Message Generation

This chapter provides information on the challenges of message generation in an internationalized product, and presents two case studies undertaken by members of the Glossasoft project.

The chapter is divided into three sections:

- Section 11.1 introduces the tasks and challenges of message generation for an internationalized product.
- Section 11.2 provides details of a case study undertaken by VTT, Finland on the use of morphological synthesis in generating messages using a message template.
- Section 11.3 presents a message generation research prototype designed by NCSR 'Demokritos', Greece, which combines natural language generation techniques with knowledge base approaches.

11.1 INTRODUCTION

Message generation is concerned with:

- Producing messages from the application for display to the user. In an internationalized software application you should have placed these messages in separate files—message catalogs or message databases—from which you can extract and translate them easily during localization.
- Translating messages to the target languages. This is made easier if you use a translation memory that contains already translated messages from previously localized versions. You may also have implemented a multilingual
terminological database, which will then contain translations of some of the terms contained in messages.

How can language technology improve message generation so that localization is easier? A first approach would be to reduce the size of the message catalog, using message templates. These are messages with slots that can be filled with actual values depending on the context in which each message is generated. For example, instead of using four messages to announce that one of the four disk drives of the system is damaged, you can use the message:

Disk <number of disk> is damaged

However, if you want to use the same message template to display a message such as:

Disks 1, 2 are damaged

you need to be able to take account of the grammatical rules and inflections of the language. Similar problems occur if you wish to combine messages.

Generating messages whose form varies depending on the words to be slotted into them can be achieved by using an extended message template system that has access to morphological synthesis routines. This is particularly helpful for languages with many inflectional word forms, such as Finnish, as it avoids the need to maintain all the different word forms inside the message catalog. It also improves the organization of message catalogs.

But what about translation of messages into other languages? Is it possible just to translate the message templates into the target languages to build the corresponding local message catalogs? Can you use such a catalog of message templates as the internationalized message catalog from which the application produces messages in the target languages? The answer to these questions is 'no', since there are so many morphological and syntactic differences between languages. For example, say that the template:

Disk is <> active

is used to generate both the messages:

Disk is active

Disk is not active

It is not enough just to translate the template in French, for example, because negation in French is expressed in a different way:

Le disque n'est pas actif

—and this is just a simple example. You would therefore have to redesign the message templates for the different languages supported.

It would help a lot if you could generate messages for all required languages from one common language-independent form by calling the appropriate syntactic and morphological generation routines. We have therefore investigated an approach that combines knowledge bases with natural language generation techniques. It also exploits the use of controlled languages and translation memories.

The following sections investigate the use of extended message templates and the use of knowledge bases for message generation in two separate case studies, realized by VTT and Demokritos respectively. The VTT case study considers the use of message templates with morphological generation techniques. They discuss the generation of messages in English and in Finnish from the software product OsiCon. Demokritos's case study in the following section concerns the knowledge-based generation of different messages in English and Greek according to a specific context, the needs of the user and the style required, using Hewlett Packard's VUE software.

11.2 MESSAGE TEMPLATES AND MORPHOLOGICAL GENERATION

As part of the Glossasoft Project, VTT proposed an architecture for internationalized software applications that is based on the use of message catalogs of extended message templates and the use of morphological generation routines.

According to this architecture, the core of the software application contains the source code, the software data and the Messages Manager. The message catalog with the extended message templates for the language currently supported 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.

11.2.1 The OsiCon Case Study

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

VTT used a Finnish morphological generator named FINGEN (FINnish GENerator). FINGEN is a trademark of Lingsoft Ltd. 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:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>katu + gen</td>
<td>katu-gen</td>
</tr>
<tr>
<td>katu - gen</td>
<td>katu</td>
</tr>
<tr>
<td>katu:gen</td>
<td>katu:gen</td>
</tr>
<tr>
<td>katu-gen</td>
<td>katu-gen</td>
</tr>
<tr>
<td>katu-gen:gen</td>
<td>katu-gen</td>
</tr>
</tbody>
</table>

The output is the word in its proper form.
where the morphological feature name 'nom' stands for nominative, 'sg' for singular and 'gen' for genitive.

The message generation system for OsisCon is able to generate messages in Finnish using FINGEN. However, apart from the messages in Finnish the corresponding messages in English and Greek are also provided in order to demonstrate the potentials of message templates.

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

The first message in the three languages appears as:

Tietojen lähetettäminen Vangelikselle onnistui.

Sending information to Vangelis succeeded.

Η απόστολη πληροφοριών στο Βαγγέλη συνήκε.

The corresponding message templates in the three message catalogs are:

```python
def txt(SEND_INFORM,
    "Tietojen lähetettäminen %s(1,fin,all-sg)
    %s(2,fin,past-act-sg3).";
)
def txt(SEND_INFORM,
    "Sending information to %s(1,eng,nom-sg)
    %s(2,eng,past-act).";
)
def txt(SEND_INFORM,
    "Η απόστολη πληροφορίας στον %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, 'acc' for 3rd singular person and 'gen' 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 OsisCon system is internationalized and has been designed for changing the locale of a form dynamically. Let's say that OsisCon currently operates in Finnish. This means that the Finnish message catalog 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:

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:

OVIT-sanomassa ei ole lähetettäjän osoitetta.

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

Το EDI μήνυμα δεν περιέχει το δείγμα του αποστολέα.

The corresponding message templates in the three message catalogs are:

```python
def txt(DATA_NOT_EXIST,
    "OVIT-sanomassa ei ole %s(2,fin,gen-sg) %s(1,fin,acc-sg).";
)
def txt(DATA_NOT_EXIST,
    "The EDI message does not contain the %s(1,eng,nom-sg) of the %s(2,eng,nom-sg).";
)
def txt(DATA_NOT_EXIST,
    "Το EDI μήνυμα δεν περιέχει το %s(1,gre,acc-sg) του %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 ('lähetettäjän' 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 'lähetettäjä' (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,"lähetettäjä","gen-sg")

The inflected lexemes 'lähetettäjän' and 'osoitetta' respectively substitute the slots in the Finnish message template, forming the final message.
11.2.2 Conclusions

The use of extended message templates is specifically suitable for synthetic languages such as Finnish. Software developers and localizers 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 realization 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 catalogs of internationalized message templates. That is, one message template for all the languages supported. However, it is extremely difficult to specify such internationalized message templates due to the large number of syntactic and morphological differences of the various languages (this has already been shown in the simple messages of the previous section). Note also that neither the extended nor the internationalized 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 an advanced knowledge-based approach that aims to achieve such goals, is described.

11.3 MESSAGE GENERATION FROM KNOWLEDGE BASES USING NATURAL LANGUAGE GENERATION TECHNIQUES

This section examines an approach that aims to overcome the drawbacks of message templates. Specifically this approach aims to:

- Generate messages for different languages from a common language-independent interlingual representation
- Express the same message in a language in different ways, according to the application context, the user's needs and the required style

The need to generate messages from a language-independent representation leads to the use of knowledge-based natural language generation techniques. In these techniques you first represent the knowledge of a specific domain in a language-independent knowledge representation form. This knowledge is then stored into a knowledge base. Whenever an application asks for information from the knowledge base, the natural language generation module is activated. This decides first on what sort of information is needed from the knowledge base, and then translates this information into the user's language using the appropriate syntactic and morphological generation routines.

The use of knowledge bases and natural language generation techniques for message generation offers the advantage of enabling a developer to express the same message in different ways. For example, according to the user’s level of expertise, different information can be acquired from the knowledge base—more detailed information for the inexperienced user and less for the expert.

Using knowledge bases and natural language generation techniques can help achieve flexibility in multilingual message generation. However, the cost of setting up a knowledge base is large, as it must hold the functional and structural knowledge of the software application. This cost can be reduced by combining the knowledge base with one for terminological work (see Chapter 10), and by using an automated or semi-automated method for acquiring the domain knowledge.

Such a method is described below in Section 11.3.3 and is based on the preprocessing of help texts using controlled and mark-up languages. The aim of using controlled languages is to control the structure, syntax and vocabulary of the help texts; then, using a mark-up language, technical authors indicate those parts of the text that are necessary for the acquisition of terminological knowledge. The preprocessing of help texts eases the acquisition of terminological knowledge, and therefore reduces the development cost of the knowledge base. This knowledge base is then used by the natural language generation module to generate multilingual messages that can be expressed in different ways according to the application context, the user's needs and the style selected.

The GlossaSoft Project examined the use of knowledge bases containing terminological knowledge for generating multilingual error messages when the user attempts to perform a task without success. The following sections present the architecture of the knowledge base, the method for generating multilingual error messages, and an example of the application of the approach using Hewlett Packard's VUE software for generating error messages in English and Greek.

11.3.1 The architecture of the knowledge base

Our method for automatically acquiring terminological knowledge leads to a hybrid architecture for the organization of the knowledge base that mixes conceptual structures representing knowledge of the domain and the application with textual descriptions (taken from marked-up controlled language phrases and sentences) that represent knowledge of the functions of the application.

Figure 11.1 shows the architecture of the knowledge base.
The main components of this architecture are:

- **Domain model** The domain model contains general domain knowledge. Domain entities are represented as concepts.

- **Software function model** The software function model contains knowledge about software tasks. This is the controlled and marked-up textual descriptions of the software tasks. These descriptions are translated in the languages to be supported, forming a translation memory.

- **Application-specific model** The application-specific model contains application-specific terminological knowledge. Application-specific aspects are represented as concepts that are classified under the concepts of the domain model through the concept-subconcept link. Those concepts that correspond with a specific task in the application contain a reference to the corresponding task in the software function model. This reference is the task's label. The link between the software function and the application-specific models in the figure makes this reference.

- **Language specific model** The language-specific model contains language-specific terminological knowledge.

11.3.2 A method for generating multilingual error messages

We developed a method for generating multilingual error messages based on this hybrid architecture. The basic steps of the method are shown in Figure 11.2 and described below.

- **User level and style specification** Whenever a user starts a new session with the application, they can set their level of expertise and the style of message they prefer. According to the user level and requested style, the application can generate different messages for the same situation.

- **History list update** The history list maintains information on successfully performed tasks. Every time the user performs a task—selects a menu option, presses a button and so on—the history list is updated. Whenever a task cannot be performed, the conceptual message generation module is invoked.

- **Generation of the conceptual message representation** The message that corresponds to the unsuccessful task is not obtained from a message catalog, but is generated dynamically using the information from the knowledge base and the history list. This is done using the user interface component that is relevant to that task (a button, menu item and so on). The corresponding concept is retrieved from the hybrid knowledge base. Based on the information for that concept (its reference attribute, generic and partitive relations) and also on the knowledge from the history list and the software function model, the application translates the message into the formalism of the knowledge representation. This conceptual representation of the error message is then passed to the natural language generation module to produce a message in the application language, according to the user level and style.

- **Generation of the final message** The natural language generation module consists of two submodules: the strategic and the tactical component. The strategic component takes the conceptual representation of the message as
input, together with the user level and style, and decides what to display. The
tactical component takes this as input together with the grammar and the
lexicon and for the target language and produces the final message.

11.3.3 An example of message generation

An application of this approach has been used for a prototype hybrid
terminological knowledge base. It was used to build a knowledge base for the
Style Manager module of Hewlett Packard’s VUE (Visual User Environment)
product. VUE 3.0 is a windows management system and user interface that runs
on top of X11R5 Windows under HP-UX Version 9.0. Style Manager is the
application that enables users to change the appearance and behavior of the
VUE screen colors, sound, keyboard, mouse, windows and sessions.

The knowledge base for VUE Style Manager

We rewrote the Style Manager help text using the rules of a controlled English
language. We then marked up the help text using a pre-specified set of mark-up
tags. This taget includes mark-ups for headings (chapters, sections, subsections, topics), tasks, glossaries terms, domain concepts, application-specific concepts, generic and partitive relations, concepts and attributes.

A sample of Style Manager help is shown below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Select a palette in the Color dialog.</td>
</tr>
<tr>
<td>2.</td>
<td>Click a color button, then click Modify to open the Modify dialog.</td>
</tr>
<tr>
<td>3.</td>
<td>Adjust the settings.</td>
</tr>
<tr>
<td>5.</td>
<td>Repeat steps 2-4 to modify another color button.</td>
</tr>
<tr>
<td>6.</td>
<td>Choose OK in the Color dialog.</td>
</tr>
</tbody>
</table>

Figure 11.3: A sample of VUE Style Manager help

Figure 11.4 shows the corresponding part of the software function model. The
controlled and marked-up textual descriptions of software tasks are organized
hierarchically.

```xml
<task> n>1.2.</n>Modify a palette.</task>
<task> n>1.2.1.</n>Select a palette in the <appl>Color dialog</appl>.</task>
<task> n>1.2.2.</n>Open the <appl>modify dialog</appl>.</task>
<task> n>1.2.2.1.</n>Click the <appl>color button</appl>.</task>
<task> n>1.2.2.2.</n>Click the <appl>Modify button</appl>.</task>
<task> n>1.2.3.</n>Adjust the settings.</task>
<task> n>1.2.4.</n>Choose the <appl>OK button</appl> in the <appl>Modify dialog</appl>.</task>
<task> n>1.2.5.</n>Repeat steps 2-4 to modify another <appl>color button</appl>.</task>
<task> n>1.2.6.</n>Choose the <appl>OK button</appl> in the <appl>Color dialog</appl>.</task>
```

Figure 11.4: Part of the software function model for VUE Style Manager help

Figure 11.5 shows part of the application-specific model. The concept ‘modify button’ is linked with the task 1.2.2.2 of the software function model above through the ‘ref’ attribute.
Generating error messages in English and Greek

We implemented a prototype message generation system for VUE Style Manager help that generates error messages in English and Greek. The basic steps of message generation in this prototype, corresponding to the steps described in the previous section, are:

(a) User level and style specification

We defined four different levels of expertise and four different styles. The user levels with their descriptions are listed in Table 11.1, and the styles in Table 11.2.

(b) History list update

Assume that the user is in the color dialog in the figure above, and wants to modify a color of the currently selected palette. As shown in the software function model description in Figure 10.4, to modify a color of the palette the user must:

- Select the palette (task 1.2.1)
- Select the color (task 1.2.2.1)
- Click the modify button to open the modify dialog (task 1.2.2.2)
- Make the appropriate adjustments (task 1.2.3)

Assume that the user has already selected the palette and makes the mistake of clicking the modify button without having first selected a color. The history list will then contain 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) Generation of the conceptual message representation

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-specific model. It finds the concept ‘modify-button’ that has the attribute ‘ref’ with value ‘1.2.2.2’. It then searches in
the software function 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 function 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. It generates a conceptual representation of this:

precondition(1.2.2., 1.2.2.1., 1.2.2.2.)

which means that to perform the current task (1.2.2.2) successfully and then task 1.2.2, the user must first perform task 1.2.2.1. This representation is passed to the natural language generation module to produce the final message to the user in the required language.

(d) Generation of the final message

Assume that the user declared his level as 4 (he is inexperienced). The strategic component uses this information to decide the content of the final message. Since the user level is 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 deciding on the content of the message, the strategic component must then decide the message style. This is based on the style requested by the user. If the user selects the style s1, the strategic component decides that the message should be in the imperative mood.

The tactical component takes the previous, current and final tasks that must be included in the resultant message as inputs. It searches for the translations of these tasks in the translation memory of the software function model, according to the user’s native 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 Κομπί-χρώμα</td>
<td>πατώ</td>
<td>το Κομπί-χρώμα</td>
</tr>
<tr>
<td>current task</td>
<td>1.2.2.2 Πατήστε το Κομπί-τροποποίηση</td>
<td>πατώ</td>
<td>το Κομπί-τροποποίηση</td>
</tr>
<tr>
<td>final task</td>
<td>1.2.2 Αναφέρετε το Διάλογο-τροποποίηση</td>
<td>ενοχώ</td>
<td>το Διάλογο-τροποποίηση</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 then generates the final message. Note that we used the same representation formalism for both English and Greek grammars and lexicons. The main difficulties we encountered had to do with the Greek language itself, which is morphologically much more complex than English.

For example, the tactical component will generate the following messages for the four different styles and a user level of 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 a color-button first and then click the modify-button to open the modify-dialog.</td>
<td>Αρχικά πατήστε ένα κομπί-χρώμα και στη συνέχεια πατήστε το κομπί-τροποποίηση για να ανοίξετε το διάλογο-τροποποίηση.</td>
</tr>
<tr>
<td>s2</td>
<td>You have to click a color-button first and then click the modify-button to open the modify-dialog.</td>
<td>Αρχικά πρέπει να πατήσετε ένα κομπί-χρώμα και στη συνέχεια να πατήσετε το κομπί-τροποποίηση για να ανοίξετε το διάλογο-τροποποίηση.</td>
</tr>
<tr>
<td>s3</td>
<td>A color-button must be clicked first. Then, click the modify-button to open the modify-dialog.</td>
<td>Αρχικά πρέπει να πατηθεί ένα κομπί-χρώμα. Στη συνέχεια πατήστε το κομπί-τροποποίηση για να ανοίξετε το διάλογο-τροποποίηση.</td>
</tr>
<tr>
<td>s4</td>
<td>You didn't click a color-button. Click a color-button first and then click the modify-button to open the modify-dialog.</td>
<td>Δεν πατήσατε ένα κομπί-χρώμα. Αρχικά πατήστε ένα κομπί-χρώμα και στη συνέχεια πατήστε το κομπί-τροποποίηση για να ανοίξετε το διάλογο-τροποποίηση.</td>
</tr>
</tbody>
</table>

11.3.4 Conclusions

The combination of knowledge bases and natural language generation techniques for the generation of messages in software applications offers the following advantages:

- **Multilingual support** More languages can be added to the language-independent knowledge base.
- **Adaptation to the user’s needs** Rules can be added that adapt the message information according to such as user’s task and level of expertise.
- **Control of the style and content of messages** We can enforce the use of a controlled language and specific stylistic rules in the generated messages.

The disadvantages of this approach are:

- **The cost of knowledge extraction** This is mainly the work required to set up the knowledge base. The exploitation of terminological knowledge along with the hybrid architecture reduces this cost.
- The computation cost: This is a significant factor, as the application has to generate messages on-line. The simple knowledge base we describe above reduces the cost of terminological information retrieval, and the simple natural language generation techniques reduce the cost of generating the message in the target language.
Chapter 7 Building an Internationalized Word Processor—A Case Study—Lingzi Jin

Chapter 8 Languages and Character Sets—Stavros Kokkotos, Costas Spyropoulos, Timo Honkela, Tuula Käpylä, Krista Lagus, Pat Hall

Chapter 9 Culture, Conventions and Local Practices—Krista Lagus, Risto Sultitala, Timo Honkela

Chapter 10 Documentation and Translation—George Vouros, Vangelis Karkaletsis, Costas Spyropoulos

Chapter 11 Message Generation—Vangelis Karkaletsis, Costas Spyropoulos, George Vouros, Timo Honkela, Krista Lagus, Aarno Lehtola

Chapter 12 Quality Assurance—Niamh McHugh, Timo Honkela, Ray Hudson

Appendix A Current Platform Support for Internationalization—Aarno Lehtola, Sakari Kalliomäki, Markku Kylänpää, Francis Magann, Lingzi Jin

Appendix B Commercial Tools to Support Localization—Vangelis Karkaletsis, Stavros Kokkotos, Costas Spyropoulos, George Vouros, Ray Hudson

Appendix C The Characteristics of Some European Languages—Tuula Käpylä, Timo Honkela

We have also produced an ‘Awareness package’ about GlossaSoft and software globalization. This was produced by Tuula Käpylä and Timo Honkela, using Toolbook.

Pat Hall, Editor,
7 July 1996
SOFTWARE WITHOUT FRONTIERS

A multi-platform, multi-cultural, multi-nation approach

Edited by

P.A.V. Hall
The Open University, UK

and

R. Hudson
Cray Systems, Luxembourg

JOHN WILEY & SONS
Chichester • New York • Weinheim • Brisbane • Singapore • Toronto