Organisation and Exploitation of Terminological Knowledge in Software Localisation

E. Karkaletsis*, C.D. Spyropoulos*, G. Vouros*, C. Halatsis+

*Institute of Informatics & Telecommunications, N.C.S.R. "Demokritos", 15310 Aghia Paraskevi, Athens, Greece
Email: costass@iit.ncps.ariadne-t.gr

+Department of Informatics, University of Athens, Panepistimiopoulis, TYPA Buildings, 15771 Ilissia, Athens, Greece

Abstract. This paper presents an architecture for organising terminological knowledge of software applications and a methodology for setting up the corresponding terminological knowledge base. This organisation of terminological knowledge assists several activities during software internationalisation and localisation such as writing, translation and message generation and it also facilitates the adaptation of terminological knowledge when new versions of the software application are created.

1 Introduction

The need to adapt software applications to specific markets leads to the development of localised versions of these applications. Software localisation is the process of adding to a software application those features that better match the target market and language. On the other hand, software internationalisation is the process of developing a software application that is free of language and culture specific features. An internationalised software application can be easily adapted to a target market, by adding the linguistic and cultural elements of this market. Activities such as technical writing and translation of a software application are strongly related to software internationalisation and localisation.

One of the most important problems during technical writing concerns the inconsistent use of terms. For instance, in the same text, the same concept may be represented by two different terms, or the same term may represent two different concepts. This is a common problem that affects the quality of the software application, since it causes problems in understanding the application structure and functions. Its solution demands the acquisition of terminological data and their organisation in terminological bases, in which every technical writer has access. The inconsistent use of terms affects not only the writing but also the translation of software. For instance, the same term may be translated in two different ways or two different terms may be translated in the same way. The solution to this problem demands the creation of multilingual terminological bases.
A monolingual/multilingual terminological base contains data relevant to the structure and the functions of the software application. The exact contents of the terminological base as well as its organisation depend on the needs and the facilities of the software company.

The fields of a terminological entry format can be organised in clusters. A typical clustering is the following [Spy93]:

- **Acquisition data**, such as information for the language, country, date of term introduction, etc. Typical fields of this cluster are: language, country, terminologist, date.

- **Linguistically relevant data**, such as part of speech, idiomatic expression, abbreviation, etc. Typical fields of this cluster are: grammatical label, collocation, abbreviation.

- **Explicatory data**, such as definition, context, examples of term use, comments, text in which the term is used. Typical fields of this cluster are: definition, context, examples, comments, text.

- **Deployment data**, such as synonyms, more general terms, more specific terms, terms that refer to parts of the specific term, etc. Typical fields of this cluster are: supeordinate, subordinate, part, part-of, synonyms.

A terminological base that contains terminological data organised according to this clustering, can provide a clear view of the terminology to both technical writers and translators, reducing the amount of terminological inconsistency. In this case, the use of conventional databases presents certain drawbacks, concerning the maintenance and presentation of the terminological information [Kar94a, Mey92]. The use of tools and methods from Knowledge Engineering seems more promising. This leads to the organisation of terminological data in Knowledge Bases (KB), the so called Terminological Knowledge Bases (TKBs) [Kar94a, Mey92]. The exploitation of TKBs by a software application offers advantages such as the support of multilingual terminological data and the low cost for adapting the terminological data whenever a new version is created. On the other hand, the exploitation of TKBs has also problems concerning the cost for setting up and managing the TKB [Kar94a].

To handle the problems of TKBs, maintaining at the same time their advantages, we propose an alternative approach for the organisation of terminological data in TKBs. This approach is based on an innovative architecture and on the preprocessing of the software application texts using controlled and markup languages. We describe the innovative architecture in Section 2. The methodology for setting up a TKB for a software application is presented in Section 3. We discuss the adaptability of such a TKB in new versions of the software application in Section 4. In Section 5 we discuss the exploitation of the hybrid TKB in software localisation. Finally in Section 6 we present our concluding remarks.

### 2 An Innovative Architecture for the Organisation of Terminological Knowledge

The sources of terminological knowledge in software applications include help texts, application glossaries and domain glossaries. The terminological knowledge contained in these sources may be categorised into:
- general domain knowledge; this is knowledge about the subject of the software application (e.g. legislative terms if this is a legislative application), and about the supporting environment of the software application (e.g. DOS terms if this is a DOS-based application);
- knowledge on the software functions; this is knowledge about the software tasks and the steps needed to perform them (functional knowledge); and
- application-specific knowledge; this is knowledge on the user interface components of the software application (structural knowledge).

Based on this categorisation we propose a hybrid architecture for the organisation of the TKB. This architecture uses knowledge-based (conceptual) structures to represent the knowledge on the domain and the application and sentences from the help text to represent the knowledge on the software functions. The proposed hybrid architecture is depicted in Fig. 1.

Figure 1. The hybrid architecture

The main components of this architecture have the following characteristics:
(a) The Domain Model contains general domain knowledge. This knowledge is represented by the domain concepts.
(b) The Software Functions Model contains knowledge on software tasks (functional knowledge). This knowledge is represented by sentences from the help texts. These sentences are used in the help text to describe the software tasks. They are then translated in the languages supported, forming a translation memory.
(c) The Application Model contains application-specific terminological knowledge. This knowledge is represented by the application concepts. The application concepts are classified under the corresponding domain concepts. Those application concepts that correspond to a specific task of the software application contain a reference to this task in the software function model. This reference is the task's label. The link between the software functions and the application-specific models in Fig. 1 expresses this reference.
(d) The Language Model contains language-specific terminological knowledge. This knowledge is represented by the local concepts. The local concepts are classified under the corresponding domain and application concepts, inheriting their language independent characteristics. Each local concept can have some additional characteristics in order to represent language-specific peculiarities that cannot be represented by the domain and application concepts.

To realise the proposed architecture, we designed and implemented a prototype Knowledge Representation (KR) system [Kar94b]. A KL-ONE knowledge representation formalism [Bra91] is used that contains the following conceptual structures:

- **Concepts.** Three types of concepts are used: domain, application and local concepts.
- **Relations.** Two types of conceptual relations are supported: concept-subconcept and partitive relations. Through the concept-subconcept relation, a concept inherits its characteristics to its subconcepts. Through the partitive relation of a concept, the parts of the concept are determined. The partitive relation values of a concept are also inherited to its subconcepts.
- **Attributes.** These are the concepts attributes. The value of an attribute may be another attribute or an individual. For example, an attribute for an application concept is its reference to a task in the software functions model. In this case, the reference attribute value is the task number. Also, an attribute for a local concept is the corresponding language.
- **Individuals.** These are specific objects, such as numbers, text strings or graphical symbols. For example, the value of a reference attribute is a task number, whereas the value of a language attribute is a text string that represents that language.

### 3 A Methodology for Setting up a Hybrid Terminological Knowledge Base

We propose a methodology that aims to reduce the cost of developing a TKB. This methodology is based on the proposed hybrid architecture and on the preprocessing of the help texts using controlled and mark up languages.

The aim of using controlled languages is to restrict the syntax and vocabulary of the help texts. Then technical writers, using a mark up language may mark up those parts in the text that are necessary for the extraction of terminological knowledge. Marking up parts of the controlled text such as glossary terms, task descriptions, user interface components, we can extract automatically a large part of the domain and application-specific terminological knowledge. This knowledge will be translated then into the knowledge representation formalism and inserted into the TKB. The marked up parts, that describe software tasks are inserted in the TKB without being translated in the knowledge representation formalism. It is enough to connect these controlled and marked up task descriptions with the application-specific knowledge involved. In this way, every time a user of the TKB asks for information on a specific user interface component, he can also get information on the corresponding software task.

The main stages of the proposed methodology are described below.
3.1 Preprocessing of Help Texts

We defined a set of writing rules for a Controlled English language. These rules are based on the writing rules of Bull's Global English [Bul93]. We organised these rules into four categories. Examples of the rules of each category are presented in Fig. 2.

Figure 2. Examples of writing rules

(a) Grammar Rules
a.1 Write in the active voice and at the present tense
a.3 Use compounds of at most three nouns. Use prepositional phrases instead.

(b) Style Rules
b.1 Do not use sentences with more than 21 words.
b.3 Use a sequence of short sentences to describe a procedure. Each sentence must express only one thought.

(c) Punctuation Rules
c.1 Do not use the punctuation symbols !, ~.
c.2 Use the word "at" instead of "@", "and" instead of "&", "no" instead of "#", "." instead of ";".

(d) Rules for Words use
d.1 Use the same word or words every time you describe the same concept.
d.2 Avoid abbreviations, acronyms and other special symbols

We defined a set of markup tags based on the markup language SGML (Standard Generalised Mark up Language) [Her92]. Following the SGML syntax, the <tag-name> marks up the beginning of the marked up text whereas the </tag-name> marks up its end.

The basic steps one may follow during the preprocessing of help texts are to mark up tasks, user interface parts and domain and application concepts. Each of these steps is analysed below.

(a) Mark up of tasks. Those parts of the help texts that describe a software task are marked up. Such a task can be described by a simple sentence or by a series of sub-tasks. In the first case, only the mark up tag <task> is used:

<task>Choose the delete button in the color dialog.</task>

In the second case, two additional mark up tags are used to mark up the head (task name) and the body (sub-tasks) of the task:

<task>
<head>Delete a palette.</head>
<body>
<task>Select a palette from the palettes list.</task>
<task>Choose the delete button in the color dialog.</task>
<task>Choose the ok button in the delete palette dialog.</task>
</body>
</task>

(b) Mark up of User Interface (UI) parts. Those parts of the help texts that describe a part of the UI are marked up. Such a part can be a "window" that contains "buttons", "text fields", "sliders", etc. The mark up tag <part> marks up the whole description, the <ui> marks up a constituent, the <head> marks us the head of the description (name of the part described) and the <body> marks up the
body of the description. All these tags are used to mark up the text for the window "Modify palette dialog", that has two buttons ("Old color button", "New color button"), and a slider ("Hue slider"):

```
<part>
<head>Modify palette dialog</head>
<body>
<ui>Old color button</ui>
<ui>New color button</ui>
<ui>Hue slider</ui>
</body>
</part>
```

(c) Mark up of domain, application concepts. Those words or phrases of help texts that represent domain or application concepts are marked up. This mark up is performed after the creation of the corresponding models of the TKB. The mark up tag `<domain>` marks up a domain concept, whereas the `<appl>` marks up an application concept.

The preprocessing stage could be facilitated by the use of two tools, a controlled writing editor and a mark up editor. The controlled writing editor could assist the writer in the application of the controlled languages rules and vocabulary and could also check the final text detecting errors and proposing corrections. A mark up editor could assist the introduction of the appropriate tags and the location and extraction of the marked up parts.

### 3.2 Creation of Domain Model

The steps one may follow for the creation of the domain model are:

(a) Extraction of terms. Terms relevant to the subject and the supporting environment of the software application are extracted from the glossaries. These terms will form the basis for the definition of the domain concepts.

(b) Definition of domain concepts. The domain concepts are defined from the collected set of terms. These concepts are classified into concepts relevant to the subject and concepts relevant to the supporting environment of the software application.

(c) Creation of the concept system for the subject. The subject relevant domain concepts are classified under the concept "subject".

(d) Creation of the concept system for the supporting environment. The supporting environment relevant concepts are classified under the concept "supporting_environment".

(e) Definition of attributes and partitive relations of the domain concepts. The attributes and the partitive relation of every concept are defined and recorded in a list.

### 3.3 Creation of Software Functions Model

The steps one may follow for the creation of the software functions model are:

(a) Extraction of tasks from the help texts. The descriptions of the application tasks are extracted using the corresponding mark up information. A mark up editor could automate this step.
(b) Organisation of tasks. The extracted tasks are organised into clusters and are labelled. Each task takes a number according to the cluster and task position into this cluster.

(c) Translation. The sentences describing the organised tasks are analysed lexically and syntactically. The collected lexical and syntactic information facilitates the translation of tasks in the target languages. When the lexical and syntactic analysis is completed, the creation of a translation memory that will form the software function model starts. Every input (task) in the translation memory is translated into the application languages. During the translation, the marked up domain and application concepts are translated to the corresponding local concepts. Thus, this translation is performed after the creation of the domain and application models. This is not an automatic translation. However, it is facilitated by the controlled and marked up phrases/sentences and the collected lexical and syntactic information. The simple syntactic structure leads to a controlled translation, facilitating the whole translation task. For the translation of the remaining words (apart from the domain and application concepts), the languages dictionaries are used. In every translation the local concepts are marked up and the corresponding lexical and syntactic information is inserted.

3.4 Creation of Application Model

The steps one may follow for the creation of the application model are:

(a) Extraction of User Interface terms. Terms relevant to the user interface are extracted from the application glossaries. These will form the basis for the definition of the application concepts. A mark up editor could automate this step.

(b) Definition of application concepts. The application concepts are defined from the set of the extracted terms.

(c) Creation of the application concept system. The application concepts are classified under the domain concepts.

(d) Definition of attributes and partitive relations of the application concepts. The definition of the partitive relation is based on the marked up parts of the help texts that describe the UI structure.

3.5 Creation of Language Model

The steps one may follow for the creation of the language model are:

(a) Translation of domain and application concepts. The domain and application concepts names and characteristics are translated in the application languages. The new words are inserted to the languages dictionaries.

(b) Creation of the language concept system. The local concepts are classified under the corresponding domain or application concepts.

(c) Definition of additional attributes of the local concepts. This is done whenever some additional attributes must be inserted to certain local concepts. These are language specific attributes that cannot be represented by the corresponding domain and application concepts. Examples of such
attributes are the term representing the local concept, synonyms, abbreviations, symbols with special local meaning, etc.

4 Adaptation of the Hybrid Terminological Knowledge Base

We have already examined the issue of setting up a hybrid TKB. But what happens when a new version of the software application is created and the TKB must be adapted to the needs of this version?

First, we must note something very important. It has been estimated [Ish94] that every time a new version is produced, the modifications in the application texts are limited. More specifically, it has been estimated that approximately 70% of the text remains the same, whereas an extra 5% consists of non translatable parts. From the remaining 25%, the 15% presents many similarities with the corresponding parts of the previous version. Finally it remains a 10% that is different from the previous version.

The problem is to locate these few modifications and then update the TKB. The solution to this problem can be given by the proposed preprocessing of the help texts. That is, we can use the mark up language during technical writing to mark up those parts that are modified. In this way, using a mark up editor, we can extract the modified parts and process them to locate those tasks and concepts affected by the modifications. What we need next is to update the TKB using the facilities of the Knowledge Representation System.

The limited number of modifications and the use of mark up tags facilitates the location of the modifications. On the other hand, the use of the Knowledge Representation system facilities ensures the consistent and fast update of the TKB. Thus, we may conclude that the proposed methodology facilitates not only the organisation and extraction of terminological knowledge from help texts but also the adaptation of terminological knowledge.

5 Exploitation of the Hybrid Terminological Knowledge Base in Software Localisation

Apart from the exploitation of terminological knowledge by technical writers and translators, we have already examined another use of the hybrid TKB in software localisation, the generation of multilingual messages [Kar94b]. More specifically, we examined the generation of multilingual messages from knowledge bases using Natural Language Generation techniques. According to this approach we have to represent first the knowledge of the specific domain in some language independent knowledge representation language. This knowledge will be stored into the knowledge base. The use of knowledge bases and natural language generation techniques for message generation offers the advantage to express the same message in different ways. For example, according to the user level of expertise different information can be acquired from the knowledge base (more detailed information for the inexperienced user and less for the experienced one). However, there is a problem concerning the development of the knowledge base. The cost of knowledge base set up is large since
we must store in the knowledge base the functional and structural knowledge of the software application. But, the hybrid TKB contains both structural and functional knowledge of the software application and the proposed methodology for setting up the TKB aims to reduce this cost. Thus, the hybrid TKB can also be used for multilingual message generation.

There are also other uses of the hybrid TKB in software localisation. It can be used along with translation tools to improve their performance. For instance, it can be exploited by a translation memory system to improve the accuracy of matching when there are differences in a few terms between the input segment and a translation memory entry [Kar94a]. In that case, the hybrid TKB can be consulted to find out about the relation that holds between the differing terms. If a conceptual relation between the differing terms exists, the similarity between the two given sentences increases. On the other hand, the lack of a relation reduces the degree of similarity.

6 Concluding Remarks

The present paper proposes an approach that aims to reduce the cost of developing and managing a TKB in software applications as well as to improve the exploitation of terminological knowledge in activities such as technical writing, translation and multilingual message generation.

According to this approach, one needs to preprocess the help texts using a controlled and a mark up language. Controlled language ensures uniformity in writing style and reduces the ambiguities that affect text understanding and translation. These features make necessary the use of controlled languages for automating tasks such as translation and information extraction. The difficult issue in this case is the decision on the controlled language contents and the training of technical writers in its use [Spy93]. We believe that a controlled language must not contain complex writing rules. Instead it must be a collection of guidelines or advises for structures that should be avoided during technical writing. These guidelines concern issues of grammar, style, punctuation and words use. Their number must be limited, facilitating their understanding and use in practice by technical writers.

The use of a mark up language is necessary for information extraction from technical texts and also for the automatic translation of technical texts. The mark up language SGML is already used from several machine translation (MT) systems [Arn94]. We believe that in the next few years there will not exist any commercial MT system without providing tools for the use of SGML Especially in the case of information extraction from help texts, the use of a mark up language offers more advantages. The definition of the types of terminological knowledge, according to the proposed hybrid architecture, permits their unambiguous mark up in the help text. Also, the structure of the help texts, the way they describe the various tasks and parts, permits the easy mark up of the corresponding text parts. Furthermore, the introduction of mark up tags for marking up the modifications in a new version facilitates the adaptation of the terminological knowledge.

All these remarks led us to the specification of a small set of mark up tags, with syntax based on SGML. The small number of these tags facilitates their use in practice by the technical writers. The specification of Document Type Definitions (DTDs) [Arn94] for the application texts could also help.
For example such a DTD for help text could specify the presentation order of the issues covered, the tasks description, the parts description, etc. Thus, a company could define its own general DTD for the help texts of its software applications. This DTD could be extended according to the application type. Such an approach could speed up the acquisition of various information types from the application texts.

References


