A knowledge-based methodology for supporting multilingual and user-tailored interfaces

Evangelos A. Karkaletsis\textsuperscript{a}, Constantine D. Spyropoulos\textsuperscript{a, b}, George A. Vouros\textsuperscript{b}

\textsuperscript{a}Institute of Informatics and Telecommunications, National Centre for Scientific Research (N.C.S.R.)
\textsuperscript{b}Department of Mathematics, University of the Aegean, Karlovassi, Samos, Greece

Abstract

The need for multilingual and user-tailored interfaces imposes new requirements upon the software industry: software applications must "speak" the language of users. Language engineering and knowledge engineering can assist the development of such interfaces. This paper presents a methodology for the creation of a language-independent knowledge base (KB), which can be used for the development of multilingual and user-tailored interfaces. This KB contains knowledge about the user interface components and functions and its creation is part of a software internationalisation process. The methodology aims at reducing the cost of setting up and managing this KB, by exploiting the benefits of controlled language use in technical writing. A case study for the dynamic generation of multilingual and user-tailored diagnostic messages is presented. Finally, the paper discusses related approaches in the area of multilinguality as well as in the area of software internationalisation and localisation, summarises the main results, and presents our plans for further exploitation of the methodology. © 1998 Elsevier Science B.V.

Keywords: Knowledge bases; Software internationalisation; Software localisation; Multilingual interfaces; User tailored interfaces; Natural language generation; Message generation

1. Introduction

The development of software applications with multilingual and user-tailored interfaces is a major challenge for software companies. To be able to develop applications with such functionalities, one should follow an internationalised approach. According to this
approach, the culture, language and user-dependent parts of the software application are isolated from the software core during development [1]. This enables the localisation of the software application to the needs and requirements of the target local markets and of the individual user.

The present paper presents a methodology for the creation of a language-independent knowledge base (KB) which can support the development of multilingual and user-tailored interfaces. This KB contains knowledge about user interface components and functions of the software applications and its creation is part of the software internationalisation process. Although knowledge-based approaches can support multilinguality and user tailoring, they also present a significant problem concerning the cost for setting up and managing the KB [2]. The methodology presented here involves a new architecture for the organisation of the KB and a new method for the set up of the KB that aim to reduce the development cost.

The methodology has been exploited in a case study for the dynamic generation of messages. Messages are displayed to the user as a response to certain events and form a vital aspect of the user interface. The specific case study concerns the dynamic generation of multilingual (English, Greek) and user-tailored diagnostic messages. The resulting prototype message generation system exploits the knowledge of functions and components of the user interface that is stored in the KB. A natural language generation (NLG) component is used by this prototype to produce a message in the specified language with the appropriate level of detail according to the user's expertise and interests on the application domain.

Section 2 of the paper presents our knowledge-based methodology and discusses its benefits and costs. The architecture for the organisation of the KB, the set up method, the addition of a new language, and the update of the KB in case a new version of the software is produced are also discussed in Section 2. Section 3 describes an application of the proposed methodology for generating multilingual and user-tailored diagnostic messages, using examples from the Style Manager case study. Finally, Section 4 examines the proposed approach in comparison with relevant works in multilinguality and software internationalisation and localisation. Finally, Section 5 summarises the results of our work and presents our future plans.

2. A knowledge-based methodology

2.1. General specifications

The set up of a KB involves first the acquisition of knowledge about the relevant application domain. Management of the KB involves the introduction of new knowledge, the modification of the existing KB, consistency checks, and the retrieval of knowledge. A large KB and a complex knowledge representation (KR) formalism increase these costs significantly. We must find ways to limit the growth of the KB, as well as devise a simple KR formalism.

In our case, the relevant domain is the user interface components and functions of the software application. The KB in this domain should represent the following types
of knowledge:

- **Domain knowledge.** This is general structural knowledge about the domain and the supporting environment of the software application;
- **Functional knowledge.** This is knowledge about the tasks that users should perform to achieve user interface functions; and
- **Application-specific knowledge.** This is structural knowledge about the user interface components of the software application.

A way of representing these types of knowledge is the use of **conceptual formalisms** [3]. The basic building block of such formalisms is the *concept*. Each concept represents a generic aspect that is characterised by a set of *attributes*. Attributes represent properties of concepts. Concepts are taxonomised in a classification hierarchy of concepts-subconcepts using *concept-subconcept relations*. In general, a system of concepts, attributes and relations used to represent the knowledge of a domain is called the conceptual system of the domain.

The use of conceptual formalisms for representing the structural knowledge in the KB requires only the use of one more relation, apart from the concept-subconcept relation. This is the *partitive relation* that represents the part–whole relationships among concepts. On the other hand, the conceptual representation of functional knowledge requires the introduction of other relations (e.g. causal or temporal), which significantly increase the size of the conceptual system, leading consequently to the specification of a complex KR formalism which is difficult to maintain and utilise. Thus, we need to examine other ways of representing the functional knowledge. We decided to follow a hybrid approach for representing functional knowledge. In the proposed KR formalism, functions are represented by English sentences and are organised in an abstract–concrete hierarchy according to the purpose that they achieve, in a manner common in traditional plan structures. Each function has a reference number which describes its position in the hierarchy. For instance, functions labeled "1.1" are more concrete than functions labeled "1".

This introduction of natural language (English) for the representation of functions significantly reduces the size of the conceptual system, which then contains only concepts for representing structural knowledge. However, the use of natural language may also affect the language-independent representation of knowledge that the use of concepts supports. At this point, the use of a *controlled language* can help us. A controlled language is a set of writing rules and a vocabulary that restrict the use of the corresponding natural language during technical writing.

The aim of using controlled languages in technical writing is the production of technical texts with simple structure and restricted vocabulary that can be read and translated easier [1, 4]. Software companies such as Bull, IBM and Perkin-Elmer as well as other companies such as Caterpillar, General Motors, Boeing and British Aerospace Airbus are already using controlled languages during technical writing of their products. Bull uses Bull’s Global English (BGE), IBM uses the IBM Style Digest, Perkin-Elmer uses the Perkin Approved Clear English (PACE), Boeing uses Simplified English (SE), British Aerospace uses AECMA Simplified English, and Caterpillar is involved in the Controlled Automotive Service Language (CASL) project. Bull has also developed a tool (Bull’s Max
editor) that checks the correct use of the controlled language by its technical writers. This tool locates errors and notifies the users, but it does not correct errors. Boeing has deployed the Boeing SE checker to help ensure that its technical writers conform to the controlled language, as defined by various industry standards and government regulations. Caterpillar is working with the Carnegie Mellon University and the Carnegie Group to develop an authoring tool called Clearcheck, hoping that this will make possible to translate manuals with virtually no human post-editing. There was also a project funded by the EU that aimed to develop a tool that would not only check for possible errors but also correct them. This is the LRE project “A Simplified English Grammar and Style Checker/Corrector” SECC [5].

We evaluated the use of controlled language in the context of the R&D project GLOSSASOFT [6]. More specifically, we used BGE [7] to rewrite the help texts of a software product. We evaluated readability and translatability (by a Machine Translation (MT) system) of the resulting controlled text. The application of BGE made the text more controlled and uniform, improving its readability — especially by non-English speaking users. The original text and the controlled one were sent for translation by the MT system SYSTRAN (using its English–Greek module). The results confirmed the advantages offered by the use of controlled language in MT, justifying its adoption in technical writing by several multi-national software (and other) companies. The controlled text translation was much more readable than the original one. In fact, few corrections were necessary while proofreading the controlled text translation. We realised that the translation quality would be even better if we were stricter in the application of some control rules.

These results convinced us that the use of a controlled English language in the KB will simplify descriptions of user interface functions and allow their automatic translation to other languages of the software application. In this way, controlled descriptions form a language-independent representation of functional knowledge.

The use of controlled language is a major strength of our methodology. We use controlled English not only to improve readability and to facilitate machine translation of technical texts, but also to reduce the cost of setting up and managing the language-independent KB.

Concerning the setting up of the KB, there is also the issue of reducing the cost of extracting each type of knowledge from knowledge sources. According to the proposed methodology, two types of knowledge must be extracted: structural and functional knowledge. The adoption of specific knowledge structures for description of the user interface (UI) components and their parts, as well as of the UI functions in which these components participate, can help to automate extraction of relevant knowledge. This can be done using a mark up language which contains a set of rules for the mark up of the appropriate information in a text, as well as a set of the corresponding mark up tags. The aims of using such languages are the creation of technical texts that follow a specific structure and the easy location and exploitation of the marked up information. A standard mark up language (SGML) has already been defined by the International
Standards Organisation (ISO) [8]. Thus, it is necessary to preprocess help texts in order to reduce the cost of setting up and managing the KB. This preprocessing involves the use of a controlled and a mark up language.

To realise the proposed knowledge-based methodology, we designed an architecture for the organisation of the KB, a method for setting up the KB, a means to support new languages, and a way to update the KB. These are presented in the following subsections. We implemented a prototype system and used it to create the KB of a software application in two case studies. The first study involved the Style Manager of the HP VUE system [9], whereas the second study involved MicroSoft's WordPad application in Windows 95 [10]. The proposed methodology for supporting multilingual and user-tailored interfaces has been exploited for generation of a significant element of the user interface: its messages. We designed the architecture of a knowledge-based message generation system and implemented a prototype for the generation of multilingual and user-tailored diagnostic messages. The prototype used the KB created for HP VUE Style Manager in a case study featuring the generation of user-tailored diagnostic messages in English and Greek. The architecture of the knowledge-based message generation system and the case study are presented in Section 3.

2.2. The architecture of the KB

The proposed architecture is depicted in Fig. 1. The main components of this architecture include:

- The Domain Model. This contains general domain knowledge. The conceptual representation is a simple KL-ONE-like knowledge representation formalism [3]. Domain entities are represented by domain concepts which are related among themselves by the concept–subconcept and partitive types of relations.
- User Interface Functions Model. Functions are described by controlled English sentences and are organised in an abstract–concrete hierarchy. Each sentence is automatically translated to the target language using a simple transfer-based MT system [4]. The translated sentences are stored in a set of databases (one for each language) which are linked to the database containing controlled sentences. In this way, a translation memory of the controlled sentences and their translations is created.
- Application Model. Application concepts represent the user interface components of the specific application and are classified under domain concepts. Application concepts have a distinguished attribute, called "reference". This is automatically filled by a reference to a function in the User Interface Functions Model. Such a reference indicates a function in which the application concept has a functional role.
- Language Model. Local concepts are classified under domain and application concepts, inheriting their language-independent characteristics. A local concept may

2HP VUE is a Windows Management System and User Interface built by Hewlett Packard over X11R5 Windows, and Style Manager is the application that enables users to change the behaviour and appearance of HP VUE system colours, sound, keyboard, mouse, windows and sessions.

3Word Pad is a word processing application that contains a limited set of the functionalities provided by MicroSoft's Word for Windows.
have additional attributes to express some language-specific peculiarities that cannot be expressed by domain and application concepts.

Summarising, the basic characteristics of the proposed architecture are the following:

- Description of tasks in the User Interface Functions Model using Controlled English. This facilitates the KB set up and allows automatic translation into languages supported.
- Clear separation of knowledge into structural (Domain and Application Models) and functional (User Interface Functions Model). This reduces the size of the conceptual system and limits the complexity of the KB formalism.
- Clear separation of structural knowledge into language-independent (Domain and Application Model) and language-dependent (Language Models). This facilitates the introduction of new languages into the KB, as well as the modification of existing ones.

2.3. Setting up the KB during software internationalisation

We have implemented a prototype system used for setting up the KB in the two case studies mentioned above. This prototype involves a knowledge editor for adding new concepts and tasks within the KB, as well as for modifying or deleting existing ones. The basic steps of the KB set up method during internationalisation are shown in Fig. 2.

2.3.1. Step 1

Text Preprocessing. We devised a controlled English language that is based on the rules of BGE [7]. We organised the rules into four categories [11] (Fig. 3). We also devised a
mark up language that is based on SGML [11, 9]. In the case of existing software (as in our case studies) the help texts describing user interface components, their parts and functions are marked up in order to facilitate the extraction of application concepts and relevant function descriptions. The function descriptions are also rewritten according to the controlled English rules and lexicon. In the case of new software, function descriptions are written in controlled English from the beginning.

2.3.2. Step 2

Creation of the Domain Model. The technical writer identifies the domain concepts

(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

Fig. 2. Setting up the KB during software internationalisation — using the KB during software localisation.

Fig. 3. Examples of controlled English writing rules.
Creation of the Application Model. The technical writer determines the application concepts and adds them to the KB using the knowledge editor (see application models in Fig. 4(b) and Fig. 5(a)).

Fig. 4. Continued.

and adds them to the KB using the knowledge editor (see domain models in Fig. 4(b) and Fig. 5(a)).
### 2.3.3. Step 3

Creation of the UI Functions Model. The controlled descriptions of functions are organised in an abstract–concrete hierarchy using the knowledge editor (see Fig. 4(c) and Fig. 5(b)).

### 2.4. Supporting a new language

The addition of a new language in the language-dependent part of the KB (Fig. 1), involves the following steps (Fig. 2):

#### 2.4.1. Step 1

Creation of the Language Model. The translator creates a lexical representation of domain and application concepts in the target language and adds them to the KB using the knowledge editor (see language models in Fig. 4(b) and Fig. 5(a)).

#### 2.4.2. Step 2

Translation of the controlled descriptions. The controlled English sentences describing user interface functions are analysed by the simple transfer MT module, producing a representation from which corresponding target language sentences are generated (see the translated functions descriptions in Fig. 4(c) and Fig. 5(a)). The MT module consists of (a) a morphological analyser of English sentences, (b) a syntactic analyser (chart parser, top-down, depth-first), (c) lexicon and grammar of controlled English (PATR notation is used), (d) lexicon and grammar rules of target languages, which contain links to the corresponding controlled English words and grammar rules (PATR notation is used), and (e) a morphological generator that produces the sentence in the target language. The notations and engines we used in this MT module are standard; we implemented them for demonstration reasons. Any other MT system could be used, exploiting the benefits of controlled English.

---

**Table: Controlled Descriptions of UI Functions**

<table>
<thead>
<tr>
<th>Verb</th>
<th>Noun Phrase</th>
<th>Verb</th>
<th>Noun Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 Modify a palette.</td>
<td>modify a palette</td>
<td>12 Τροποποιήστε μία παλέττα</td>
<td>τροποποιήστε μία παλέττα</td>
</tr>
<tr>
<td>121 Select a palette from the palettes list</td>
<td>select a palette from the palettes list</td>
<td>121 Επιλέξτε μία παλέττα από τη λίστα παλεττών</td>
<td>επιλέξτε μία παλέττα από τη λίστα παλεττών</td>
</tr>
<tr>
<td>122 Open the modify dialog</td>
<td>open the modify-dialog</td>
<td>122 Ανοίξτε το διάλογο τροποποίησης</td>
<td>ανοίξτε το διάλογο τροποποίησης</td>
</tr>
<tr>
<td>1221 Click a color_button</td>
<td>click a color-button</td>
<td>1221 Πατήστε ένα κουμπί χρώμα</td>
<td>πατήστε το κουμπί χρώμα</td>
</tr>
<tr>
<td>1222 Click the modify button</td>
<td>click the modify-button</td>
<td>1222 Πατήστε το κουμπί τροποποίησης</td>
<td>πατήστε το κουμπί τροποποίησης</td>
</tr>
</tbody>
</table>

---

**Fig. 4.** (a) The color dialog of Style Manager. (b) Part of the conceptual system of the knowledge base for Style Manager. (c) Part of the User Interface Functions Model of the knowledge base for Style Manager.
2.5. Updating the KB

We have already examined the issue of setting up the KB during the development of an internationalised software application. But what happens when the KB must be updated in order to support a new version of the software? It has been estimated [12] that every time a new version is produced, 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. Of the remaining 25%, 15% has many similarities to corresponding parts of the previous version. Finally 10% remains that is different from the previous version.

Therefore, the problem is to locate these few modifications (the 10%) and to update the KB. The solution to this problem can be the proposed preprocessing of 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 KB using the knowledge editor. The limited number of modifications and the use of mark up tags facilitates location of modifications. Therefore, the tools provided by our prototype facilitate the consistent update of the KB. Thus, we may conclude that the proposed methodology facilitates not only the set up of knowledge stored in the KB, but also its maintenance.

3. Exploiting the knowledge base for generation of diagnostic messages

3.1. Current approaches for message generation in internationalised software applications

In an internationalised software application, messages are stored separately from the source code, in message catalogues. In this way, the localisation of messages does not involve rewriting of software code, but only the adaptation of the message catalogues [13].

The most common approach for message generation in internationalised software applications is the use of canned messages. Since canned messages are fixed, they cannot be tailored to the user’s expertise level, the task he performs, and the current application context. Also, different message catalogues, one for each language, have to be maintained. A major drawback of this approach concerns the organisation and size of the message catalogues in each language, since software developers have to maintain separate entries for messages with many similarities.

A second approach concerns the generation of messages that are not maintained in the message catalogue, but instead are the result of combining two or more canned messages. Although this approach reduces the size of the message catalogue, it can cause problems during software localisation. For instance, let us consider the following messages

1 No rights to
2 Open
3 Modify
4 File <file name >

When these messages are combined, they can generate messages such as "No rights to open file <file name >", "No rights to modify file <file name >", "No rights to file <file name >". This message combination is not the proper one for an internationalised software application. In the first two messages, word "to" is the first part of the verb
infinitive, whereas in the third one "to" is a preposition that refers to the noun phrase "file <file name>". It is extremely difficult to translate the message "No rights to" into other languages in a way that covers both uses of "to". The morphological and grammatical differences between languages make the application of such an approach problematic in practice.

A third approach concerns the use of message templates. These are messages with slots which can be replaced by actual values, depending on the context in which each message is generated. For example, instead of having four different messages to announce that one of the four system disk drives is damaged, we can use the message template "Disk <nuHr of disk> is damaged". This message template can be extended "< Disk(s) > <nuHr(s) of disk(s) > <be > damaged" in order to be used for the generation of messages in the singular (e.g. Disk 1 is damaged) and in the plural (e.g. Disks 2, 3 are damaged), if a morphological generation routine is called to make the appropriate morphological adaptations. The use of these extended message templates is especially important for languages with many inflectional word forms (synthetic languages, such as Finnish). There will be no need to maintain separate entries inside the message catalogue for the messages that have only different word forms.

In Ref. [13] we presented an architecture for internationalised software applications (see Fig. 6) that is based on the use of extended message templates and morphological generation routines.

According to this architecture, the catalogue of the extended message templates for the language Li is outside the core of the software application. Two features are included in each extended message template: the language and the linguistic specification (morphological features) for lexemes that will replace slots in the message template. The software source code asks for the appropriate message from the Messages Manager, giving the message template code and slot fillers. The Messages Manager retrieves the corresponding message template and sends the lexemes and their morphological features to the Morphological Generator. The Morphological Generator inflects the lexemes to their proper forms, and returns them to Messages Manager. The inflected lexemes replace the slots in the message template, forming the message that will be presented to the user.

Fig. 6. On-line generation of messages using extended message templates.
Although this approach improves significantly the organisation of message catalogues, it still requires that different extended message templates are maintained for each of the languages supported, and it cannot support the generation of user-tailored messages.

It would be advantageous if we could generate messages for all languages from one common language-independent ("interlingual") representation. It would also be useful to be able to express the same message in a single language in different ways, according to user expertise, style and plans.

3.2. Exploiting the KB for the generation of multilingual and user-tailored diagnostic messages

Our approach is based on the existence of a "language-independent" KB that contains knowledge about user interface functions and about components of the internationalised software application. This KB is created semi-automatically during the internationalisation process and is exploited by a message generation system which dynamically generates multilingual and user-tailored messages.

The general architecture we propose for message is depicted in Fig. 7. This architecture is similar to that depicted in Fig. 6, but it does not include any message catalogues. Messages are generated dynamically using the knowledge of user interface functions and components that is stored in the KB. The main modules of the message generation system according to this architecture are as follows.

- **User level and style identification.** This module acquires the level of user expertise and the message style that the user prefers.
- **Messages Manager.** This module generates a language-independent representation of the appropriate message. Messages Manager is invoked by the source code; it takes into account the current context of the software, along with information from the KB. The resulting representation is passed to the Natural Language Generation (NLG) system.
Natural Language Generation system. The NLG system decides first on the content of information that will be presented to the user. Depending on user level, different information is extracted from the KB (i.e. more detailed information for an inexperienced user and less detailed information for a more experienced user). The NLG system then converts this information into the user's language, using the appropriate grammar, lexical and morphological rules.

This approach offers many advantages with regard to multilinguality, user tailoring, maintainability and enforcement of specific style and content of messages. Specifically,

- the NLG system can generate the same message in more than one language, using the grammar, lexical and morphological rules for the supported language;
- messages can be tailored to the user's level of expertise and to the current software context;
- the cost of development of a new version and of creation of new local versions becomes lower, since the work required is reduced to an update of the KB and of the lexicons and grammars of local languages; and
- message generation by an NLG system permits enforcement of externally imposed writing rules and content requirements.

3.3. A case study

We implemented a prototype message generation system that generates diagnostic messages in English and Greek every time the user performs an unsuccessful task in a software application. We used the KB we created for the HP VUE Style Manager (see Fig. 4(b) and (c)); we simulated in MS Access the Style Manager user interface (see Fig. 4(a)). The architecture of this prototype (depicted in Fig. 8) is based on the more general architecture of Fig. 7. The basic modules of the message generation prototype are as follows.

![Fig. 8. Generating multilingual diagnostic messages.](Image)
User level identification. Every time users start a new session with the software application, they declare their level of expertise and the style of message they prefer. According to the user level and style type, different diagnostic messages for the same fault can be generated. To exemplify this, we defined four user levels of expertise and four styles. The user levels with their descriptions are as follows:

<table>
<thead>
<tr>
<th>User-level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (more experienced)</td>
<td>Information only on the task that should be executed previously, in order to perform successfully the current task.</td>
</tr>
<tr>
<td>2</td>
<td>Information on the previous task and the purpose of the unsuccessful current task.</td>
</tr>
<tr>
<td>3</td>
<td>Information on the previous task and the unsuccessful current task.</td>
</tr>
<tr>
<td>4 (less experienced)</td>
<td>Information on the previous task, the unsuccessful current task and its purpose.</td>
</tr>
</tbody>
</table>

The styles with their descriptions are as follows:

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Imperative (e.g. Click a color-button)</td>
</tr>
<tr>
<td>s2</td>
<td>Have to (e.g. You have to click a color-button)</td>
</tr>
<tr>
<td>s3</td>
<td>Must (e.g. A color-button must be clicked)</td>
</tr>
<tr>
<td>s4</td>
<td>Did not (e.g. You did not click a color button)</td>
</tr>
</tbody>
</table>

- Messages Manager. The History-List maintains information on successfully performed tasks. Every time the user performs a task successfully (selects a menu option, presses a button, etc.), the History-List is updated. Whenever a task cannot be performed, the Messages Manager is invoked to diagnose the cause, extracting the appropriate information from the KB and the History-List. The Messages Manager translates the diagnosis into the knowledge representation formalism. This representation is passed to the NL Generation module in order to produce the final message in the language of the user.

- NL Generation system. The NLG system consists of strategic and tactical components. The strategic component takes the diagnosis as input together with user level and style, and decides the content of information that will be displayed to the user. The tactical component takes this content as input, together with the grammar and lexicon for the target language, and produces the final message.

To clarify the knowledge-based approach for generation of diagnostic messages, we offer the following example. Let us assume that the user is in the colour dialogue and wants to modify a colour of the currently selected palette. As shown in the User Interface Functions Model of the KB (Fig. 4(c)), to modify a colour of the palette the user must first select the palette (Task 1.2.1) (see Fig. 9(a)), then select the colour (Task 1.2.2.1), and then click the modify-button to open the modify-dialogue (Task 1.2.2.2). Let us assume that the user has already selected the palette, and makes the mistake of clicking the modify-button without having selected a colour first. The History-List will contain at that time the information that Task 1.2.1 was performed. The Messages Manager will be activated
to produce a message using the information contained in the History-List and in the KB. The Messages Manager uses this information to find the corresponding concept in the Application Model (see Fig. 4(b)). It finds the concept "modify-button" which has the attribute "ref" with value "1.2.2.2". It then searches in the User Interface Functions Model for Task 1.2.2.2. This is the task "1.2.2.2. Click the Modify button.". Based on the information from the History-List and the Software Functions Model, the Messages Manager diagnoses the error. That is, it finds that Task 1.2.2.1 had to be performed first. The representation of this diagnosis is as follows:

\[\text{precondition}(1.2.2.2,[1.2.2.1,1.2.2.2])\]

which means that in order to perform successfully the current task (1.2.2.2) and finally Task 1.2.2, the user must first perform Task 1.2.2.1.

Let us assume that the user declared that his user level is equal to 4 at the beginning of his session with the software application. The strategic component will use this information in order to decide the content of the final message. Since the user level is equal to 4, the message must contain information on the previous task (1.2.2.1), the unsuccessful current task (1.2.2.2), and the final task (1.2.2). After content determination, the strategic component must decide on the message style. This decision is based on the style declared by the user at the beginning of his session. If the user selects the style \(sl\), the strategic component decides that the message will be in the imperative.

The tactical component takes this information as input, plus the current and the final task, which must be included in the final message. It searches for translations of these tasks in the translation memory of the User Interface Functions Model, according to the user's language. From these translations, the tactical component extracts the verbs and noun
phrases it needs for the final message. The translations of the tasks in Greek and the words extracted are shown below:

<table>
<thead>
<tr>
<th>TASK</th>
<th>Greek Translation</th>
<th>Verb</th>
<th>Noun Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous task</td>
<td>1.2.2.1 Πατήστε το κουμπί-γράφημα</td>
<td>πατήστε</td>
<td>το κουμπί-γράφημα</td>
</tr>
<tr>
<td>current task</td>
<td>1.2.2.2 Πατήστε το κουμπί-τροποκοίτηση</td>
<td>πατήστε</td>
<td>το κουμπί-τροποκοίτηση</td>
</tr>
<tr>
<td>final task</td>
<td>1.2.2 Ανοίξτε το διάλογο-τροποκοίτηση</td>
<td>ανοίξτε</td>
<td>το διάλογο-τροποκοίτηση</td>
</tr>
</tbody>
</table>

Using these verbs and noun phrases, the tactical component activates the grammar rule for imperatives, converts the words into the appropriate morphological forms (using the morphological rules of the language), and generates the final message (see Fig. 9(b)). Note that the same representation formalism was used for both English and Greek grammars and lexicons.

For example, the tactical component will generate the following messages for the four styles and for user-level equal to 4.

<table>
<thead>
<tr>
<th>Style</th>
<th>Message in English</th>
<th>Message in Greek</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>Click the button and then click the modify-dialog.</td>
<td>Αρχικά πατήστε το κουμπί-γράφημα και στη συνέχεια πατήστε το κουμπί-τροποκοίτηση για να ανοίξτε το διάλογο-τροποκοίτηση.</td>
</tr>
<tr>
<td>s2</td>
<td>You have to click the button first and then click the modify-button to open the modify-dialog.</td>
<td>Αρχικά πατήστε το κουμπί-γράφημα και στη συνέχεια πατήστε το κουμπί-τροποκοίτηση για να ανοίξτε το διάλογο-τροποκοίτηση.</td>
</tr>
<tr>
<td>s3</td>
<td>The button must be clicked first. Then, click the modify-button to open the modify-dialog.</td>
<td>Αρχικά πατήστε το κουμπί-γράφημα. Στη συνέχεια πατήστε το κουμπί-τροποκοίτηση για να ανοίξτε το διάλογο-τροποκοίτηση.</td>
</tr>
<tr>
<td>s4</td>
<td>You didn’t click the button. Click a button first and then click the modify-button to open the modify-dialog.</td>
<td>Δεν πατήστε το κουμπί-γράφημα. Αρχικά πατήστε το κουμπί-γράφημα και στη συνέχεια πατήστε το κουμπί-τροποκοίτηση για να ανοίξτε το διάλογο-τροποκοίτηση.</td>
</tr>
</tbody>
</table>

If the user specifies that, in addition to being at level 4, he prefers the first style, then the diagnostic message shown in Fig. 8 will appear.

4. Discussion of related work

Our methodology for supporting the development of multilingual and user tailored interfaces is related to work from the areas of multilinguality, user tailoring, software internationalisation and localisation. Such work is discussed in this section, in comparison with our methodology. Concerning multilinguality and since the KB set up is a major aspect of our methodology, the work presented addresses knowledge-based translation, organisation of multilingual KBs, and knowledge-based generation. Concerning user tailoring, we present the work of our team in the tailoring of information delivered to users by information systems, and we discuss how this work could be integrated into the proposed methodology. Finally, concerning software internationalisation and localisation, we present a knowledge-based approach for localisation of cultural aspects of user interfaces (an approach suggesting the use of controlled languages in software internationalisation), as well as work addressing the creation of an internationalised and multilingual computing environment.
The issue of setting up a language-independent KB is central in the discussion of interlingual machine translation (MT) systems, where the main idea is to analyse source language sentences, converting them to the interlingual representation of the KB, and then to generate from that representation sentences in the target language [4, 14]. The main problem in interlingual MT lies in the setting up of the language-independent KB. A solution to this problem is to restrict either the domain or the application. AVENTINUS is a multilingual project funded by the EC [15], which is based on the assumption that it is possible, when dealing with a particular domain or application, to construct a language-independent representation of concepts relevant to the domain. English is used to label domain concepts (events and objects) and their relations. The authors call this representation a domain model. AVENTINUS produces a language-independent, conceptual representation of text (English and French), using this domain model. The conceptual representation is “translated” to the required language by mapping the concepts to the language’s lexicon. The AVENTINUS approach is similar to ours, especially concerning the restricted domain. The domain of software application is restricted and well organised by its own nature. Thus, it constitutes an ideal test bed for application of approaches based on language-independent KBs.

EuroWordNet [16] is another EC-funded project which aims to implement a multilingual lexical KB (EWN). The EWN KB will store semantic relations between words in four different languages: Dutch, Italian, Spanish and English. WordNet is a freely available lexical database for English, which consists of semantic relations between words (words with similar meanings are grouped together in the so-called synsets). The EuroWordNet project aims to use the EWN knowledge base to index documents in terms of the EWN Interlingual Index (ILI). ILI is essentially WordNet plus a set of complex translation links between language-specific synsets and WordNet synsets. Thus, documents from different languages essentially are indexed in terms of an English KB, the WordNet. English is used as an interlingua but not in a specific domain as it was the case in AVENTINUS. This approach is in agreement with our methodology, in which the controlled English description of functions constitutes a language-independent representation of functional knowledge.

The automatic multilingual generation of texts from a knowledge base has been the object of many recent works. IDAS [2] is a work about knowledge-based generation, which although not multilingual is discussed here because it shares basic principles with our proposed methodology. IDAS organises knowledge into a KB using a KL-ONE-like knowledge representation system, and uses this KB along with NLG techniques to generate on-line documentation and help messages for users of complex physical machinery. IDAS researchers emphasise the KB set up cost, noting that “the single biggest cost of producing documentation with NL Generation techniques is setting up the requisite KBs for the generation system to generate text from” [2]. They propose a hybrid technique for representing functional knowledge, based on the mixing of canned texts with concepts for reducing cost. This technique precludes multilingual exploitation in IDAS [17]. Although there have been some experiments including other languages, no truly multilingual version of IDAS yet exists.

DRAFTER [18] is a system developed to support technical writers in drafting multilingual software manuals. DRAFTER includes a domain model, implemented using a
KL-ONE-like language, which is a collection of entities (concepts and relations) occurring in the software domain. DRAFTER provides the technical writer with an interface that allows him to specify software functions formally using concepts and relations in the domain model. The system also provides the writer with a tool which generates draft instructional text (English and French) for the software functions defined. Although the DRAFTER work is similar in many ways to our approach, it does not address the problem of setting up a language-independent KB. In our approach, the technical writer does not have to describe software functions using concepts and relations, but instead can use a linguistic description (controlled English) to facilitate his interaction with the knowledge editor.

Manfred Stede [19] discusses multilingual generation from a language-independent KB, based on work in the TECHDOC project. He emphasises the problem of concept lexicalisation: not all lexical items map one-to-one onto concepts. Stede proposes a two-step lexicalisation process in order to deal with this problem. We have not investigated this in our work, since we emphasised setting up and managing the language-independent KB. However, we believe that our use of controlled language in the description of software functions limits the effects of this problem. Restricted vocabulary and writing rules reduce ambiguities in technical writing and translation, thereby reducing the number of lexical items that do not map one-to-one onto concepts.

Sharoff and Sokolova [20] investigated characteristic features of technical documents in several European languages (English, French, German and Russian). The authors identified and described language-independent functional structures found in all the documents. Identification of such structures is further evidence that adoption of a common language-independent structure in technical documents is feasible. The use of controlled English in sentences or phrases to describe the content of functional structures (as is done in our approach) enforces the language-independent character of technical documents structured and written in this way. The adoption by software companies of mark up and controlled languages for technical writing is the current trend. This practice creates the appropriate foundation for successful integration of our approach within the internationalisation process.

User-tailored interfaces in software applications are designed to modify their behaviour according to the individual needs of human users, within the application context. In order to provide such individualised behaviour, a software application must possess knowledge about the user (i.e. his user model). A user model is a knowledge source which contains explicit assumptions regarding all aspects of the user that are relevant to the interactive behaviour of the software application (such as the user’s knowledge of the domain, his interests, and his goals). A user modeling component, on the other hand, is that part of a software application whose function is to construct a user model incrementally; to store, update and delete entries in this model; to maintain the consistency of the model; and to supply other software application components with assumptions about the user. Benaki et al. [21] discuss the application of user modeling techniques to tailor information delivered to users by information systems, and present a prototype user modeling component developed by our team for supporting information extraction systems. Our message generation prototype currently uses a simple approach to identify a user’s experience (knowledge) in the application domain, (the user must select his own expertise level from a small set of predefined user levels). Integration of the user modeling component described by Benaki et al. [21] would allow automatic identification of a user’s knowledge and
interests in the domain without requiring predefined user levels, thereby leading to more intelligent user-tailored interfaces.

In our methodology, we propose the creation of an internationalised linguistic base (the language-independent KB) that can support development of multilingual and user-tailored interfaces, without examining cultural aspects of user interfaces. The localisation of cultural aspects using a knowledge-based (conceptual) approach was examined by Semmar et al. [22]. Aiming at icon localisation, the authors describe a tool that consists of a module for icon analysis and a multilingual information retrieval system. The icon analysis module is based on a neural classification model, which aims to interpret the icon's content into a conceptual representation which then can be used to support icon retrieval during localisation. A database containing conceptual representations of icons in different languages is created. SPIRIT, the multilingual information retrieval system they propose, uses the conceptual description of source-language icons in order to find target-language icons with equivalent descriptions.

In the same paper, the authors also examine the internationalisation and localisation of the language-dependent parts of a software application. They describe translation tools and discuss possible scenarios for text translation, using these tools. They separate the translation process into post-edition and pre-edition. The difference between these processes is that, in the latter case, source text is written in a way that facilitates translation. In other words, text is prepared (pre-edited), reducing ambiguities (by the use of standard terminologies and restricted vocabularies and restricted writing rules). The authors recommend this "preparation" of source text to improve the quality of translation and to reduce translation time when using translation tools. In other words, the authors recommend the "text-preprocessing" stage that we apply during software internationalisation as a first step for development of a KB.

Kataoka et al. [23] discuss requirements for developing multilingual computer environments and describe their internationalised computing environment, the System 1. The authors discuss problems in the first level of the internationalisation process: encoding, displaying and handling various characters in the languages of the world. Their study of these problems resulted in a character representation in which all language-dependent features are extracted, then attached to characters as attributes. System 1 is able to convert characters to/from this representation, and to manipulate them. In other words, System 1 can be used to isolate language-dependent aspects at the character level. However, it is still necessary to isolate culture- and user-dependent parts (as well as language-dependent parts) at a higher level, in order to complete internationalisation. System 1 provides a basis upon which our methodology can be built, to test its multilingual capabilities.

5. Current status and future work

In this paper, we presented a methodology for setting up a language-independent KB that can support development of multilingual and user-tailored interfaces. The setting up takes place during software internationalisation of a new or existing software application. The KB uses a conceptual representation to embody structural (domain and application) knowledge and an abstract–concrete hierarchy of function descriptions in
controlled English to represent functional knowledge. In the case of internationalising new
software, text is written in controlled English from the beginning, whereas in the case of
existing software, the function descriptions are rewritten using controlled English. During
software localisation, domain and application concepts are lexicalised in the
target languages, and controlled descriptions are translated automatically to these
languages. The language model and translation memory connect the translated descrip-
tions to the controlled function descriptions. We implemented a prototype system that
includes: (a) a knowledge editor for adding, modifying, and deleting concepts and rela-
tions, and (b) a simple transfer MT module that morphologically and syntactically
analyses controlled English sentences, producing a representation from which target
language sentences are generated.

The proposed methodology was exploited in a case study for generation of multilingual
and user-tailored diagnostic messages. Messages are generated dynamically using inform-
ation from the KB and information about the application context (i.e. tasks performed
and current position at the user interface). We implemented a message-generation proto-
type that is able to produce diagnostic messages in English and Greek, using an NLG
system. This prototype is also able to tailor messages according to a set of user level and
style rules. The prototype includes: (a) the messages manager, which makes the diagnosis,
and (b) the NLG system, which contains strategic and tactical modules.

User-tailoring in our message generation prototype was handled by a very simple
approach. We need to devise a more intelligent user tailoring mechanism. That is why
we intend to integrate the prototype user modeling into the message generation process.
This will make it possible to identify the user’s level of expertise and his interests in the
application domain.

The cultural aspects of the user interface is another issue we need to examine. The
knowledge-based approach proposed by Semmar et al. [22] encourages us to examine
extension of our knowledge representation formalism, in order to represent graphical
symbols (icons corresponding to user interface components).

There are several issues we must still examine. The encouraging results of our case
studies, the results of relevant work in the area of multilinguality, and the market require-
ments for multilingual and user-tailored interfaces motivate us to

- improve automation of the setting up method: add domain and application concepts
  automatically in the case of new software, and extract them automatically from
  software manuals in the case of existing software, thereby implementing a new (or
  existing) controlled English editor/checker;
- integrate user modeling techniques for development of more intelligent user-tailored
  interfaces;
- investigate exploitation of the methodology for generation of other parts of the user
  interface, such as menus, labels, and on-line help;
- evaluate our approach in a variety of complete software applications producing real
  performance results;
- increase the number of languages supported, which will require examination of the the
  lexicalisation problems as well as of differences and commonalities between
  languages; and
• investigate extension of the knowledge representation formalism, in order to be able to embody cultural aspects of user interface components.

References


