A Support Tool for Writing Multilingual Instructions

Cécile Paris, Keith Vander Linden, Markus Fischer, Anthony Hartley,
Lyn Pemberton, Richard Power and Donia Scott

June, 1995

Information Technology Research Institute Technical Report Series ITRI-95-11

ITRI, Univ. of Brighton, Lewes Road, Brighton BN2 4AT, UK - TEL: +44 273 642900 - FAX: +44 273 606653

A Support Tool for Writing Multilingual Instructions*

Cécile Paris, Keith Vander Linden, Markus Fischer,
Anthony Hartley, Lyn Pemberton, Richard Power, and Donia Scott
Information Technology Research Institute
University of Brighton
Lewes Road
Brighton BN2 4AT, UK
email: {first-name.last-name}@itri.bton.ac.uk

Abstract

Multilingual instructions generation has been the object of many studies recently, motivated by the increased need to produce multilingual manuals coupled with the cost of technical writing and translating. These studies concentrate on the *automatic* generation of instructions, leaving technical writers out of the loop. In many cases, however, it is not possible to dispense with human intervention entirely, for at least two reasons. First, the system must be provided with a semantic knowledge base from which the instructions can be generated. Second, it is the technical writers who have the expertise necessary for producing instructions appropriate for a specific product or company, and it is not necessarily an easy task to make this expertise available to a system. The results of a requirement analysis study confirm the view that the most useful tool is not a stand-alone writing tool but rather one that supports technical writers in their task. In this paper, we describe such a support tool, which we developed based on the results of our user requirement analysis.

1 Introduction

The automatic generation of instructional texts has been the object of many studies recently, motivated by the increased need to produce manuals coupled with the cost of technical writing, the time required to produce documentation, and the potential flexibility offered by the automatic generation of instructions. Researchers have concentrated on designing methods for integrating graphics and text, e.g., [Wahlster *et al.*, 1993; Feiner and McKeown, 1990], and for tailoring instructions to the user's level of expertise, e.g., [Peter and Rösner, 1994]. At a more linguistic level of concern, others have studied various ways of realising purpose expressions in English, e.g., [Vander Linden, 1993], and of generating appropriate referring expressions, e.g., [Dale, 1992]. More recently, there has been an emphasis on the generation of multilingual instructions, e.g., [Rösner and Stede, 1991; Kosseim and Lapalme, 1994]. The latter is not entirely surprising since multilingual manuals are important not only for European manufacturers, who are required to produce manuals in the language of the end-user, but also for other multinational companies, whose overseas sales are reported to constitute over half of their total sales. Multilingual generation is also more appealing than monolingual generation followed by translation because (1) the texts can be generated in several languages simultaneously, rather than waiting for the translation process, (2) the underlying knowledge being expressed in monolingual instructions can be used to generate instructions in different languages, and (3) generating directly from the underlying knowledge base can produce more natural texts as the output text is not constrained by a source text.

Most of the prototypes developed so far are intended to be used as stand-alone tools, leaving the technical writers out of the loop. They assume that an underlying knowledge base containing all the information necessary to produce instructions (or documentation) is already available to the generation system, or can be easily obtained. However, this is unlikely in the near future. Generating texts from an underlying knowledge base is indeed a very knowledge intensive task. Furthermore, this knowledge base must contain user-oriented information, as user-oriented documentation is recognised to be more effective than product-oriented documentation. Such information concerns the goals of the user and the ways in which the product can be used to achieve these goals. It is not always available from the design specification of the product. Consequently, it needs to be entered by hand, a task which is neither simple nor straightforward. In addition, most companies have specific 'house-styles'; these are not always set out in the form of detailed and explicit rules but tend to be tacitly learnt by technical writers. For all these reasons, the expertise of technical writers might not easily be embodied in a computer system.

Available evidence thus suggests that it would be desirable at this point to provide a *support drafting tool* (as opposed to a stand-alone writing tool), recognising that it would need to be integrated into the technical writers' wider working practices. Such a tool would not be intended to bypass the human authors, but would rather help them in their task by automat-

*This work is partially supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant J19221, by BC/DAAD ARC Project 293, and by the Commission of the European Union Grant LRE-62009.

ically generating drafts in several languages. Our first step towards developing such a tool was to conduct a user requirements analysis, identifying the wider environment in which the proposed tool would be used. Based on the results of our study, we have developed DRAFTER, a drafting tool intended to be used by technical writers in producing multilingual instructions. Our current domain of application is software manuals. In this paper, we briefly describe the results of our user requirements analysis, present the resulting DRAFTER architecture, and, finally, illustrate the system with an example of how a technical writer might work with DRAFTER. We give examples for creating multilingual instructions for the OpenWindows Calendar Manager.

2 The User Requirements Analysis

To study the technical writers' needs, we conducted interviews with technical authors (mostly software documentation specialists), both in-house and freelance. The discussions covered a range of issues, from overarching constraints of time and budget to the areas of the job perceived as interesting, difficult, etc. The authors explained the succession and timing of the processes in the documentation task, and their coordination and monitoring. They also described the form in which the evolving document is represented, the sources and channels of information, and the tools and resources used. While lack of space prevents us describing this study in detail, we present the main findings. (See [Power *et al.*, 1994] for details.)

2.1 The Technical Writer's Tasks

Interestingly, we found that technical writers spend little time working on new texts. The greater part of their work is updating existing documents. The notion of reuse is thus quite important. Five main tasks emerged from our discussions: knowledge acquisition, document planning, composition, validation and maintenance. These tasks are of course interleaved in the production process.

Knowledge Acquisition: Technical writers have to work in close collaboration with designers and engineers to gather and structure the information about the product or procedure they need to document. This is done by consulting the designers, reading the comments in actual code, and experimenting with prototype versions of the new product. The task of knowledge acquisition is very difficult, and it occupies as much time as the writing proper. The main burden of knowledge acquisition is borne by the authors when they first encounter the product, at which time they must construct a mental model of the product *from the end-user's perspective*. Authors acknowledge that a formal record of this model would be useful in documenting subsequent modifications of the product by the same writer or by colleagues. It could also be used when the same procedure needs to be explained again but in a different context, such as a different part of a manual. However, such a model typically is not created explicitly.

Document Planning: Writers need to establish the overall structure and purpose of the document. It is widely recognised that a task-oriented viewpoint is more communicatively effective than a product-oriented one. Taking such a viewpoint, a typical structure for an instructional manual is to have a short (about a page) chapter for each self-contained task, broken down into operations of about six or seven lines.

Composition: Technical writers typically write several drafts of a document. They aim to be effective communicators, avoiding jargon, and conveying their message in clear and concise terms. By training and experience, authors become conversant with general standards of technical writing (e.g., the convention for distinguishing notes from warnings).

Most companies also have style guides, formal or informal, which further constrain the authors. A style guide might, for example, recommend or even prescribe the use of specific constructions and terminology. A rigorously formalised style guide imposes a controlled language, which is sometimes difficult and time-consuming to master.

Validation: Quality assurance mechanisms range from informal proofreading by colleagues to formal reviews by committee. Some organisations require their writers to submit their output for critiquing by an automatic terminology and grammar checker. This process may be repeated over several drafts, depending on the time available for preparation.

Maintenance: A significant proportion of a writer's time is spent on maintaining documentation when changes are made to existing products.

2.2 Desiderata for a Support Writing Tool

From our discussions with technical writers and our understanding of their task, the following desiderata for a Support Writing Tool emerged:

- *Support for knowledge reuse by helping authors create a formal model of the knowledge they acquire* — As a lot of time is spent in knowledge acquisition and knowledge is reused frequently, authors indicated that they would welcome a tool that would help them formalise their knowledge about the product, allowing them to structure it in a consistent manner, examine it later, and share it with colleagues.
- *Production of alternative formulations when possible* — As there are often several ways to express a set of instructions, the authors expressed a desire to have several drafts produced, from which they can choose the most appropriate one.
- *Availability of early drafts produced simultaneously in several languages* — The possibility of producing drafts as soon as some mental model of the task is formalised would help authors find out what underlying knowledge is still missing to provide good instructions. It also speeds up the whole process.
- *Propagation of changes throughout document and languages* — When a change is required, authors would like to make the change only once. Modifying the same text in several places is a tedious task, and it jeopardises consistency.

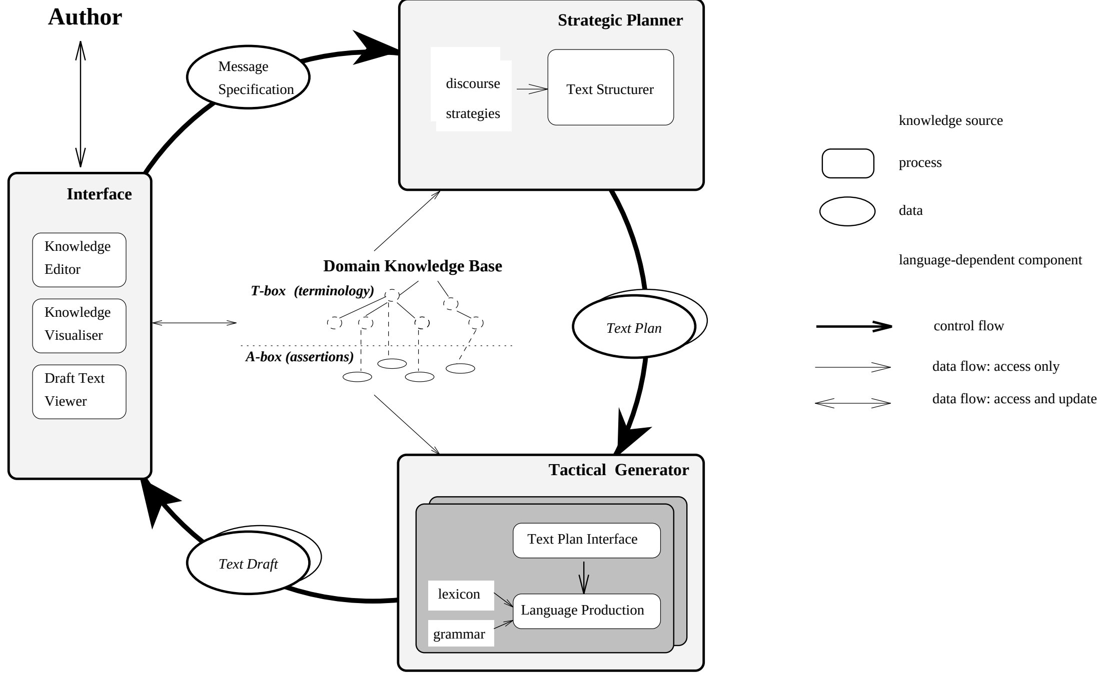

Figure 1: Block Diagram of the Architecture

- *Support for accurate and consistent terminology* — Technical terms need to be employed consistently within and between documents, even if these are produced by several authors. Furthermore, there are often constraints imposed by the company. Authors would welcome a tool to help them learn these constraints and ensure that they are applied systematically.
- *Retain creative satisfaction of technical writing*. — A tool to support writers should automate those aspects writers find tedious, such as revision and some of the rudimentary aspects of composition (e.g., consistent terminology and syntax), and leave to the authors the tasks they find interesting and challenging, such as structuring knowledge and expressing ideas.

3 DRAFTER

Based on the user requirements analysis described above, we have designed and implemented DRAFTER, a software manual drafting tool for English and French. The overall architecture of DRAFTER is shown in Figure 1. It contains three processing modules, which form two main support tools:

- **An interface for the technical writer**. This allows authors to specify formally the procedures necessary for the user to achieve their goals, thus supporting user-oriented instructions. It also allows them to control the drafting process.
- **The drafting tool**. This comprises two major components: the strategic planner and the tactical generator. The strategic planner determines the content and structure of the text, and the tactical generator performs the realisation of the sentences. The result is English and French drafts of the instructions for the procedures defined so far by the author using the interface.

Underlying the processing components is a *domain model*, i.e., the main repository of information about the domain.

3.1 The Domain Model

The Domain Model, implemented in LOOM [MacGregor, 1988], is a collection of entities representing the information commonly occurring in the software domain. These entities include actions, states, objects, and a set of relations between them. This knowledge, derived from a study of a multilingual corpus of software manuals, is treated as language-independent, an important requirement for multilingual generation. It is hierarchically organised, using the Upper Model [Bateman *et al.*, 1990][1] as its root, and maintaining three further levels of structure corresponding to: (1) the concepts and relations general to all instructions; (2) those general only to software interfaces; and (3) those specific to the chosen software application domain.

The technical authors use the concepts and relations in the domain model to specify the procedures appropriate for the particular software system being documented. These procedures are represented as hierarchical configurations of actions, states, and objects, in a manner common in traditional plan structures [Fikes and Nilsson, 1971; Sacerdoti, 1977]. Because our interest is in the expression of these procedures rather than their execution, our representation allows authors to include information not typically found

[1] The Upper Model is an ontology of distinctions employed to determine how to express the concepts linguistically.

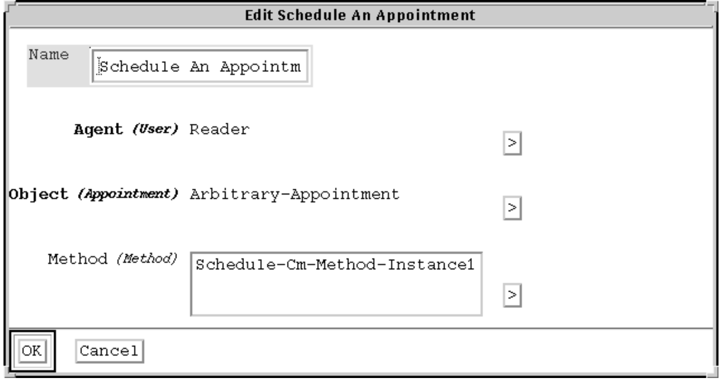

Figure 2: The Schedule Action

in traditional plan representations,[2] such as: user-oriented motivational goals, helpful side-effects (often included in written instructions), and explicit specifications of alternative plans, conditions, and options.

3.2 The Interface

DRAFTER provides an interface for creating and maintaining a formal record of the knowledge authors learn during the knowledge acquisition task. It allows them to specify the conceptual knowledge required for the task to be documented, and information important from the user's point of view. It exploits the domain model to provide guidance and structure. The interface encompasses the following functions:

- Construction and maintenance of the assertional knowledge base containing the descriptions of functions the user can perform with the software being documented.
- Visualisation of aspects of the knowledge base.
- Viewing and editing the automatically generated drafts.

All these functions are invoked from menus, interface widgets and other mouse-sensitive objects, in a style common to systems such as Motif.

Constructing the Assertional Knowledge Base

Authors can enter or change the information to be included in the documentation for the software system under consideration using the *Knowledge Editor*. This editor relies on the presence of a domain model constraining and structuring the data to be entered.

Before making a new assertion, the author chooses the appropriate node in the domain model to which the new information will be subordinated. This is done by navigating through a sub-graph of the domain model via cascading menus. These contain at each point the types of concepts available in the domain. Once a concept is chosen, the interface determines the properties to be specified, and dynamically constructs a dialog box for the author to fill in. The interface also allows the writer to create the concepts 'on the fly', thus providing a flexible and un-constraining interaction style. Figure 2 shows a sample dialog box for entering information about the action of *scheduling appointments* in the OpenWindows Calendar Manager. The *agent*, *object* and *method* slots are derived from the domain model.[3] The interface also provides defaults, if these are present in the model (e.g., `Reader` as the default agent in Figure 2). For each slot, the author can choose to keep the default value when present, or to specify a new value, again guided by the interface. In the example, the author kept the default values for the *agent* and *object*, and defined one method: *Schedule-Cm-Method-Instance1* for achieving the action. It is in the method that the author specifies the sub-actions that a user performs. The method also includes preconditions and side-effects when necessary.

DRAFTER also provides a *Window Description Facility* allowing the author to annotate images of application windows with semantic information. These images are obtained either by scanning in from hardcopy or by taking an electronic snapshot directly from the screen.[4] This allows buttons or other interface gadgets that need to be mentioned in the instructions to be identified easily and later referred to when defining procedures (they are automatically asserted in the knowledge base).

Visualising the Knowledge Base

To prevent the author from losing orientation, DRAFTER provides the *Knowledge Visualiser*, which keeps the author informed about the status and content of the knowledge base. It can be used to view several aspects or areas of the knowledge base in tabular or graphical form. The author may, for example, view the relationship between actions, methods and sub-actions, as illustrated in Figure 3. There, we see the method the author has defined for the action *Schedule an Appointment* (shown on the left). The method, *Schedule-CM-Method-Instance1*, is shown to have four sub-actions: type the description of the appointment, choose the start time of the appointment, choose the end time of the appointment, and click on the insert button. The two small squares under the name of the method indicate that this method has a precondition and a side-effect, namely in this case: the CM Editor

[2] For more detail, see [Delin *et al.*, 1994].

[3] The bold face for *agent* and *object* indicate that these slots are *required*, while *method* is optional.

[4] It is also conceivable that such images be extracted automatically by consulting appropriate resources, e.g., output files of interface building tools.

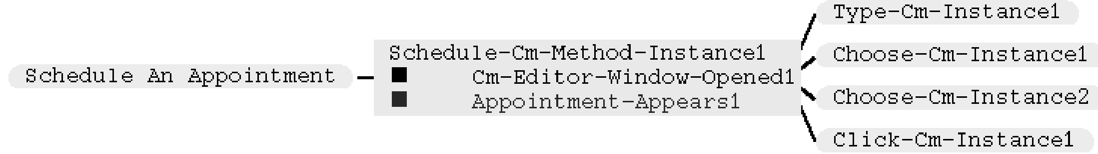

Figure 3: The Scheduling Procedure

Window must be opened, and the appointment just defined appears in a list of appointments. These can be viewed (and updated) by clicking on the rectangle representing the method.

The Knowledge Visualiser is fully integrated with the facility to construct and update the knowledge base, so that the writer can trigger interface functions such as editing or generating on every constituent of a visualisation, allowing a parallel development of knowledge base and natural language text.

The Draft Text Viewer

After the procedures and objects have been specified, text in French and English can be generated. The text is mouse-sensitive, allowing the author to access the knowledge base entry for selected part of the text. In this way, the author can modify the underlying knowledge base while working from the text. In some cases the writer will decide to modify the generated text rather than the underlying knowledge. For this purpose, a text editor is currently provided. We intend to develop a more sophisticated tool that will constrain and record this post-editing.

3.3 The Strategic Planner

We use an existing text planning system that constructs text by explicitly reasoning about the communicative goal to be achieved, as well as how the goals relate to each other rhetorically to form a coherent text [Moore and Paris, 1993]. Given a communicative goal, the system finds from its library of discourse strategies (or plans)[5] a plan capable of achieving this goal. Plans typically post further sub-goals to be satisfied. These are expanded, and planning continues until primitive speech acts are achieved. The result of this planning process is a *discourse tree*, in which the nodes represent goals at various levels of abstraction (the root being the initial goal, and the leaves the primitive realisation statements, speech acts such as `INFORM`). The discourse tree also includes *coherence relations* [Mann and Thompson, 1988] indicating how the various portions of the text are related rhetorically.

Some of the constraints imposed by writing standards or house style concerning the structure of a manual can be embodied in the discourse strategies, and, by associating several strategies for the same discourse goal, it is possible to provide alternative drafts, as desired by the authors.

3.4 The Tactical Generator

We employ the KPML environment [Bateman, 1994] for our tactical generators. We have extended its coverage in English to generate the types of sentences found in instructional manuals, and are using the flexible environment it provides to develop a French grammar. KPML, a descendent of Penman [Mann, 1983], is based on Systemic Functional Linguistics (SFL) [Halliday, 1978], expressing its grammar in terms of system networks. The rules dictated by the general standards of technical writing which are formally defined can be added to the linguistic resources available to constrain the general potential of the generators.

The output of the strategic planner is passed through the text plan interface which constructs statements in the Sentence Plan Language (SPL) [Kasper, 1989], KPML's input language. This interface takes into account the discourse tree and the potentially different forms of expression appropriate in English and French.[6]

4 Working with DRAFTER: An Example

Suppose the author wishes to generate instructions for scheduling an appointment with the OpenWindows Calendar Manager. He or she must specify the exact steps a user must carry out. This might be done by defining all the objects the user will see in the *Appointment Editor Window* (using the Drafter Window Description Interface), and specifying the method proper. The author can then ask for the instructions to be drafted for this action. At this point, DRAFTER calls the strategic planner with the discourse goal: make the user competent to perform the action of scheduling an appointment. The strategic planner builds a discourse tree, which provides the deep representation of the text to be generated. This tree is passed through the text plan interface, and then to KPML. The generator is called using the linguistic resources for English and for French. The generated drafts are shown in Figure 4.

In these texts, we see that the top-level action (or the goal of the method) is given as a title to the series of steps. Then, the steps to be performed to achieve this goal are given. Recall that the method for scheduling an appointment had four sub-steps and a precondition (as shown in Figure 3). The strategic planner chose to express the precondition as the first step to perform when scheduling an appointment. But instead of simply stating: 'Open the CM Editor Window', it told the user how to achieve this step: 'Choose Edit . . . '. It also described the result of achieving it: 'The CM Window appears'. The four sub-steps for scheduling an appointment are then listed. Finally, the side-effect is explicitly provided ('The appointment appears in the list . . . '), so that the user can monitor whether the action has been performed correctly.

[5] It is possible that there will be different plans for the different languages.

[6] We are currently working to allow the specification of the input to the generator to be at a level of abstraction such that this interface would built the same structure regardless of the language, and the differences in syntactic realisations would be dealt with within the tactical generator proper.

French
Entrée d'un rendez-vous:
Choisir Edition→Rendez-vous pour afficher la fenêtre Edition de Rendez-Vous CM.
La fenêtre Edition de Rendez-Vous CM apparaît.
Introduire la description du rendez-vous.
Choisir l'heure de début, ensuite choisir l'heure de fin.
Cliquer SELECTIONNER sur le bouton Insertion.
Le rendez-vous apparaît dans la liste dans la fenêtre Edition de Rendez-vous CM.

English
To schedule an appointment:
Choose Edit→Appointment to display the CM Appointment Editor window.
The CM Appointment Editor Window appears.
Type the description of the appointment.
Choose the start time, then choose the end time.
Click SELECT on Insert.
The appointment appears in the list in the CM Appointment Editor window.

Figure 4: Generated drafts of the instructions in French and English

French
Entrée d'un rendez-vous:
Il faut être dans la fenêtre Edition de Rendez-Vous CM.
Choisir Edition→Rendez-vous pour afficher la fenêtre Edition de Rendez-Vous CM.
Dans la fenêtre Edition de Rendez-Vous CM :
Introduire la description du rendez-vous.
. . .

English
To schedule an appointment:
You must be in the CM Appointment Editor Window.
Choose Edit→Appointment to display the CM Appointment Editor window.
In the CM Appointment Editor Window:
Type the description of the appointment.
. . .

Figure 5: Alternative texts

Some differences between French and English are apparent in these texts. First, the title is expressed as a to-infinitive clause in English, and as a nominalisation of the goal verb in French (literally: 'The scheduling of an appointment'). Another difference appears in the way to which the interface objects are being referred. While, in English, it is acceptable to refer to a button simply by its name (i.e., click on 'insert'), this is less common in French, and the type of the object, here 'Bouton', has to be explicitly stated. Finally, the user is being addressed in a slightly different manner: while the English employs a direct address (the imperative), the French uses a more distant form of address: the infinitive form. Such variations, although not impossible in a translation paradigm, are less likely there than they are in a multilingual generation paradigm.

4.1 Producing Alternative Texts

One of the requirements for a support drafting tool was the flexibility to produce several texts corresponding to the same set of instructions. In our approach, the strategic planner can be told to re-plan the text. If other discourse strategies are available, they will be employed, and a different discourse tree will be produced. This new tree is again passed to the text plan interface and KPML. For example, there are several strategies available to express a precondition: one can either include it as if it was the first sub-action to achieve a goal (as was done in the drafts shown in Figure 4), or, alternatively, one can express it explicitly as a precondition. The modified text resulting from using this alternative strategy are shown in Figure 5. Note also that, in French, in order to be consistent with the distant form of address, the indirect formula 'il faut' ('one must') was chosen instead of the more direct: 'vous devez' ('you must').

4.2 Update and Re-use

Suppose the author decides that the specification of the procedure for scheduling an appointment is not appropriate, and that there is no need for a precondition. Instead of having to update the instructions themselves (and risk inconsistency between the two texts), he or she can simply change the underlying specification for this procedure, removing the precondition. The change made, DRAFTER can be asked to re-generate the instructions in the two languages. These will automatically reflect the change, and the precondition will be absent from both texts. The rest remains the same.

5 Summary

In this paper we have discussed DRAFTER, a tool we have developed to support the technical author in the drafting of multilingual software manuals, based on a user requirements analysis. We described the facilities which allow the author first to specify the procedural knowledge necessary for using the software, and then to generate drafts in English and French, and illustrated them with an example. In our future work, we will be developing additional tools to provide a richer drafting environment and evaluating the system with professional technical authors.

References

[Bateman *et al.*, 1990] Bateman, John A.; Kasper, Robert T.; Moore, Johanna D.; and Whitney, Richard 1990. A general organization of knowledge for natural language processing: The Penman Upper Model. Technical report, USC/ISI.

[Bateman, 1994] Bateman, John A. 1994. KPML: The KOMET-Penman (Multilingual) Development Environment. Technical report, Institut für Integrierte Publikations- und Informationssysteme (IPSI), GMD, Darmstadt. Release 0.6.

[Dale, 1992] Dale, Robert 1992. *Generating Referring Expressions: Constructing Descriptions in a Domain of Objects and Processes*. MIT Press, Cambridge, MA.

[Delin *et al.*, 1994] Delin, Judy; Hartley, Anthony; Paris, Cécile; Scott, Donia; and Vander Linden, Keith 1994. Expressing procedural relationships in multilingual instructions. In *Proceedings of the Seventh International Workshop on Natural Language Generation,* Kennebunkport, ME, 21–24 June 1994.

[Feiner and McKeown, 1990] Feiner, Steve and McKeown, Kathleen 1990. Coordinating Text and Graphics in Explanation Generation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, Massachusetts. 442–449.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, Nils 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

[Halliday, 1978] Halliday, Michael A. K. 1978. *Language as a Social Semiotic: The Social Interpretation of Language and Meaning*. University Park Press, Baltimore.

[Kasper, 1989] Kasper, Robert T. 1989. A flexible interface for linking applications to Penman's sentence generator. In *Proceedings of the DARPA Speech and Natural Language Workshop* Philadelphia.

[Kosseim and Lapalme, 1994] Kosseim, Leila and Lapalme, Guy 1994. Content and rhetorical status selection in instructional texts. In *Proceedings of the Seventh International Workshop on Natural Language Generation,* Kennebunkport, ME, 21–24 June 1994. 53–60.

[MacGregor, 1988] MacGregor, Robert 1988. A Deductive Pattern Matcher. In *Proceedings of the 1988 Conference on Artificial Intelligence*, St Paul, Mn. American Association of Artificial Intelligence.

[Mann and Thompson, 1988] Mann, William C. and Thompson, Sandra A. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text: An Interdisciplinary Journal for the Study of Text* 8(2):243–281.

[Mann, 1983] Mann, William C. 1983. An overview of the PENMAN text generation system. In *Proceedings of the National Conference on Artificial Intelligence*. AAAI. 261–265. Also appears as USC/Information Sciences Institute, RR-83-114.

[Moore and Paris, 1993] Moore, Johanna D. and Paris, Cécile L. 1993. Planning text for advisory dialogues: Capturing intentional and rhetorical information. *Computational Linguistics* 19(4):651–694.

[Peter and Rösner, 1994] Peter, Gerhard and Rösner, Dietmar 1994. User-Model-Driven Generation of Instructions. *User Modeling and User Adapted Interaction*.

[Power *et al.*, 1994] Power, Richard; Pemberton, Lyn; Hartley, Anthony; and Gorman, Louise 1994. User requirements analysis.

[Rösner and Stede, 1991] Rösner, Dietmar and Stede, Manfred 1991. Towards the Automatic Production of Multilingual Technical Documents. Technical Report FAW-R-91022, Research Institute for Applied Knowledge Processing (FAW), Ulm, Germany.

[Sacerdoti, 1977] Sacerdoti, Earl D. 1977. *A Structure for Plans and Behavior*. Elsevier, New York.

[Vander Linden, 1993] Vander Linden, Keith 1993. *Speaking of Actions: Choosing Rhetorical Status and Grammatical Form in Instructional Text Generation*. Ph.D. Dissertation, University of Colorado. Available as Technical Report CU-CS-654-93.

[Wahlster *et al.*, 1993] Wahlster, Wolfgang; André, Elisabeth; Finkler, Wolfgang; Profitlich, Hans-Jürgen; and Rist, Thomas 1993. Plan-based integration of natural language and graphic generation. *Artificial Intelligence* 63:387–427.