concept dm-Object  
 :lexical-root "SocketInputStream"
```

```
)
(def-dm-instance Print-Option
 :dm-concept menu-item-concept
 :lexical-root "Print Option"
 :dm-relations ((dm-part-of file-menu))
)
(def-dm-instance MenuPeer
 :dm-concept dm-Object
 :lexical-root "MenuPeer"
)
(def-dm-instance WindowPeer
 :dm-concept dm-Object
 :lexical-root "WindowPeer"
)
(def-dm-instance Checkbox
 :dm-concept dm-Object
 :lexical-root "Checkbox"
)
(def-dm-instance Dimension
 :dm-concept dm-Object
 :lexical-root "Dimension"
)
(def-dm-instance MenuItem
 :dm-concept dm-Object
 :lexical-root "MenuItem"
)
(def-dm-instance OutputStream
 :dm-concept dm-Object
 :lexical-root "OutputStream"
)
(def-dm-instance PrintStream
 :dm-concept dm-Object
 :lexical-root "PrintStream"
)
(def-dm-instance ArrayStoreException
 :dm-concept dm-Object
 :lexical-root "ArrayStoreException"
)
(def-dm-instance Boolean
 :dm-concept dm-Object
 :lexical-root "Boolean"
)
(def-dm-instance ClassFormatError
 :dm-concept dm-Object
 :lexical-root "ClassFormatError"
)
(def-dm-instance LinkageError
 :dm-concept dm-Object
 :lexical-root "LinkageError"
)
(def-dm-instance UnknownContentHandler
 :dm-concept dm-Object
 :lexical-root "UnknownContentHandler"
)
(def-dm-instance URLConnection
 :dm-concept dm-Object
 :lexical-root "URLConnection"
)
(def-dm-instance Vector
 :dm-concept dm-Object
 :lexical-root "Vector"
)
(def-dm-instance file-menu
 :dm-concept menu-concept
 :lexical-root "file menu"
 :dm-relations ((dm-part-of STE))
)
```



```
(def-dm-instance CanvasPeer
  :dm-concept dm-Object
  :lexical-root "CanvasPeer"
)
(def-dm-instance MenuComponentPeer
  :dm-concept dm-Object
  :lexical-root "MenuComponentPeer"
)
(def-dm-instance AWTError
  :dm-concept dm-Object
  :lexical-root "AWTError"
)
(def-dm-instance BorderLayout
  :dm-concept dm-Object
  :lexical-root "BorderLayout"
)
(def-dm-instance CheckboxMenuItem
  :dm-concept dm-Object
  :lexical-root "CheckboxMenuItem"
)
(def-dm-instance GridBagConstraints
  :dm-concept dm-Object
  :lexical-root "GridBagConstraints"
)
(def-dm-instance FocusManager
  :dm-concept dm-Object
  :lexical-root "FocusManager"
)
(def-dm-instance FileInputStream
  :dm-concept dm-Object
  :lexical-root "FileInputStream"
)
(def-dm-instance FilterInputStream
  :dm-concept dm-Object
  :lexical-root "FilterInputStream"
)
(def-dm-instance StringBufferInputStream
  :dm-concept dm-Object
  :lexical-root "StringBufferInputStream"
)
(def-dm-instance SecurityManager
  :dm-concept dm-Object
  :lexical-root "SecurityManager"
)
(def-dm-instance StackOverflowError
  :dm-concept dm-Object
  :lexical-root "StackOverflowError"
)
(def-dm-instance UnknownHostException
  :dm-concept dm-Object
  :lexical-root "UnknownHostException"
)
(def-dm-instance Hashtable
  :dm-concept dm-Object
  :lexical-root "Hashtable"
)
(def-dm-instance HashtableEntry
  :dm-concept dm-Object
  :lexical-root "HashtableEntry"
)
(def-dm-instance Print-Dialog
  :dm-concept dm-Object
  :lexical-root "Print Dialog"
  :dm-relations ((dm-part-of STE))
)
(def-dm-instance ContainerPeer
  :dm-concept dm-Object
```

```
:lexical-root "ContainerPeer"
)
(def-dm-instance MenuComponent
:dm-concept dm-Object
:lexical-root "MenuComponent"
)
(def-dm-instance FilterOutputStream
:dm-concept dm-Object
:lexical-root "FilterOutputStream"
)
(def-dm-instance Process
:dm-concept dm-Object
:lexical-root "Process"
)
(def-dm-instance RuntimeException
:dm-concept dm-Object
:lexical-root "RuntimeException"
)
(def-dm-instance StringBuffer
:dm-concept dm-Object
:lexical-root "StringBuffer"
)
(def-dm-instance Random
:dm-concept dm-Object
:lexical-root "Random"
)
(def-dm-instance document
:dm-concept dm-Object
:lexical-root "document"
)
(def-dm-instance folder-name
:dm-concept dm-Object
:lexical-root "folder name"
:dm-relations ((dm-part-of document))
)
(def-dm-instance file-name
:dm-concept dm-Object
:lexical-root "file name"
:dm-relations ((dm-part-of document))
)
(def-dm-instance AppletContext
:dm-concept dm-Object
:lexical-root "AppletContext"
)
(def-dm-instance DialogPeer
:dm-concept dm-Object
:lexical-root "DialogPeer"
)
(def-dm-instance TextAreaPeer
:dm-concept dm-Object
:lexical-root "TextAreaPeer"
)
(def-dm-instance MediaEntry
:dm-concept dm-Object
:lexical-root "MediaEntry"
)
(def-dm-instance DataInput
:dm-concept dm-Object
:lexical-root "DataInput"
)
(def-dm-instance EOFException
:dm-concept dm-Object
:lexical-root "EOFException"
)
(def-dm-instance InputStream
:dm-concept dm-Object
:lexical-root "InputStream"
```

```
)
(def-dm-instance Error
 :dm-concept dm-Object
 :lexical-root "Error"
)
(def-dm-instance IllegalAccessException
 :dm-concept dm-Object
 :lexical-root "IllegalAccessException"
)
(def-dm-instance Integer
 :dm-concept dm-Object
 :lexical-root "Integer"
)
(def-dm-instance UnsatisfiedLinkError
 :dm-concept dm-Object
 :lexical-root "UnsatisfiedLinkError"
)
(def-dm-instance UnknownServiceException
 :dm-concept dm-Object
 :lexical-root "UnknownServiceException"
)
(def-dm-instance BitSet
 :dm-concept dm-Object
 :lexical-root "BitSet"
)
(def-dm-instance Stack
 :dm-concept dm-Object
 :lexical-root "Stack"
)
(def-dm-instance PrintJob
 :dm-concept dm-Object
 :lexical-root "PrintJob"
)
(def-dm-instance FileDialogPeer
 :dm-concept dm-Object
 :lexical-root "FileDialogPeer"
)
(def-dm-instance Canvas
 :dm-concept dm-Object
 :lexical-root "Canvas"
)
(def-dm-instance Component
 :dm-concept dm-Object
 :lexical-root "Component"
)
(def-dm-instance FileDialog
 :dm-concept dm-Object
 :lexical-root "FileDialog"
)
(def-dm-instance GridBagLayout
 :dm-concept dm-Object
 :lexical-root "GridBagLayout"
)
(def-dm-instance Insets
 :dm-concept dm-Object
 :lexical-root "Insets"
)
(def-dm-instance MediaTracker
 :dm-concept dm-Object
 :lexical-root "MediaTracker"
)
(def-dm-instance DataOutputStream
 :dm-concept dm-Object
 :lexical-root "DataOutputStream"
)
(def-dm-instance IOException
 :dm-concept dm-Object
```

```
:lexical-root "IOException"
)
(def-dm-instance RandomAccessFile
 :dm-concept dm-Object
 :lexical-root "RandomAccessFile"
)
(def-dm-instance ArrayIndexOutOfBoundsException
 :dm-concept dm-Object
 :lexical-root "ArrayIndexOutOfBoundsException"
)
(def-dm-instance Character
 :dm-concept dm-Object
 :lexical-root "Character"
)
(def-dm-instance Class
 :dm-concept dm-Object
 :lexical-root "Class"
)
(def-dm-instance IllegalAccessException
 :dm-concept dm-Object
 :lexical-root "IllegalAccessException"
)
(def-dm-instance IncompatibleClassChangeError
 :dm-concept dm-Object
 :lexical-root "IncompatibleClassChangeError"
)
(def-dm-instance ThreadDeath
 :dm-concept dm-Object
 :lexical-root "ThreadDeath"
)
(def-dm-instance ContentHandler
 :dm-concept dm-Object
 :lexical-root "ContentHandler"
)
(def-dm-instance SocketException
 :dm-concept dm-Object
 :lexical-root "SocketException"
)
(def-dm-instance SocketImpl
 :dm-concept dm-Object
 :lexical-root "SocketImpl"
)
(def-dm-instance HashtableEnumerator
 :dm-concept dm-Object
 :lexical-root "HashtableEnumerator"
)
(def-dm-instance MemoryImageSource
 :dm-concept dm-Object
 :lexical-root "MemoryImageSource"
)
(def-dm-instance PanelPeer
 :dm-concept dm-Object
 :lexical-root "PanelPeer"
)
(def-dm-instance CardLayout
 :dm-concept dm-Object
 :lexical-root "CardLayout"
)
(def-dm-instance Dialog
 :dm-concept dm-Object
 :lexical-root "Dialog"
)
(def-dm-instance FontMetrics
 :dm-concept dm-Object
 :lexical-root "FontMetrics"
)
(def-dm-instance BufferedOutputStream
```

```
:dm-concept dm-Object
:lexical-root "BufferedOutputStream"
)
(def-dm-instance DataInputStream
:dm-concept dm-Object
:lexical-root "DataInputStream"
)
(def-dm-instance PipedOutputStream
:dm-concept dm-Object
:lexical-root "PipedOutputStream"
)
(def-dm-instance ClassNotFoundException
:dm-concept dm-Object
:lexical-root "ClassNotFoundException"
)
(def-dm-instance OutOfMemoryError
:dm-concept dm-Object
:lexical-root "OutOfMemoryError"
)
(def-dm-instance VerifyError
:dm-concept dm-Object
:lexical-root "VerifyError"
)
(def-dm-instance DatagramPacket
:dm-concept dm-Object
:lexical-root "DatagramPacket"
)
(def-dm-instance InetAddress
:dm-concept dm-Object
:lexical-root "InetAddress"
)
(def-dm-instance Observable
:dm-concept dm-Object
:lexical-root "Observable"
)
(def-dm-instance Applet
:dm-concept dm-Object
:lexical-root "Applet"
)
(def-dm-instance Event
:dm-concept dm-Object
:lexical-root "Event"
)
(def-dm-instance MenuContainer
:dm-concept dm-Object
:lexical-root "MenuContainer"
)
(def-dm-instance Scrollbar
:dm-concept dm-Object
:lexical-root "Scrollbar"
)
(def-dm-instance FileNotFoundException
:dm-concept dm-Object
:lexical-root "FileNotFoundException"
)
(def-dm-instance ClassCastException
:dm-concept dm-Object
:lexical-root "ClassCastException"
)
(def-dm-instance Float
:dm-concept dm-Object
:lexical-root "Float"
)
(def-dm-instance NegativeArraySizeException
:dm-concept dm-Object
:lexical-root "NegativeArraySizeException"
)
```

```
(def-dm-instance Number
  :dm-concept dm-Object
  :lexical-root "Number"
)
(def-dm-instance Observer
  :dm-concept dm-Object
  :lexical-root "Observer"
)
(def-dm-instance AppletStub
  :dm-concept dm-Object
  :lexical-root "AppletStub"
)
(def-dm-instance PixelGrabber
  :dm-concept dm-Object
  :lexical-root "PixelGrabber"
)
(def-dm-instance CheckboxPeer
  :dm-concept dm-Object
  :lexical-root "CheckboxPeer"
)
(def-dm-instance Container
  :dm-concept dm-Object
  :lexical-root "Container"
)
(def-dm-instance GridBagLayoutInfo
  :dm-concept dm-Object
  :lexical-root "GridBagLayoutInfo"
)
(def-dm-instance Image
  :dm-concept dm-Object
  :lexical-root "Image"
)
(def-dm-instance FileDescriptor
  :dm-concept dm-Object
  :lexical-root "FileDescriptor"
)
(def-dm-instance ClassCircularityError
  :dm-concept dm-Object
  :lexical-root "ClassCircularityError"
)
(def-dm-instance IllegalThreadStateException
  :dm-concept dm-Object
  :lexical-root "IllegalThreadStateException"
)
(def-dm-instance InstantiationException
  :dm-concept dm-Object
  :lexical-root "InstantiationException"
)
(def-dm-instance InternalError
  :dm-concept dm-Object
  :lexical-root "InternalError"
)
(def-dm-instance Math
  :dm-concept dm-Object
  :lexical-root "Math"
)
(def-dm-instance Win32Process
  :dm-concept dm-Object
  :lexical-root "Win32Process"
)
(def-dm-instance ContentHandlerFactory
  :dm-concept dm-Object
  :lexical-root "ContentHandlerFactory"
)
(def-dm-instance Enumeration
  :dm-concept dm-Object
  :lexical-root "Enumeration"
```

```
)
(def-dm-instance Properties
 :dm-concept dm-Object
 :lexical-root "Properties"
 )
(def-dm-instance New-option
 :dm-concept menu-item-concept
 :lexical-root "New option"
 :dm-relations ((dm-part-of file-menu))
 )
(def-dm-instance Save_button
 :dm-concept dm-Object
 :lexical-root "Save_button"
 :dm-relations ((dm-part-of Save-file-dialog))
 )
(def-dm-instance Save-file-dialog
 :dm-concept dm-Object
 )
(def-dm-instance ImageConsumer
 :dm-concept dm-Object
 :lexical-root "ImageConsumer"
 )
(def-dm-instance ComponentPeer
 :dm-concept dm-Object
 :lexical-root "ComponentPeer"
 )
(def-dm-instance ListPeer
 :dm-concept dm-Object
 :lexical-root "ListPeer"
 )
(def-dm-instance CheckboxGroup
 :dm-concept dm-Object
 :lexical-root "CheckboxGroup"
 )
(def-dm-instance Rectangle
 :dm-concept dm-Object
 :lexical-root "Rectangle"
 )
(def-dm-instance TextArea
 :dm-concept dm-Object
 :lexical-root "TextArea"
 )
(def-dm-instance TextField
 :dm-concept dm-Object
 :lexical-root "TextField"
 )
(def-dm-instance StreamTokenizer
 :dm-concept dm-Object
 :lexical-root "StreamTokenizer"
 )
(def-dm-instance InstantiationException
 :dm-concept dm-Object
 :lexical-root "InstantiationException"
 )
(def-dm-instance Open-option
 :dm-concept menu-item-concept
 :lexical-root "Open option"
 :dm-relations ((dm-part-of file-menu))
 )
(def-dm-instance DirectColorModel
 :dm-concept dm-Object
 :lexical-root "DirectColorModel"
 )
(def-dm-instance ChoicePeer
 :dm-concept dm-Object
 :lexical-root "ChoicePeer"
 )
```

```
(def-dm-instance DataOutput
  :dm-concept dm-Object
  :lexical-root "DataOutput"
)
(def-dm-instance PushbackInputStream
  :dm-concept dm-Object
  :lexical-root "PushbackInputStream"
)
(def-dm-instance Compiler
  :dm-concept dm-Object
  :lexical-root "Compiler"
)
(def-dm-instance Thread
  :dm-concept dm-Object
  :lexical-root "Thread"
)
(def-dm-instance VectorEnumerator
  :dm-concept dm-Object
  :lexical-root "VectorEnumerator"
)
(def-dm-instance LabelPeer
  :dm-concept dm-Object
  :lexical-root "LabelPeer"
)
(def-dm-instance AWTException
  :dm-concept dm-Object
  :lexical-root "AWTException"
)
(def-dm-instance Frame
  :dm-concept dm-Object
  :lexical-root "Frame"
)
(def-dm-instance MenuBar
  :dm-concept dm-Object
  :lexical-root "MenuBar"
)
(def-dm-instance File
  :dm-concept dm-Object
  :lexical-root "File"
)
(def-dm-instance FileOutputStream
  :dm-concept dm-Object
  :lexical-root "FileOutputStream"
)
(def-dm-instance LineNumberInputStream
  :dm-concept dm-Object
  :lexical-root "LineNumberInputStream"
)
(def-dm-instance PipedInputStream
  :dm-concept dm-Object
  :lexical-root "PipedInputStream"
)
(def-dm-instance SequenceInputStream
  :dm-concept dm-Object
  :lexical-root "SequenceInputStream"
)
(def-dm-instance UTFDataFormatException
  :dm-concept dm-Object
  :lexical-root "UTFDataFormatException"
)
(def-dm-instance Cloneable
  :dm-concept dm-Object
  :lexical-root "Cloneable"
)
(def-dm-instance ExceptionInInitializerError
  :dm-concept dm-Object
  :lexical-root "ExceptionInInitializerError"
```



```
)  
(def-dm-instance Long  
  :dm-concept dm-Object  
  :lexical-root "Long"  
)  
(def-dm-instance NoSuchMethodException  
  :dm-concept dm-Object  
  :lexical-root "NoSuchMethodException"  
)  
(def-dm-instance UnknownError  
  :dm-concept dm-Object  
  :lexical-root "UnknownError"  
)  
(def-dm-instance VirtualMachineError  
  :dm-concept dm-Object  
  :lexical-root "VirtualMachineError"  
)  
(def-dm-instance URL  
  :dm-concept dm-Object  
  :lexical-root "URL"  
)  
(def-dm-instance URLStreamHandlerFactory  
  :dm-concept dm-Object  
  :lexical-root "URLStreamHandlerFactory"  
)  
(def-dm-instance OK_button  
  :dm-concept dm-Object  
  :lexical-root "OK_button"  
  :dm-relations ((dm-part-of Print-Dialog))  
)  
(def-dm-instance FilteredImageSource  
  :dm-concept dm-Object  
  :lexical-root "FilteredImageSource"  
)  
(def-dm-instance ButtonPeer  
  :dm-concept dm-Object  
  :lexical-root "ButtonPeer"  
)  
(def-dm-instance CheckboxMenuItemPeer  
  :dm-concept dm-Object  
  :lexical-root "CheckboxMenuItemPeer"  
)  
(def-dm-instance TextComponentPeer  
  :dm-concept dm-Object  
  :lexical-root "TextComponentPeer"  
)  
(def-dm-instance TextFieldPeer  
  :dm-concept dm-Object  
  :lexical-root "TextFieldPeer"  
)  
(def-dm-instance FlowLayout  
  :dm-concept dm-Object  
  :lexical-root "FlowLayout"  
)  
(def-dm-instance Point  
  :dm-concept dm-Object  
  :lexical-root "Point"  
)  
(def-dm-instance ByteArrayInputStream  
  :dm-concept dm-Object  
  :lexical-root "ByteArrayInputStream"  
)  
(def-dm-instance ByteArrayOutputStream  
  :dm-concept dm-Object  
  :lexical-root "ByteArrayOutputStream"  
)  
(def-dm-instance IllegalMonitorStateException
```

```

:dm-concept dm-Object
:lexical-root "IllegalMonitorStateException"
)
(def-dm-instance IndexOutOfBoundsException
:dm-concept dm-Object
:lexical-root "IndexOutOfBoundsException"
)
(def-dm-instance MalformedURLException
:dm-concept dm-Object
:lexical-root "MalformedURLException"
)
(def-dm-instance SocketOutputStream
:dm-concept dm-Object
:lexical-root "SocketOutputStream"
)
(def-dm-instance URLEncoder
:dm-concept dm-Object
:lexical-root "URLEncoder"
)
(def-dm-instance Dictionary
:dm-concept dm-Object
:lexical-root "Dictionary"
)
(def-dm-instance StringTokenizer
:dm-concept dm-Object
:lexical-root "StringTokenizer"
)
(def-dm-instance Close-option
:dm-concept menu-item-concept
:lexical-root "Close option"
:dm-relations ((dm-part-of file-menu))
)
(def-dm-instance Open-File-Dialog
:dm-concept dm-Object
:lexical-root "Open File Dialog"
:dm-relations ((dm-part-of STE))
)
(def-dm-instance folder-list
:dm-concept dm-Object
:lexical-root "folder list"
:dm-relations ((dm-part-of Save-file-dialog))
)
(def-dm-instance CropImageFilter
:dm-concept dm-Object
:lexical-root "CropImageFilter"
)
(def-dm-instance ImageProducer
:dm-concept dm-Object
:lexical-root "ImageProducer"
)
(def-dm-instance IndexColorModel
:dm-concept dm-Object
:lexical-root "IndexColorModel"
)
(def-dm-instance MenuBarPeer
:dm-concept dm-Object
:lexical-root "MenuBarPeer"
)
(def-dm-instance MenuItemPeer
:dm-concept dm-Object
:lexical-root "MenuItemPeer"
)
(def-dm-instance Font
:dm-concept dm-Object
:lexical-root "Font"
)
(def-dm-instance List

```

```
:dm-concept dm-Object
:lexical-root "List"
)
(def-dm-instance NoClassDefFoundError
:dm-concept dm-Object
:lexical-root "NoClassDefFoundError"
)
(def-dm-instance NumberFormatException
:dm-concept dm-Object
:lexical-root "NumberFormatException"
)
(def-dm-instance Runnable
:dm-concept dm-Object
:lexical-root "Runnable"
)
(def-dm-instance SecurityException
:dm-concept dm-Object
:lexical-root "SecurityException"
)
(def-dm-instance StringIndexOutOfBoundsException
:dm-concept dm-Object
:lexical-root "StringIndexOutOfBoundsException"
)

(def-dm-instance System
:dm-concept dm-Object
:lexical-root "System"
)
(def-dm-instance ProtocolException
:dm-concept dm-Object
:lexical-root "ProtocolException"
)
(def-dm-instance EmptyStackException
:dm-concept dm-Object
:lexical-root "EmptyStackException"
)
(def-dm-instance save-option
:dm-concept menu-item-concept
:lexical-root "save option"
:dm-relations ((dm-part-of file-menu))
)
(def-dm-instance ColorModel
:dm-concept dm-Object
:lexical-root "ColorModel"
)
(def-dm-instance ImageFilter
:dm-concept dm-Object
:lexical-root "ImageFilter"
)
(def-dm-instance ImageObserver
:dm-concept dm-Object
:lexical-root "ImageObserver"
)
(def-dm-instance LayoutManager
:dm-concept dm-Object
:lexical-root "LayoutManager"
)
(def-dm-instance ImageMediaEntry
:dm-concept dm-Object
:lexical-root "ImageMediaEntry"
)
(def-dm-instance Menu
:dm-concept dm-Object
:lexical-root "Menu"
)
(def-dm-instance AbstractMethodError
:dm-concept dm-Object
```

```
:lexical-root "AbstractMethodError"
)
(def-dm-instance NullPointerException
 :dm-concept dm-Object
 :lexical-root "NullPointerException"
)
(def-dm-instance Object
 :dm-concept dm-Object
 :lexical-root "Object"
)
(def-dm-instance Runtime
 :dm-concept dm-Object
 :lexical-root "Runtime"
)
(def-dm-instance Date
 :dm-concept dm-Object
 :lexical-root "Date"
)
(def-dm-instance ObserverList
 :dm-concept dm-Object
 :lexical-root "ObserverList"
)
(def-dm-instance AudioClip
 :dm-concept dm-Object
 :lexical-root "AudioClip"
)
(def-dm-instance GridLayout
 :dm-concept dm-Object
 :lexical-root "GridLayout"
)
(def-dm-instance Panel
 :dm-concept dm-Object
 :lexical-root "Panel"
)
(def-dm-instance BufferedInputStream
 :dm-concept dm-Object
 :lexical-root "BufferedInputStream"
)
(def-dm-instance InterruptedIOException
 :dm-concept dm-Object
 :lexical-root "InterruptedIOException"
)
(def-dm-instance Exception
 :dm-concept dm-Object
 :lexical-root "Exception"
)
(def-dm-instance IllegalArgumentException
 :dm-concept dm-Object
 :lexical-root "IllegalArgumentException"
)
(def-dm-instance Throwable
 :dm-concept dm-Object
 :lexical-root "Throwable"
)
(def-dm-instance ServerSocket
 :dm-concept dm-Object
 :lexical-root "ServerSocket"
)
(def-dm-instance URLStreamHandler
 :dm-concept dm-Object
 :lexical-root "URLStreamHandler"
)
(def-dm-instance NoSuchElementException
 :dm-concept dm-Object
 :lexical-root "NoSuchElementException"
)
(def-dm-instance Open_button
```

```
:dm-concept dm-Object
:lexical-root "Open_button"
:dm-relations ((dm-part-of Open-File-Dialog))
)
(def-dm-instance RGBImageFilter
:dm-concept dm-Object
:lexical-root "RGBImageFilter"
)
(def-dm-instance ScrollbarPeer
:dm-concept dm-Object
:lexical-root "ScrollbarPeer"
)
(def-dm-instance Button
:dm-concept dm-Object
:lexical-root "Button"
)
(def-dm-instance Choice
:dm-concept dm-Object
:lexical-root "Choice"
)
(def-dm-instance Color
:dm-concept dm-Object
:lexical-root "Color"
)
(def-dm-instance Graphics
:dm-concept dm-Object
:lexical-root "Graphics"
)
(def-dm-instance Label
:dm-concept dm-Object
:lexical-root "Label"
)
(def-dm-instance Polygon
:dm-concept dm-Object
:lexical-root "Polygon"
)
(def-dm-instance Window
:dm-concept dm-Object
:lexical-root "Window"
)
(def-dm-instance CloneNotSupportedException
:dm-concept dm-Object
:lexical-root "CloneNotSupportedException"
)
(def-dm-instance NoSuchFieldError
:dm-concept dm-Object
:lexical-root "NoSuchFieldError"
)
(def-dm-instance ThreadGroup
:dm-concept dm-Object
:lexical-root "ThreadGroup"
)
(def-dm-instance SocketImplFactory
:dm-concept dm-Object
:lexical-root "SocketImplFactory"
)
(def-dm-instance system
:dm-concept dm-Object
)
(def-dm-instance new-document
:dm-concept dm-Object
)
(def-dm-instance existing-document
:dm-concept dm-Object
)
(def-dm-instance stePro
:dm-concept dm-Object
```

```
)
(def-dm-instance new
 :dm-concept dm-Object
)
(def-dm-instance folder
 :dm-concept dm-Object
)
(def-dm-instance printer
 :dm-concept dm-Object
)
(def-dm-instance No-of-copy
 :dm-concept dm-Object
)
(def-dm-instance print-range
 :dm-concept dm-Object
)
(def-dm-instance directory
 :dm-concept dm-Object
)
(def-dm-instance create-document
 :dm-concept dm-action
 :lexical-root "create"
 :dm-relations ((dm-actor user)
                (dm-actee document))
)
(def-dm-instance user-choose-New-option-1
 :dm-concept dm-action
 :lexical-root "choose"
 :dm-relations ((dm-actor user)
                (dm-actee New-option))
)
(def-dm-instance system-create-new-document-2
 :dm-concept dm-action
 :lexical-root "create"
 :dm-relations ((dm-actor system)
                (dm-actee new-document))
)
(def-dm-instance save-document
 :dm-concept dm-action
 :lexical-root "save"
 :dm-relations ((dm-actor user)
                (dm-actee document))
)
(def-dm-instance save-new-document
 :dm-concept dm-action
 :lexical-root "save"
 :dm-relations ((dm-actor user)
                (dm-actee new-document))
)
(def-dm-instance user-choose-save-option-3
 :dm-concept dm-action
 :lexical-root "choose"
 :dm-relations ((dm-actor user)
                (dm-actee save-option))
)
(def-dm-instance save-option-show-Save-file-dialog-4
 :dm-concept dm-action
 :lexical-root "show"
 :dm-relations ((dm-actor system)
                (dm-actee Save-file-dialog))
)
(def-dm-instance user-choose-folder-7
 :dm-concept dm-action
 :lexical-root "choose"
 :dm-relations ((dm-actor user)
                (dm-actee folder))
)
```

```
(def-dm-instance user-enter-file-name-8
  :dm-concept dm-action
  :lexical-root "enter"
  :dm-relations ((dm-actor user)
                 (dm-actee file-name))
  )
(def-dm-instance user-click-Save_button-7
  :dm-concept dm-action
  :lexical-root "click"
  :dm-relations ((dm-actor user)
                 (dm-actee Save_button))
  )
(def-dm-instance system-save-document-10
  :dm-concept dm-action
  :lexical-root "save"
  :dm-relations ((dm-actor system)
                 (dm-actee document))
  )
(def-dm-instance save-existing-document
  :dm-concept dm-action
  :lexical-root "save"
  :dm-relations ((dm-actor user)
                 (dm-actee existing-document))
  )
(def-dm-instance user-choose-save-option-9
  :dm-concept dm-action
  :lexical-root "choose"
  :dm-relations ((dm-actor user)
                 (dm-actee save-option))
  )
(def-dm-instance system-save-existing-document-13
  :dm-concept dm-action
  :lexical-root "save"
  :dm-relations ((dm-actor system)
                 (dm-actee existing-document))
  )
(def-dm-instance open-document
  :dm-concept dm-action
  :lexical-root "open"
  :dm-relations ((dm-actor user)
                 (dm-actee document))
  )
(def-dm-instance user-Choose-Open-option-11
  :dm-concept dm-action
  :lexical-root "Choose"
  :dm-relations ((dm-actor user)
                 (dm-actee Open-option))
  )
(def-dm-instance STE-Show-Open-File-Dialog-12
  :dm-concept dm-action
  :lexical-root "Show"
  :dm-relations ((dm-actor system)
                 (dm-actee Open-File-Dialog))
  )
(def-dm-instance user-Choose-directory-17
  :dm-concept dm-action
  :lexical-root "Choose"
  :dm-relations ((dm-actor user)
                 (dm-actee directory))
  )
(def-dm-instance user-Choose-File-18
  :dm-concept dm-action
  :lexical-root "Choose"
  :dm-relations ((dm-actor user)
                 (dm-actee File))
  )
(def-dm-instance user-Click-Open_button-15
```

```
:dm-concept dm-action
:lexical-root "Click"
:dm-relations ((dm-actor user)
              (dm-actee Open_button))
)
(def-dm-instance system-Open-document-20
:dm-concept dm-action
:lexical-root "Open"
:dm-relations ((dm-actor system)
              (dm-actee document))
)
(def-dm-instance print-document
:dm-concept dm-action
:lexical-root "print"
:dm-relations ((dm-actor user)
              (dm-actee document))
)
(def-dm-instance user-Choose-Print-Option-17
:dm-concept dm-action
:lexical-root "Choose"
:dm-relations ((dm-actor user)
              (dm-actee Print-Option))
)
(def-dm-instance Toolkit-show-Print-Dialog-18
:dm-concept dm-action
:lexical-root "show"
:dm-relations ((dm-actor system)
              (dm-actee Print-Dialog))
)
(def-dm-instance user-Select-printer-24
:dm-concept dm-action
:lexical-root "Select"
:dm-relations ((dm-actor user)
              (dm-actee printer))
)
(def-dm-instance user-set-No-of-copy-25
:dm-concept dm-action
:lexical-root "set"
:dm-relations ((dm-actor user)
              (dm-actee No-of-copy))
)
(def-dm-instance user-set-print-range-26
:dm-concept dm-action
:lexical-root "set"
:dm-relations ((dm-actor user)
              (dm-actee print-range))
)
(def-dm-instance user-Click-OK_button-22
:dm-concept dm-action
:lexical-root "Click"
:dm-relations ((dm-actor user)
              (dm-actee OK_button))
)
(def-dm-instance system-print-document-28
:dm-concept dm-action
:lexical-root "print"
:dm-relations ((dm-actor system)
              (dm-actee document))
)
(def-dm-instance Close-document
:dm-concept dm-action
:lexical-root "Close"
:dm-relations ((dm-actor user)
              (dm-actee document))
)
(def-dm-instance user-choose-Close-option-24
:dm-concept dm-action
```



```
:lexical-root "choose"
:dm-relations ((dm-actor user)
              (dm-actee Close-option))
)
(def-dm-instance Close-option-quit-STE-25
:dm-concept dm-action
:lexical-root "quit"
:dm-relations ((dm-actor system)
              (dm-actee STE))
)
(def-dm-instance use-stePro
:dm-concept dm-action
:lexical-root "use"
:dm-relations ((dm-actor user)
              (dm-actee stePro))
)
```