Investigating On Line Message Generation in Software Applications: The GLOSSASOFT[1] Results

Constantine D. Spyropoulos, Evangelos A. Karkaletsis, George A. Vouros, Timo Honkela[2], Krista Lagus*, and Aarno Lehtola**

Institute of Informatics & Telecommunications,
National Centre for Scientific Research (N.C.S.R.) "DEMOKRITOS",
153 10 Aghia Paraskevi, Athens, Greece
Tel: +301 6510310, +301 6513110 ext.520, Fax: +301 6532175,
E-mail: costass@iit.nrcps.ariadne-t.gr

*VTT Information Technology
PB 1201, FIN - 02044 VTT, Finland
Tel: +358 0 456 6032, Fax: +358 0 456 6027, E-mail: aarno.lehtola@vtt.fi

Abstract. This paper presents the results of the GLOSSASOFT project in the area of on line message generation in software applications. First, it presents the existing approaches for generating messages and discusses their drawbacks. Then two new approaches aiming to tackle these drawbacks are investigated. The first concerns with the use of extended message templates and the second one with the use of a language independent knowledge base that contains knowledge about the structure and functions of a software application. The two approaches are presented using case studies examples and their costs and benefits are analysed.

1. INTRODUCTION

The adaptation of a software application to the needs and requirements of local markets, which is called localisation, is a very important consideration if companies wish to survive the fierce competition in the growing markets. Software products must "speak" the language of the users. On the other hand, the aim of internationalisation, is to create software products that can be easily localised to several markets.

A major part of software localisation concerns the adaptation of messages to the needs and requirements of the local markets. In an internationalised software application messages are stored, separately from the source code of the software, in message catalogues. In this way, the localisation of messages does not involve the rewriting of the software code but only the adaptation of the message catalogue.

The most common approach for organising messages in a software application is the use of canned messages. Since canned messages are fixed, software developers have to maintain

---

[1] GLOSSASOFT is a project partially supported by the EU under the contract LRE-61003 together with Open University (GB), N.C.S.R. "Demokritos" (GR), Claris (IR), HP Hellas (GR), VTT (Fi), and BULL (Fr).

[2] On leave at Helsinki University of Technology

separate entries in the message catalogue for messages with many similarities. This drawback can be tackled using message templates. These are messages with slots which can be substituted by actual values depending on the context in which each message is generated [Spyropoulos 93]. For example, instead of using four messages to announce that one of the four disk drives of the system is damaged, we can use the message template "Disk <number of disk> is damaged". This message template can also be used to announce that "Disks 1, 2 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[3]). There will be no need to maintain separate entries inside the message catalogue for the messages that have only different word forms.

The use of extended message templates improves the organisation of message catalogues. However, we still have to maintain different extended message templates for the different languages supported. It would help a lot if we could generate the messages for all the different languages from one common language-independent ("interlingual") representation by calling the appropriate syntactic and morphological generation routines.

These problems motivated us to investigate the on-line message generation in software applications. The paper describes two different approaches examined in the context of two restricted case studies of the GLOSSASOFT project [Spyropoulos 94b]. The first approach considers the combined use of message templates and morphological generation, that is the use of extended message templates. The second approach considers the dynamic on-line generation of messages from a language independent representation using knowledge bases and natural language generation techniques.

Section 2 presents the method for generating messages in a software application using extended message templates. Section 3 describes the method for generating multilingual diagnostic messages by exploiting knowledge of the software application. Finally, section 4 summarises the results of our work and describes our future plans.

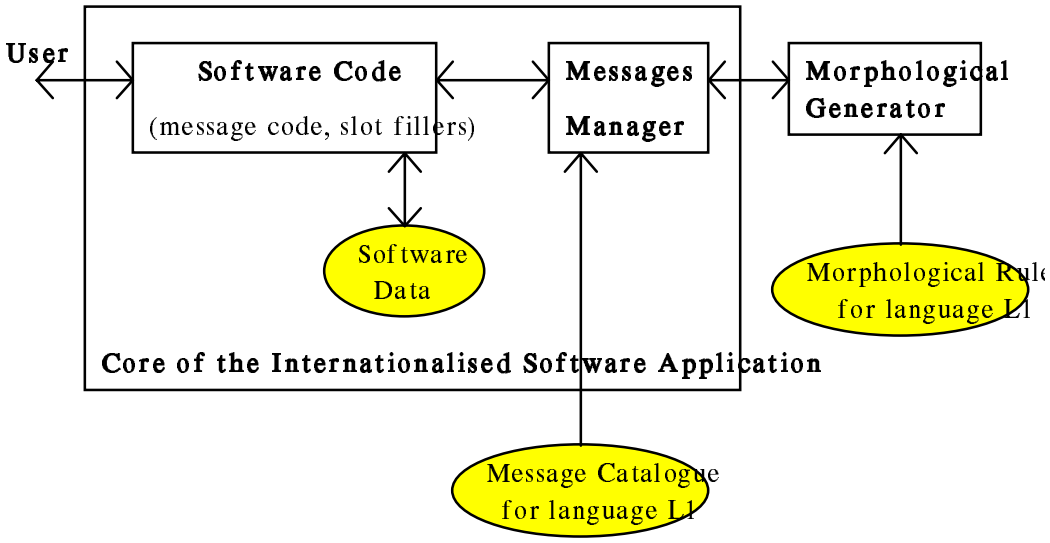
2. ON-LINE GENERATION OF MESSAGES USING EXTENDED MESSAGE TEMPLATES

In our first case study we investigated the use of extended message templates. We proposed an architecture for internationalised software applications (see Fig. 1) that is based on the use of message catalogues of extended message templates and the use of morphological generation routines.

---

3 The complexity of Finnish morphology can be described by stating that a single noun has almost 2000 potential inflectional word forms and each verb over 10000

Figure 1. On-line generation of messages using extended message templates



According to this architecture, the core of the software contains the source code, the software data and the Messages Manager. The message catalogue with the extended message templates for the language currently supported (L1) 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 the lexemes that will substitute the slots in the message template. The basic idea in this architecture is the use of a Morphological Generator for inflecting the set of lexemes to their proper forms.

The software source code asks for the appropriate message from the Messages Manager giving the message template code and the slot fillers. The Messages Manager retrieves the corresponding message template and sends the lexemes (from the slot fillers) and their morphological features (from the message template) to the Morphological Generator. The Morphological Generator inflects the lexemes to their proper forms, using the morphological rules for language L1, and returns them to the Messages Manager. The inflected lexemes will substitute the slots in the message template , forming the message that will be presented to the user.

2.1. The OsiCon Case Study

During this case study a message generation system was implemented for the system OsiCon [Spyropoulos 94b]. OsiCon Form Designer (OcForm/D) and OsiCon Form Filler (OcForm) are platform independent GUI based form interfaces, that belong to VTT Finland [Honkela 94].

VTT used a Finnish morphological generator named FINGEN (FINnish GENerator). FINGEN is a trademark of Lingsoft Ltd [Linsoft 94]. The input provided to FINGEN includes the lexemes and their morphological feature specifications. In the actual input, the basic word form and the features are separated with dashes. The output is the word in its proper form. For example, some input-output pairs for the Finnish word 'katu' (a street) are as follows:

Input	*Output*
katu-nom-sg	"katu"
katu-gen-sg	"kadun"

where the morphological feature name "nom" stands for nominative, "sg" for singular and "gen" for genitive.

We will exemplify this approach using two messages generated by OsiCon using our message generation system. This system is able to generate messages in Finnish using FINGEN. However, apart from the messages in Finnish we also provide the corresponding messages in English and Greek in order to demonstrate the potentials of extended message templates.

In order to support the generation of messages in Finnish, English and Greek we need three different message catalogues of message templates and three different sets of morphological rules one for each language.

The first message in the three languages appears as:

```
Tietojen lähettäminen Vangelikselle onnistui.

Sending information to Vangelis succeeded.

Η αποστολή πληροφορίας στο Βαγγέλη πέτυχε.
```

The corresponding message templates in the three message catalogues are:

```
defTxt(SEND_INFORM,
  "Tietojen lähettäminen %s(1,fin,all-sg) %s(2,fin,past-act-sg3).");
defTxt(SEND_INFORM,
  "Sending information to %s(1,eng,nom-sg) %s(2,eng,past-act).");
defTxt(SEND_INFORM,
  "Η αποστολή πληροφορίας στο %s(1,gre,acc-sg) %s(2,gre,past-act-
sg3).");
```

where SEND_INFORM is the message_template code and %s represents the slot that must be filled by a lexeme with the morphological features included in the parentheses. The morphological feature name "sg" stands for singular, "all" for the allative case, "nom" for the nominative case, "past" for past tense, "act" for active voice, "sg3" for 3rd singular person and "acc" for the accusative case. The language feature name "fin" stands for Finnish, "eng" for English and "gre" for Greek. The numbers "1" and "2" specify the order of the slot fillers.

The OsiCon system is internationalised and has been designed for changing the locale[4] of a form dynamically [Honkela 94]. Let's say that OsiCon currently operates in Finnish. This means that the Finnish message catalogue and the Finnish set of morphological rules are active. The source code invokes Messages Manager giving the message template code (SEND_INFORM) and the two slot fillers (lexemes). The first lexeme is the name "Vangelis" and the second is the verb "onnistua" (succeed). Messages Manager invokes the Morphological Generator using the function "m-generator" with the two lexemes and their morphological features:

---

4 The collection of linguistic and cultural aspects for a specific language and region.

```
m-generator(fin,"Vangelis","all-sg")

m-generator(fin,"onnistua","past-act-sg3")
```

The Morphological Generator (FINGEN) inflects the lexemes, using the morphological rules for Finnish, and returns them to Messages Manager. The inflected lexemes "Vangelikselle" and "onnistui" respectively substitute the slots in the Finnish message template, forming the final message.

The second message in the three languages appears as:

```
OVT-sanomassa ei ole lähettäjän osoitetta.

The EDI message does not contain the address of the sender.

Ôï EDI ìÞíõìá äåí ðåñéÝ÷åé ôç äéåýèõíóç ôïõ áðïóôïëÝá.
```

The corresponding message templates in the three message catalogues are:

```
defTxt(DATA_NOT_EXIST,

  "OVT-sanomassa ei ole %s(2,fin,gen-sg) %s(1,fin,acc-sg).");

defTxt(DATA_NOT_EXIST,

  "The EDI message does not contain the %s(1,eng,nom-sg) of the
%s(2,eng,nom-sg).");

defTxt(DATA_NOT_EXIST,

  "Tï EDI ìÞíõìá äåí ðåñéÝ÷åé ôç %s(1,gre,acc-sg) ôïõ %s(2,gre,gen-
sg).");
```

where DATA_NOT_EXIST is the message template code and "gen" stands for the genitive case. Note that in the Finnish message template the 2nd slot filler ("lähettäjän" which means "the sender" in the accusative case) goes first, according to Finnish syntax rules. In the English and Greek message templates the order of the slot fillers is followed.

Again, the source code invokes Messages Manager giving the message template code (DATA_NOT_EXIST) and the two slot fillers (lexemes). The first lexeme is "osoite" (address) and the second is "lähettäjä" (sender). The function "m-generator" with the two lexemes and their morphological features is invoked by the Morphological generator as follows:

```
m-generator(fin,"osoite","acc-sg")

m-generator(fin,"lähettäjä","gen-sg")
```

The inflected lexemes "lähettäjän" and "osoitetta" respectively substitute the slots in the Finnish message template, forming the final message.

2.2. Conclusions

The use of extended message templates is specifically suitable for synthetic languages such as Finnish. Software developers and localisers need not to maintain all the message templates that have morphological differences. It is enough to extend the message templates in order to include calls to the appropriate morphological generation routines. Two features must be included in each extended message template to achieve this: the language and the linguistic specification. This approach is feasible since the components for its realisation are readily available and can be exploited easily.

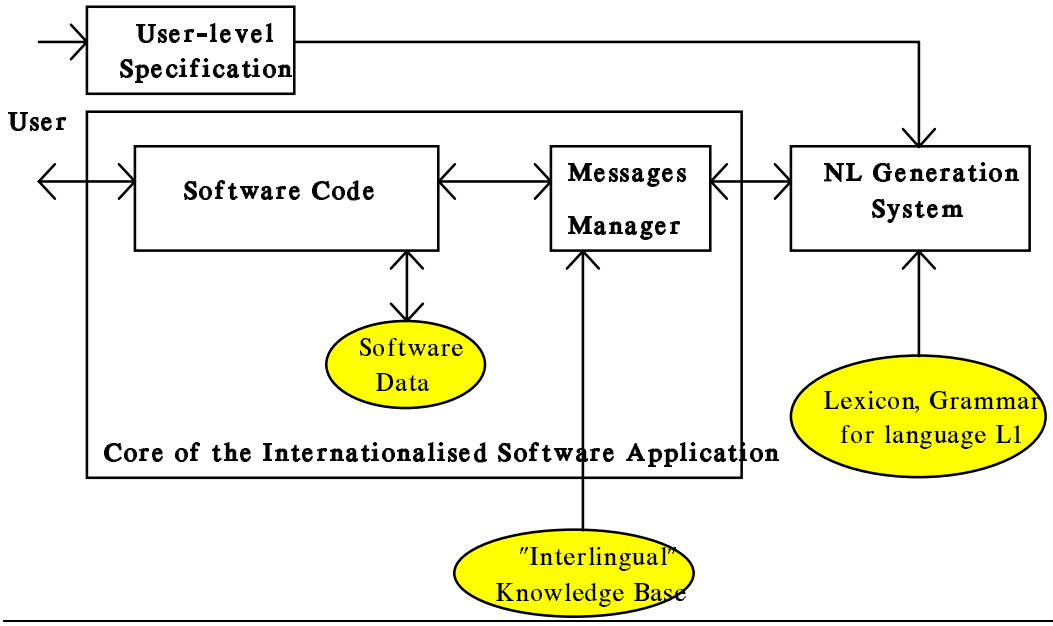
If the software application supports three different languages, then there will be three different extended message templates, one for each language. It would be advantageous to create

message catalogues of internationalised message templates. That is, one message template for all the languages supported. However, it is extremely difficult to specify such internationalised message templates due to the large number of syntactic and morphological differences of the various languages (we have already seen this in the simple messages of the previous section). Note also that neither the extended nor the internationalised message templates can be used to generate the same message in different ways according to the user knowledge, tasks, plans, etc. In the following section we describe an advanced knowledge-based approach to achieve such goals.

3. DYNAMIC ON-LINE GENERATION OF MESSAGES FROM AN "INTERLINGUAL" KNOWLEDGE BASE

In our second case study NCSR "Demokritos" investigated the combined use of knowledge bases and natural language generation techniques for on-line message generation in software applications. We proposed an architecture (see Fig. 2) to generate messages on-line for different languages from a common language-independent representation, as well as to express the same message in a language in different ways, according to the user needs and the required style.

Figure 2. Dynamic on-line generation of messages from an "Interlingual" Knowledge Base



According to this architecture, we do not have to maintain a message catalogue. Messages are generated dynamically using the knowledge for the software functions and components that is stored in the language independent ("interlingual") KB. Each time a message must be generated, the source code invokes the Messages Manager, which uses the current context of the software along with the information of the KB to generate a language-independent representation of the appropriate message. This representation is then passed to the NLG system. The NLG system decides first on what sort of information from the language-independent representation will be presented to the user. According to the user level of experience, different information can be extracted from the KB (more detailed for the inexperienced user and less detailed for a more experienced one). The NLG system then translates this information into the user's language using the appropriate lexicon and grammar.

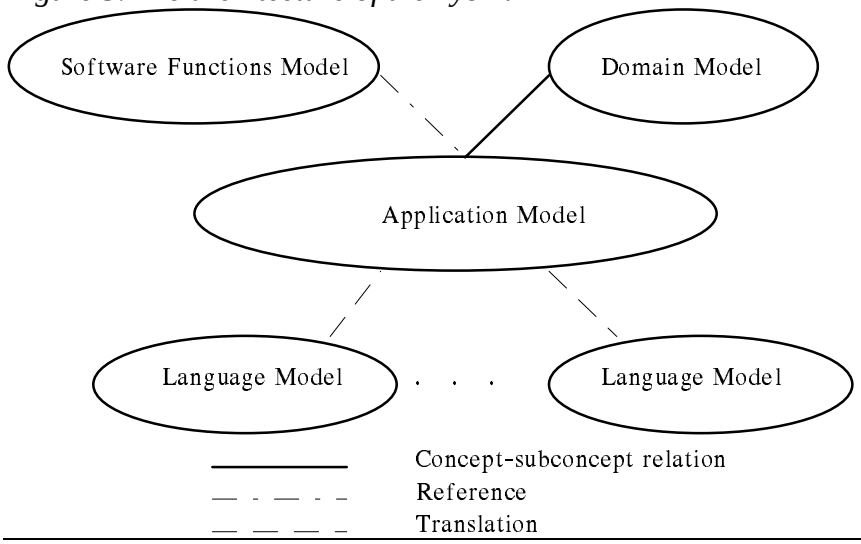
This approach requires a knowledge representation (KR) system for representing and managing the knowledge stored in the KB, and an NLG system for generating messages from the KB in the user's language and according to user experince, tasks, plans, etc.

Unfortunately, as it is the case with all knowledge-based approaches [Reiter 93], the knowledge-based on-line message generation has disadvantages related to the:

- **cost of setting up the KB**. The set up of a KB involves the acquisition of knowledge concerning the software functions and components as well as the effective organisation of this knowledge into the KB. This is a costly process and thus the main reason for not using KBs in practice.
- **cost of managing the KB**. The management of the KB involves the introduction of new knowledge, the modification of the existing one, the consistency checks, as well as the retrieval of knowledge. A complex knowledge representation formalism and an ineffective organisation increases this cost significantly.

To reduce these costs we need an effective organisation of the knowledge and a simple KR formalism. This remark along with our experience working with Terminological KBs (TKBs) [Karkaletsis 95a,b,c] led us to investigate the use of TKBs in message generation [Karkaletsis 94,95c, Spyropoulos 94b]. TKBs are KBs containing terminological knowledge for the software application, that is general domain knowledge (the ontology upon which the TKB is built), knowledge on the software functions (functional knowledge) and knowledge on the software components (structural knowledge) [Karkaletsis 95a,c]. We examined methods and techniques that improve organisation of TKBs facilitating their set up and management. Our efforts resulted in an innovative hybrid architecture for TKBs [Karkaletsis 95a,c], a simple knowledge representation formalism [Karkaletsis 95c], an effective KR system [Karkaletsis 94,95c] and an effective method for setting up such a TKB [Karkaletsis 95b,c]. This architecture is depicted in Fig. 3.

Figure 3. The architecture of the hybrid TKB



The main components of this architecture have the following characteristics:

(a)The *Domain Model* contains general domain knowledge. Domain entities are represented as concepts (domain concepts).

(b) The *Software Functions Model* contains knowledge on software tasks. This is the controlled and marked up textual descriptions of the software tasks and the steps needed to perform them. These descriptions are extracted from the help texts, organised and labelled. They are then translated in the application languages supported, forming a translation memory. This translation memory is the Software Functions Model.

(c) The *Application Model* contains application-specific terminological knowledge. Application-specific aspects are represented as concepts (application concepts) that are classified under the concepts of the Domain Model, through the concept-subconcept relation. Those concepts that correspond to a specific task of the software application contain a reference to this task in the Software Functions Model. This reference is the task's label.

(d) The *Language Model* contains language-specific terminological knowledge. We use local concepts to express this knowledge. The local concepts are classified under the domain and application concepts inheriting their language independent characteristics.

We decided to investigate the use of hybrid TKBs in dynamic on-line generation of diagnostic messages. This work is presented in the next section.

3.1. The Style Manager Case Study

Style Manager of the HP VUE system was used to examine this approach in the second case study. HP VUE 3.0 is a Windows Management System and User Interface, built by Hewlett Packard over X11R5 Windows for the HP-UX 9.00 operating system series. Style Manager is the application that enables users to change the appearance and behaviour of the HP VUE screen colours, sound, keyboard, mouse, windows and sessions [Spyropoulos 94a].

To set up the hybrid TKB, we used the set up methodology of [Karkaletsis 95b,c]. According to its principles we rewrote the Style Manager help text using the rules of a controlled English language. The help text was then marked up using a prespecified set of mark up tags. This tag set includes mark ups for tasks, glossaries terms, domain concepts, application-specific concepts, generic and partitive relations, concepts attributes.

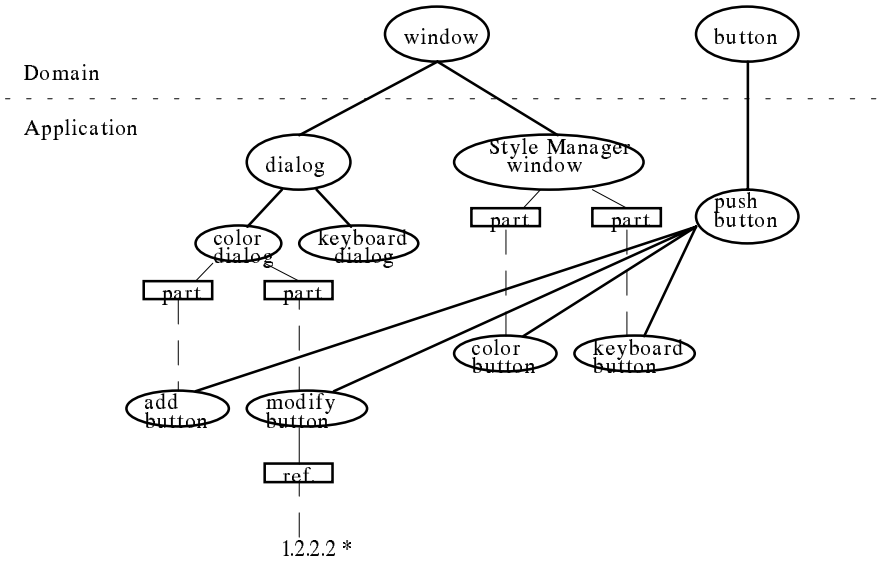
A controlled sample of Style Manager help text is depicted in Fig. 4. It concerns the tasks required to "modify a palette".

Figure 4 Sample of Style Manager help text

12 Modify a palette.
121 Select a palette from the palettes-list.
122 Open the modify-dialog. 1221 Click a colour-button. 1222 Click the modify-button.
123 Adjust the settings.
124 Choose the ok-button in the modify-dialog.
125 Choose the ok-button in the colour-dialog

A part of the Application Model is depicted in Fig. 5. The concept "modify button" is part of the "colour-dialogue" and is linked with the task 1.2.2.2 of the Software Functions Model (see Fig. 4), through the "ref" attribute.

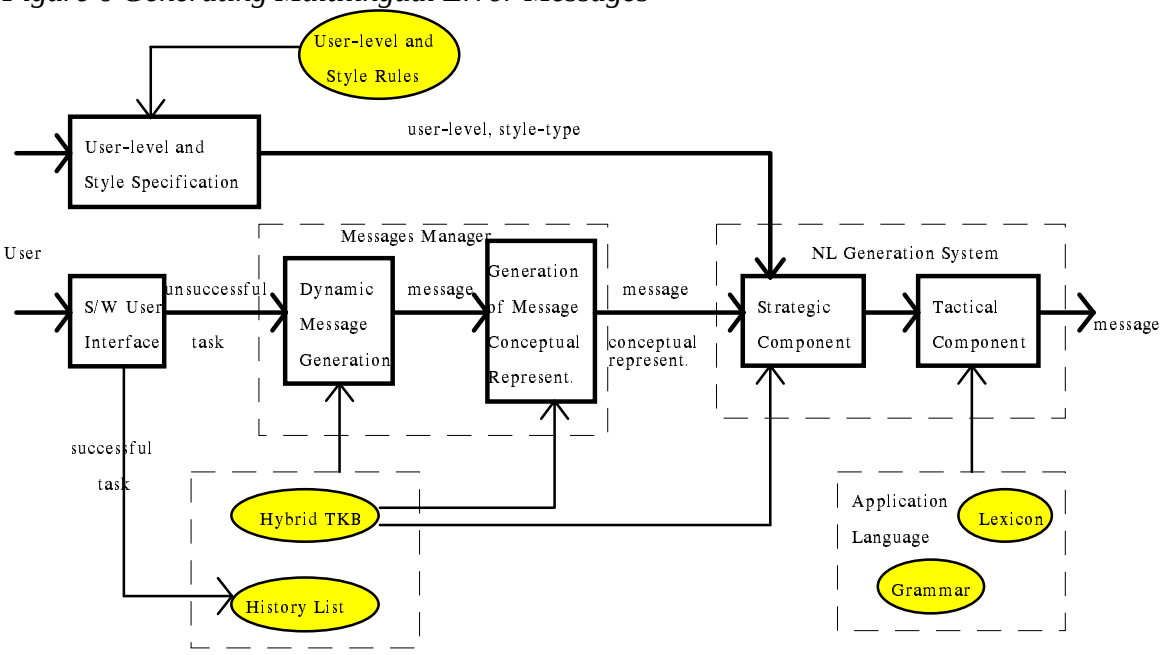
Figure 5 Part of the Application Model.



* These numbers correspond to tasks labels in the S/W function model

We implemented a prototype message generation system that generates diagnostic messages in English and Greek, every time the user performs an unsuccessful task. The basic steps of message generation, as they are realised by this prototype, are depicted in Fig. 6.

Figure 6 Generating Multilingual Error Messages



(a) User level and style specification. 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 messages for the same situation can be

generated. To exemplify this we defined four different user levels of expertise and four different styles. The user levels with their descriptions are:

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

The different styles with their descriptions are:

Style	Description
s1	Imperative (e.g. Click a colour-button)
s2	Have to (e.g. You have to click a colour-button)
s3	Must (e.g. A colour-button must be clicked)
s4	Did not (e.g. You did not click a colour button)

(b) History-List update. The History-List maintains the information on the successfully performed tasks. Every time the user performs a task (selects a menu option, presses a button, etc.), the History-List is updated. Whenever a task cannot be performed, the Conceptual Message Generation module is invoked.

Let's assume that the user is in the colour dialogue and wants to modify a colour of the currently selected palette. As shown in Fig. 4, to modify a colour of the palette the user must first select the palette (task 1.2.1.), 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.), where the appropriate adjustments can be made (task 1.2.3.). Let's 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 Conceptual Message Generation module will be activated to produce a message using the information contained in the history list and the knowledge base.

(c) Dynamic Message Generation. The message that corresponds to the unsuccessful task is not obtained from a message catalogue. It is generated dynamically using the information of the KB and the History-List.

In the previous example, the error happened when the user attempted to click the modify-button. The Conceptual Message Generation module uses this information to find the corresponding concept in the Application Model. It finds the concept "modify-button" which has the attribute "ref" with value "1.2.2.2" (see Fig. 5). It then searches in the Software Functions Model for the 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 Conceptual Message Generation module detects the error. That is, it finds that the task 1.2.2.1. had to be performed first.

(d) Generation of the Message Conceptual Representation. This component takes the diagnosis and translates it into the knowledge representation formalism. This representation is

passed to the NL Generation module in order to produce the final message to the language of the user.

The conceptual representation of the diagnostic message of the previous example is:

precondition(1.2.2.,[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 the task 1.2.2., the user must perform first the task 1.2.2.1.

(e) Generation of the final message. The NL Generation module consists of two sub-modules: the strategic and the tactical component. The strategic component takes as input the conceptual representation of the message together with the user level and style and decides on what will be displayed to the user. The tactical component takes this as input together with the grammar and lexicon and for the target language and produces the final message.

Let's 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 the content determination, the strategic component has to 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 s1, the strategic component decides that the message will be in imperative.

The tactical component takes as input the previous, the current and the final task which must be included in the final message. It searches for the translations of these tasks in the translation memory of the Software 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:

TASK	Greek Translation	Verb	Noun Phrase
previous task	1.2.2.1 ÐáôÞóôå ôï êïõìðß-÷ñþìá	ðáôþ	ôï êïõìðß-÷ñþìá
current task	1.2.2.2 ÐáôÞóôå ôï êïõìðß-ôñïðïðïßçóç	ðáôþ	ôï êïõìðß- ôñïðïðïßçóç
final task	1.2.2 Áíïßîôå ôï äéÜëïãï-ôñïðïðïßçóç	áíïßãù	ôï äéÜëïãï-ôñïðïðïßçóç

Using these verbs and noun phrases, the tactical component activates the grammar rule for imperatives, converts the words in the appropriate morphological forms (using the morphological rules of the language) and generates the final message. 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 different styles and for user-level equal to 4.

Style	Message in English	Message in Greek
s1	Click a colour-button first and then click the modify-button to open the modify-dialog.	Áñ÷éêÜ ðáôÞóôå Ýíá êïõìðß-÷ñþìá êáé óôç óõíÝ÷åéá ðáôÞóôå ôï êïõìðß-ôñïðïðïßçóç ãéá íá áíïßîåôå ôï äéÜëïãï-ôñïðïðïßçóç.
s2	You have to click a colour-button first and then click the modify-button to open the modify-dialog.	Áñ÷éêÜ ðñÝðåé íá ðáôÞóåôå Ýíá êïõìðß-÷ñþìá êáé óôç óõíÝ÷åéá íá ðáôÞóåôå ôï êïõìðß-ôñïðïðïßçóç ãéá íá áíïßîåôå ôï äéÜëïãï-ôñïðïðïßçóç.
s3	A colour-button must be clicked first. Then, click the modify-button to open the modify-dialog.	Áñ÷éêÜ ðñÝðåé íá ðáôçèåß Ýíá êïõìðß-÷ñþìá. Óôç óõíÝ÷åéá ðáôÞóôå ôï êïõìðß-ôñïðïðïßçóç ãéá íá áíïßîåôå ôï äéÜëïãï-ôñïðïðïßçóç.
s4	You didn't click a colour-button. Click a colour-button first and then click the modify-button to open the modify-dialog.	Äåí ðáôÞóáôå Ýíá êïõìðß-÷ñþìá. Áñ÷éêÜ ðáôÞóôå Ýíá êïõìðß-÷ñþìá êáé óôç óõíÝ÷åéá ðáôÞóôå ôï êïõìðß-ôñïðïðïßçóç ãéá íá áíïßîåôå ôï äéÜëïãï-ôñïðïðïßçóç.

3.2. Conclusions

The advantages offered by the combination of KR and NLG techniques according to [Reiter 93] include maintainability, multilinguality, enforcement of syntax and style rules and adaptabilty. The investigation of such an approach for the on-line dynamic generation of messages in software applications led us to the same conclusions:

- **Maintainability**. Lower cost for the development of a new version and the creation of new local versions. Message generation in these versions would involve the update of the KB and the lexicons of the local languages.
- **Multilinguality**. Since the KB is language-independent, then the NLG system can generate the same message in more than one language using the grammar, lexicon and morphological rules for the supported language.
- **Control of the style and content of the messages.** Message generation from an NLG system permits the enforcement of specific rules for syntax, content and style.
- A**daptability.** Messages can be adapted according to the user level of experience, user tasks and plans. This adaptation concerns the use of different syntax, vocabulary, and more or less detail in the message content.

On the other hand the disadvantages concerning the cost for setting up and managing the KB are tackled through the use of the hybrid TKB that contains the structural and functional knowledge of the software application [Karkaletsis 94,95a,95b].

4. SUMMARY AND FUTURE WORK

In the framework of the GLOSSASOFT project we investigated on-line message generation in software applications. We examined two new approaches in two separate case studies. The first approach concerned the use of extended message templates. A message generation system for Finnish was developed for the system OsiCon. The message templates were extended to include two features: the language and the morphological features of the lexemes that will substitute the slots of the message template. We used the morphological generator FINGEN for inflecting the lexemes according to their morphological features. Such an approach is very useful especially for synthetic languages since it reduces the number of entries in the message catalogues, improving their organisation and facilitating the localisation of messages.

Two interesting extensions of this work that we would like to investigate in the future are the enhancement of the message generation system to generate phrasal constituents and the development of a translation memory system of extended message templates. The first extension will improve even more the organisation of message catalogues, while the second one will help technical writers to express messages uniformly across new software versions and products.

It would be advantageous to generate the same message in different languages and ways according to the user experience, tasks, plans, etc. The generation of messages from an "interlingual" KB that contains the functional and structural knowledge of the software application is an approach that can help to achieve these goals. We examined this approach in the second case study. A message generation system for English and Greek was developed for the Style Manager of the HP VUE system. This system does not require to maintain message catalogues. Messages are generated dynamically using the information from the KB and the information on the application context (i.e. tasks performed, current position at the user interface). The message generation system is able to generate diagnostic messages in English and Greek using an NLG system along with the corresponding lexicons and grammars. It is also able to adapt the messages according to a set of user level and style rules that we specified in order to exemplify our approach. To reduce the cost of this knowledge-based approach we used a hybrid TKB, a simple and effective KR system and a cost-effective method for setting up TKBs.

We intend to evaluate the performance of the message generation system in a complete software application. This will probably lead us to modify the KR system, the method of TKB set up and the NLG system in order to make the message generation system more robust. Further we are going to investigate the effectiveness of this approach for the generation of other types of messages apart from the diagnostic ones, as well as for the introduction of other languages in the NLG system. Another issue that needs to be investigated is the specification of an intelligent user modelling. We also believe that this approach can be used for the generation of multilingual interfaces in software applications.

REFERENCES

[Honkela 94] Honkela, T., Kalliomaki, S., Lagus, K. 1994. GLOSSASOFT Deliverable 14.1 "VTT Case Study", v1.0, July 1994.

[Linsoft 94] Linsoft Ltd., 1994. "FINGEN Reference Manual".

[Karkaletsis 94] Karkaletsis, E., Spyropoulos, C.D., Vouros, G. 1994. A Knowledge-based Approach for Organising Terminological Data and Generating Messages in Software Applications. In Proceedings of the "Language Engineering on the Information Highway" Workshop, Santorini, 26-30 September 1994.

[Karkaletsis 95a] Karkaletsis, E., Spyropoulos, C.D., Vouros, G. 1995. The Use of Terminological Knowledge Bases in Software Localisation. In Lecture Notes of Artificial Intelligence (LNAI), no 898, pp. 175-188.

[Karkaletsis 95b] Karkaletsis, E., Spyropoulos, C.D., Vouros, G., Halatsis, C. 1995. Organisation and Exploitation of Terminological Knowledge in Software Localisation. In TermNet News - Journal of the International Network for Terminology, no 48, pp. 42-48.

[Karkaletsis 95c] Karkaletsis, E., 1995. Terminological Knowledge Bases and their Exploitation in Multilingual Software Applications. Ph.D. Thesis. Department of Informatics, University of Athens, March 1995 (in Greek).

[Reiter 93] Reiter, E., Mellish, C., 1993. Optimising the Costs and Benefits of Natural Language Generation. In Proceedings of IJCAI 1993, pp. 1164-1169.

[Spyropoulos 93] Spyropoulos, C.D., Karkaletsis, E., Vouros, G., 1993. GLOSSASOFT, Deliverable 4.1 "Guidelines for Linguistics for Interaction Interlinguality", v1.0, December 1993.

[Spyropoulos 94a] Spyropoulos, C.D., Karkaletsis, E., Kokkotos, S., Vouros, G., 1994. GLOSSASOFT, Deliverable 8.1 "HP Case Study", v1.0, July 1994.

[Spyropoulos 94b] Spyropoulos, C.D., Vouros, G., Kokkotos, S., Karkaletsis, E., Honkela, T., Lagus, K., 1994. GLOSSASOFT, Deliverable 10.1. "Guidelines for Linguistics", v1.0, December 1994.