Terminology Extraction Techniques for Improving Traceability from Formal Models to Textual Requirements

Farid Cerbah[1] and Jérôme Euzenat[2]

[1] Dassault Aviation - DPR/DESA - 78, quai Marcel Dassault
92552 cedex 300 Saint-Cloud - France
farid.cerbah@dassault-aviation.fr

[2] Inria Rhône-Alpes - 655, avenue de l'Europe
38330 Monbonnot St Martin - France
Jerome.Euzenat@inrialpes.fr – http://www.inrialpes.fr/exmo/

Abstract. This article deals with traceability in sotfware engineering. More precisely, we concentrate on the role of terminological knowledge in the mapping between (informal) textual requirements and (formal) object models. We show that terminological knowledge facilitates the production of traceability links, provided that language processing technologies allow to elaborate semi-automatically the required terminological resources. The presented system is one step towards incremental formalization from textual knowledge.

1 Introduction

Modern information systems tend to integrate textual knowledge and formal knowledge in common repositories. The informal is richer and familiar to any user while the formal is more precise and necessary to the computer. It is recognized that linking formal knowledge to informal knowledge has several benefits including, (1) establishing the context for formalized knowledge and documenting it, and (2) providing a natural way to browse through formalized knowledge. Two significant examples can be pointed out:

– **Product Data Management.** Design and production data are formalized in product trees. This formalization improves data consistency and evolutivity. The components of the product tree are related to documents, such as maintenance manuals or manufacturing notices. Connecting formal models to informal sources guarantees a better synchronization between technical data and documents.
– **Software Engineering.** Formal models, and more particularly object oriented models are widely used in software development. In this context, textual knowledge represents specification and design documents. These informal sources are used as a basis for building the formal models.

Several works focused on the advantages of using a corpus-based terminology for supporting formal knowledge acquisition [4], [1], [2]. These contributions emphasize the central role of terminological resources in the mapping between informal text sources and formal knowledge bases. In the same spirit, the present work uses terminology software support for generation and management of traceability links between initial software requirements and formal object representations resulting from the modeling processes. We describe a fully implemented system that provides high-level hypertext generation, browsing and model generation facilities. From a more technical viewpoint, we introduce an original XML based model for integrating software components.
The rest of the paper is organized as follows. Section 2 introduces the main concepts of our approach and the basic tasks that should be performed by a user support tool which takes advantage of terminological knowledge for improving traceability. Section 3 gives a detailed and illustrated description of the implemented system. Finally, section 4 briefly compares our contribution to related works and the conclusion provides some directions for further research.

2 Principles

2.1 Traceability in Software Engineering

In a software development process, design and implementation decisions should be "traceable", in the sense that it should be possible to find out the requirements impacted, directly or indirectly, by the decisions. This mapping is useful in many respects:

- It helps to ensure exhaustiveness: By following traceability links, the user or a program can easily identify the requirements which are not satisfied by the software.
- It facilitates the propagation of changes: At any time in the development process, traceability information allows to find out the elements impacted by changes (upstream and downstream). For instance, the user can evaluate the incidence on the software design and implementation of a late change in the initial customer requirements.
- When traceability is established with hyperlinks, the browsing capabilities provided by design support tools are increased.

In an object-oriented framework, many traceability links aim at relating textual fragments of the documents in natural language and model fragments. Putting on these links manually is a tedious and time consuming task and current tools for requirement analysis provide no significant help for doing that job.

2.2 The Role of Terminological Resources

In many information systems where both textual knowledge and formal knowledge are involved to describe related concepts, terminology can play an intermediate role. As mentioned earlier, previous works in the fields of knowledge

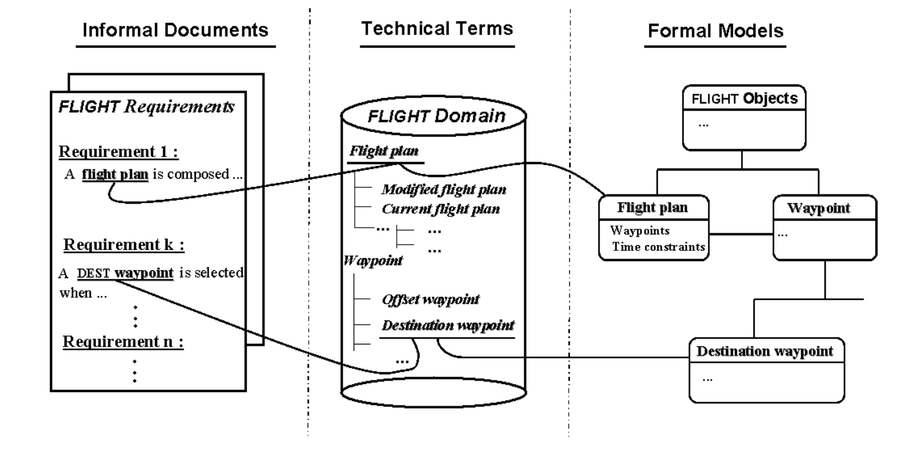

Fig. 1. Using terminological items to link textual requirements and object models

acquisition and natural language processing have shown that terminological resources extracted from corpora can help the incremental formalization processes from texts to formal models.
There exist other demonstrative examples in related domains, such as product data management and software engineering.
For example, in the DOCSTEP project [8], which deals with product data management, terminological resources are used to connect multilingual technical documentation and items of product trees. Hyperlinks are established between term occurrences in documents and corresponding objects in product trees.
In software engineering, the role of terminological knowledge in the modeling process has often been pointed out [15, 10, 3]. One of the first step in the modeling process consists of a systematic identification of the technical terms (simple and compound nouns) in the documents, namely the terminology used to describe the problem. Some of these technical terms represent concepts which will be subsequently introduced in the formal models. These terms can be seen as an intermediary level between the textual requirements and the formal models. (see figure 1).

2.3 Functional View of a System that Exploits Terminology

A system that takes advantage of terminological resources may involve techniques pertaining to several technological areas, and particularly natural language processing, information retrieval and knowledge management:

Terminology Extraction. In technical domains, many precise and highly relevant concepts are linguistically represented by compound nouns. The multiword nature of the technical terms facilitates their automatic identification in texts. Relevant multi-word terms can be easily identified with high accuracy using partial syntactic analysis [4], [11] or statistical processing [6] (or even both paradigms [7]). Terminology extraction techniques are used to automatically build term hierarchies that will play the intermediate role between documents and models.

Document and Model Indexing. The technical terms are used for indexing text fragments in the documents. Fine grained indexing, i.e paragraph level indexing, is required while most indexing systems used in information retrieval work at the document level. Besides, most descriptors used in this kind of indexing are multi-word phrases. The terms are also used for indexing the model fragments (classes, attributes . . .).

Hyperlink Generation. The terminology driven indexing of both texts and objects with the same terminology is the basis of the hyperlink generation mechanisms. Futhermore, hyperlink generation mechanisms should be controlled interactively, in the sense that the user should be able to exclude automatically generated links or add links that have not been proposed by the system.

Model Generation. It is quite common that the concept hierarchies mirror the term hierarchies found in the documents. This property can be used to generate model skeletons which will be completed manually.

These features are implemented in the system presented in the next section.

3 A User Support Tool for Improving Traceability

The implemented system consists of two components, XTerm and Troeps. XTerm deals with the document management and linguistic processing functions, more particularly terminological extraction and document indexing. Troeps deals with model management and model indexing. The model generation function is spread over both components.

3.1 XTerm

XTerm [5] is a natural language processing tool that provides two services to end users:

- Terminology acquisition from documents. It analyzes a French or English technical documentation in order to build a hierarchy of potential technical terms. The user can explore and filter the extracted data via a graphical interface.
- Terminology-centred hypertext navigation. XTerm can be seen as a hypertext browser. The extracted terms are systematically linked to their textual contexts in the documents. The user can easily access the textual fragments containing term occurrences.

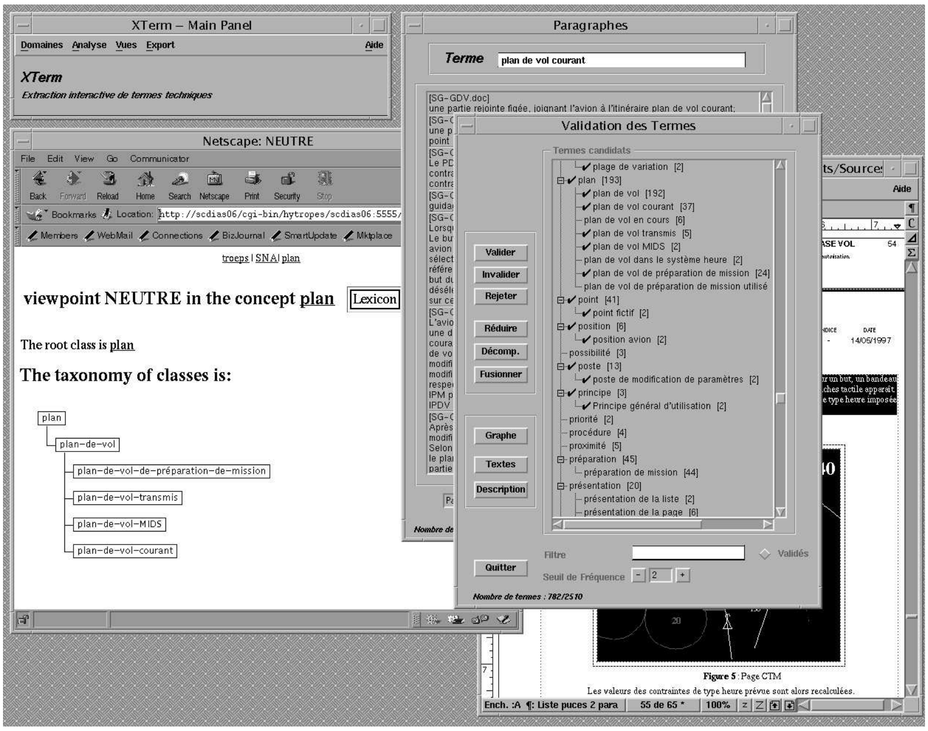

Fig. 2. The integrated system based on XTerm and Troeps.

XTerm is made of four components:

Document Manager. This component provides textual data to the linguistic components. It scans all document building blocks (paragraphs, titles, figures, notes) in order to extract the text fragments. The extracted units are then prepared for linguistic processing.
Additionally, the document manager provides the mechanisms for indexing and hyperlink generation from technical terms to document fragments. Hyperlink generation is a selective process: To avoid overgeneration, the initial set of links systematically established by the system can be reduced by the user.

Part of Speech Tagger. The word sequences provided by the document manager are processed by a tagger based on the Multex morphological parser [14]. POS tagging starts with a morphological analysis step which assigns to each word its possible morphological realisations. Then, contextual desambiguation rules are applied to choose a unique realization for each word. At the end of this process, each word is unambigeously tagged.

Term Extractor. As mentioned in section 2.3, the morpho-syntactical structure of technical terms follows quite regular formation rules which represent

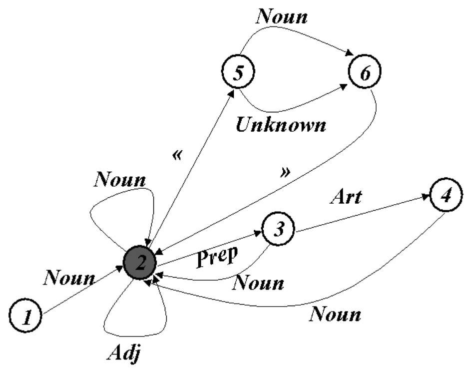

Fig. 3. a term extraction automaton

a kind of local grammar. For instance, many French terms can be captured with the pattern "*Noun Preposition (Article) Noun*". Such patterns can be formalized with finite state automata, where transition crossing conditions are expressed in terms of morphological properties. The figure 3 gives an example of a simplified automaton (state 2 is the unique final state).

To identify the potential terms, the automata is applied on the tagged word sequences provided by the POS tagger. A new potential term is recognized each time a final state is reached. During this step, the extracted terms are organized hierarchically. For example, the term "*flight plan*" ("*plan de vol*" in figure 2) will have the term "*plan*" as parent and "*modified flight plan*" as a child in the hierarchy.

Actually, term extraction with automata is just the first filtering step of the overall process. The candidate set obtained after this step is still too large. Additional filtering mechanisms are involved to reduce that set. In particular, grouping rules are used to identify term variants. For instance, in French technical texts, prepositions and articles are often omitted for the sake of concision (the term "*page des buts*" can occur in the elided form: "*page buts*)"[1]. Term variants are systematically conflated into a single node in the term hierarchy.

Management/Browsing Component. This component ensures the basic term management functionalities (editing, search, validation). XTerm is highly interactive. Many browsing facilities are provided to facilitate the manipulation of large data sets (extracted terms + text fragments). XTerm can be used as an access tool to documentation repositories.

[1] Whose English literal translations are respectively: "*page of the waypoints*" and "*page waypoints*". A plausible equivalent term in English could be "*Waypoint page*".

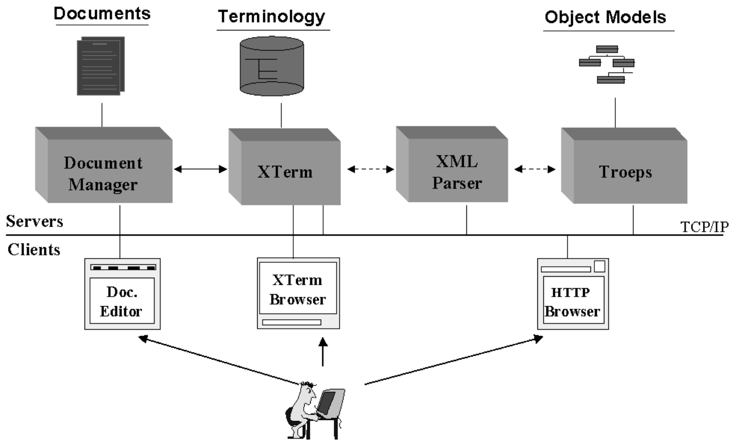

Fig. 4. System architecture

3.2 Troeps

Troeps [12, 16] is an object-based knowledge representation system, i.e. a knowledge representation system inspired from both frame-based languages and object-oriented programming languages. It is used here for expressing the models.
An object is a set of field-value pairs associated to an identifier. The value of a field can be known or unknown, it can be an object or a value from a primitive type (e.g. character string, integer, duration) or a set or list of such. The objects are partitioned into disjoint concepts (an object is an instance of one and only one concept) which determines the key and structure of its instances. For example, the "*plan*" concept identifies a plan by its number which is an integer. The fields of a particular "*plan*" are its time constraints which must be a duration and its waypoints which must contain a set of instances of the "*waypoint*" concept.
Objects can be seen under several viewpoints, each corresponding to a different taxonomy. An object can be attached to a different class in each viewpoint. For instance, a particular plan is classified as a "*flight plan*" under the nature viewpoint and as a "*logistic plan*" under the functional viewpoint. This is unlike other object systems, which usually allow only one class hierarchy.
Troeps knowledge bases can be used as HTTP servers whose skeleton is the structure of formal knowledge (mainly in the object-based formalism) and whose flesh consists of pieces of texts, images, sounds and videos tied to the objects. Turning a knowledge base into a HTTP server is easily achieved by connecting it to a port and transforming each object reference into an URL and each object into a HTML page. If HTML pages already document the knowledge base, they remain linked to or integrated into the pages corresponding to the objects. The Troeps user (through an Application Programming Interface) can explicitly manipulate each of the Troeps entities. The entities can also be displayed on a HTTP client through their own HTML page. The Troeps program generates all the pages on demand

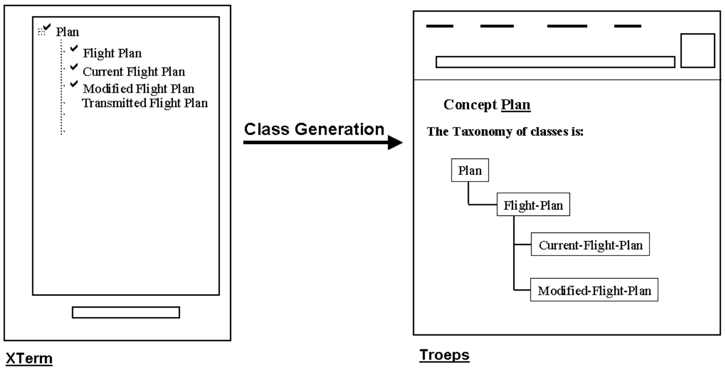

Fig. 5. Class generation

(i.e. when a URL comes through HTTP). The pages make numerous references to each others. They also display various documentation (among which other HTML pages and lexicon) and give access to Troeps features. From a Troeps knowledge server it is possible to build complex queries grounded on formal knowledge such as filtering or classification queries. The answer will be given through a semantically sound method instead of using a simple full-text search. Moreover, it is possible to edit the knowledge base. The system presented here takes advantage of this last feature.

3.3 Communication between the Components

The communication between the linguistic processing environment and the model manager is bidirectional: Upon user request, XTerm can call Troeps to generate class hierarchies from term hierarchies. Conversely, Troeps can call XTerm to provide the textual fragments related to a concept (via a technical term).
For example, figure 5 illustrates the class generation process from a hierarchy of terms carefully validated by the user (a hierarchy rooted in the term "*Plan*"). The class hierarchy constructed by Troeps mirrors the hierarchy of the validated terms (under the root "*Plan*").
At the end of the generation process, the created classes are still linked to their corresponding terms, which means that the terminology-centred navigation capabilities offered by XTerm are directly available from the Troeps interface. As illustrated by figure 6, the Troeps user has access to the multi-document view of the paragraphs which concern the "*Flight-Plan*" concepts[2]. From this view, the user can consult the source documents if required.

[2] More precisely, this view displays the paragraphs where the term "*flight plan*" and its variants occur.

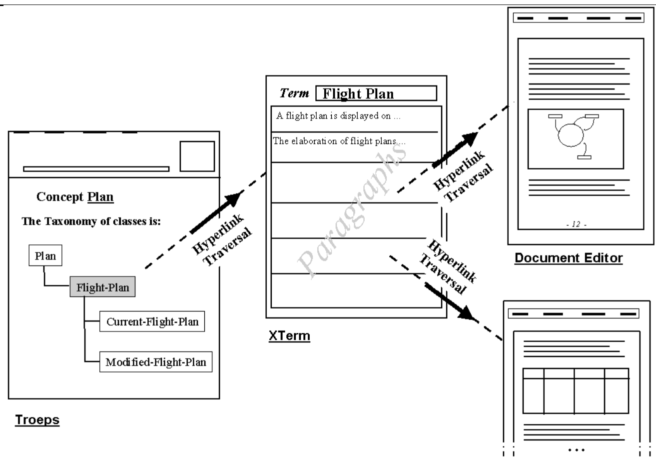

Fig. 6. Traceability through hypertext links.

Data exchanges between XTerm and Troeps are based on the XML language (see figure 4). Troeps offers an XML interface which allows to describe a whole knowledge base or to take punctual actions on an existing knowledge base. This last feature is used in the interface where XTerm sends to Troeps short XML statements corresponding to the action performed by the user. These actions correspond to the creation of a new class or a subclass of an existing class and the annotation of a newly created class with textual elements such as the outlined definition of the term naming the class. For example, to generate classes from the term hierarchy rooted at the term "*plan*", XTerm sends to Troeps an XML stream containing a sequence of class creation and annotation statements. XML representation of object models . We give below an extract of this sequence, corresponding to the creation of classes "*Flight-Plan*" and "*Current-Flight-Plan*":

```
<trp:ADD>
     <trp:CLASS>
         <trp:CLASSDSC name="Flight-Plan">
             <trp:CLASSREF name="Plan"/>
        </trp:CLASSDSC>
     </trp:CLASS>
</trp:ADD>

<trp:ADD>
     <trp:CLASS>
        <trp:CLASSDSC name="Current-Flight-Plan">
             <trp:CLASSREF name="Flight-Plan"/>
        </trp:CLASSDSC>
     </trp:CLASS>
</trp:ADD>

<trp:ANNOTATE label="comment">
     <trp:CLASSREF name="Flight-Plan"/>
    <trp:CONTENT>
              A flight plan is a sequence of waypoints ...
    </trp:CONTENT>
</trp:ANNOTATE>
```

The term definition filled out in the XTerm description of the term is added as a textual annotation in the class description. After these automated steps, the classes can be completed manually.

This XML interface has the advantage of covering the complete Troeps model (thus it is possible to destroy or rename classes as well as adding new attributes to existing classes). Moreover, it is relatively standard in the definition of formalized knowledge so that it will be easy to have XTerm generating other formats (e.g. XMI [13] or Ontolingua) which share the notion of classes and objects.
More details about this approach of XML-based knowledge modeling and exchange are given in [9].

4 Related Work

Terminology acquisition is one of the most robust language processing technology [4, 11, 7] and previous works have demonstrated that term extraction tools can help to link informal and formal knowledge. The theoretical apparatus depicted in [4], [1] and [2] provides useful guidelines for integrating terminology extraction tools in knowledge management systems. However, the models and implemented systems suffer from a poor support for traceability, restricted to the use of hyperlinks from concepts and terms to simple text files. On this aspect, our proposal is richer. The system handles real documents, in their original format, and offers various navigation and search services for manipulating "knowledge structures" (i.e., documents, text fragments, terms, concepts . . .). Moreover, the management services allow users to build their own hypertext network.
With regard to model generation, our system and Terminae [2] provide complementary services. Terminae resort to the terminologist to provide a very precise description of the terms from which a precise formal representation, in description logic, can be generated. In our approach, the system does not require users to provide additional descriptions before performing model generation from term hierarchies. Model generation strictly and thoroughly concentrates on hierarchical structures that can be detected at the linguistic level using term extraction techniques. For example, the hierarchical relation between the terms "*Flight Plan*" and "*Modified Flight Plan*" is identified by XTerm because of the explicit relations that hold between the linguistic structures of the two terms. Hence, such term hierarchies can be exploited for class generation. However, XTerm would be unable to identify the hierarchical relation that hold between the terms "*vehicle*" and "*car*" (which is the kind of relations that Terminae would try to identify in the formal descriptions). As a consequence, the formal description provided by our system is mainly a hierarchy of concepts while that of Terminae is more structural and the subsumption relations is computed by the description logic system.
In the field of software engineering, object-oriented methods concentrate on the definition of formal or semi-formal formalisms, with little consideration for the informal-to-formal processes [15, 10, 3]. However, to identify the relevant require-

ments and model fragments, designers should perform a deep analysis of the textual specifications. The recommendations discussed in section 2.2 on the use of terminological resources can be seen as a first step.
The transition from informal to formal models is also addressed in [17]. The approach allows users to express the knowledge informally (like in texts and hypertexts) and more formally (through semantic networks coupled with an argumentation system). In this modeling framework, knowledge becomes progressively more formal through small increments. The system, called “Hyper-Objet substrate”, provides an active support to users by suggesting formal descriptions of terms. The integrated nature of this system allows to make suggestions while the users are manipulating the text, and to exploit already formalized knowledge to deduce new formalization steps (this would be adapted to our system with profit). However, our natural language component is far more developed.

5 Conclusion

We have presented a fully implemented system which:

- analyzes text corpora and generates terminological resources organized in a hierarchical way;
- allows users to validate particular elements of the terminology;
- generates class hierarchies in a formal model and communicates them to the Troeps knowledge server through an XML stream;
- provides a way back from the model to the documents through the federating action of the terminology.

It thus provides both assisted generation of formal models from texts and traceability of these models back to the documents. To our opinion, this is a valuable tool for elaborating structural or formal knowledge repositories (as well as databases or software models) from legacy texts.
To improve the current system, more developments are underway for:

- improving knowledge generation by automatically detecting potential attributes and their types (the same could be possible for events, actions ...);
- implementing definition detection in texts;
- using the knowledge model as an index for providing query-by-formalized-content of the documents.

Acknowledgements

This work has been partially realized in the GENIE II program supported by the French ministry of education, research and technology (MENRT) and the DGA/SPAé.

References

1. N. Aussenac-Gilles, D. Bourigault, A. Condamines, and C. Gros. How can knowledge acquisition benefit from terminology ? In *Proceedings of the 9th Knowledge Acquisition for Knowledge Based System Workshop (KAW '95)*, Banff, Canada, 1995.
2. B. Biébow and S. Szulman. Une approche terminologique pour la construction d'ontologie de domaine à partir de textes : TERMINAE. In *Proceedings of 12th RFIA Conference*, pages 81–90, Paris, 2000.
3. G. Booch. *Object-Oriented Analysis and Design with Applications.* Addison-Wesley, 2d edition, 1994.
4. D. Bourigault. Lexter, a terminology extraction software for knowledge acquisition from texts. In *Proceedings of the 9th Knowledge Acquisition for Knowledge Based System Workshop (KAW '95)*, Banff, Canada, 1995.
5. F. Cerbah. Acquisition de ressources terminologiques – description technique des composants d'ingénierie linguistique. Technical report, Dassault Aviation, 1999.
6. K. W. Church and P. Hanks. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.
7. B. Daille. Study and implementation of combined techniques for automatic extraction of terminology. In J.L. Klavans and P. Resnik, editors, *The Balancing Act: Combining Symbolic and Statistical Approaches to Language.* MIT Press, Cambridge, 1996.
8. K. Elavaino and J. Kunz. Docstep — technical documentation creation and management using step. In *Proceedings of SGML '97*, 1997.
9. Jérôme Euzenat. XML est-il le langage de représentation de connaissance de l'an 2000 ? In *Actes des 6eme journées langages et modèles à objets*, pages 59–74, Mont Saint-Hilaire, CA, 2000.
10. I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.
11. J. S. Justeson and S. M. Katz. Technical terminology: Some linguistic properties and an algorithm for identification in text. *Natural Language Engineering*, 1(1):9–27, 1995.
12. O. Mariño, F. Rechenmann, and P. Uvietta. Multiple perspectives and classification mechanim in object-oriented representation. In *Proceeding of 9th ECAI*, pages 425–430, Stockholm, 1990.
13. OMG. XML Metadata Interchange (XMI). Technical report, OMG, 1998.
14. D. Petitpierre and G. Russell. MMORPH – the Multext morphology program. Technical report, Multext Deliverable 2.3.1, 1995.
15. J. Rumbaugh. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.
16. Projet Sherpa. Troeps 1.2 reference manual. Technical report, Inria, 1998.
17. F. Shipman and R. McCall. Supporting incremental formalization with the hyper-object substrate. *ACM Transactions on information systems*, 17(2):199–227, 1999.