Linguistic Processing Modules in ALEP for Natural Language Interfaces

Montserrat Marimon, Axel Theofilidis
IAI - Saarbrücken
{montse, axel}@iai.uni-sb.de

Thierry Declerck, Andrew Bredenkamp
DFKI - Saarbrücken
{declerck, andrewb}@dfki.de

In this paper we present the linguistic resources - the text handling and the linguistic processing modules - which have been developed for the MELISSA project using the ALEP platform. In particular, we will see how generic (large scale) grammars involving deep linguistic analysis can be efficiently used for NL interfaces, and how a modularized design of the linguistic resources allows us to deal with the peculiarities of sub-languages, while at the same time keeping the resources as general as possible.

1 Introduction

The MELISSA project (cf. [7] [1] provides technology, tools, and linguistic resources for Spanish, English and German that will allow software developers to integrate NL interfaces (supporting spoken and written input) into their products. In that, MELISSA focusses on NL understanding.

The architecture of a MELISSA runtime system is set up to construct, primarily, meaning representations of NL utterances which users make in interacting with a given software application. In the context of one of the applications serving as the validation base of MELISSA and assuming Spanish users[2], typical instructions will be:

Propuesta 98/65 (Proposal 98/65)

Elaborar una propuesta de tipo compras generales (Process a proposal of type 'compras generales')

Elaborar una propuesta que tenga como referencia VARIOS y cuyo importe sea de 25.000 ptas (Process a proposal which has reference 'VARIOS' and whose budget is 25.000 ptas)

The meaning representations obtained for such user instructions are interpreted against the background of an application knowledge model. Successful semantic interpretation, in turn, triggers the appropriate executable function call. The particular modules integrated in this process of natural language understanding are the THM (Text Handling Module), LPM (Linguistic Processing Module), SAM (Semantic Analysis Module), FGM (Function Generatior Module), AKR (Application Knowledge Repository).

In this paper we concentrate on the linguistic processing strategies and resources that are used in MELISSA. The approach chosen involves deep linguistic analysis of input expressions using unification-based grammars. We will outline and discuss design features of the underlying linguistic resources that relate to the major challenge of deploying unification-based, deep linguistic analysis in the context of the MELISSA project: to meet with the pecularities of the sub-language of computer instructions by introducing robustness features, while at the same time providing for accurate and efficient performance.

[1] The MELISSA project is funded by the Commission of the EU (DG III - 6) under ESPRIT 22252. Project partners are Software AG Espana (SAGE), Spain (coordinator); Anite Systems (Anite), Luxembourg (partner); Institut für Angewandte Informationsforschung (IAI), Saarbrücken, Germany (partner); SEMA Group (SEMA), France/Spain (partner); Organizacion Nacional de Ciegos de Espana (ONCE), Spain (user); NET-Cologne - Gesellschaft für Telekommunikation mbH, Köln, Germany (user). `http://www.anite-systems.lu/melissa`.

[2] ICAD, an administrative purchase and acquirement handling system, eployed at ONCE, dealing with budget proposals and providing information to help decision makers.

2 Linguistic Platform

The core of the MELISSA LPM is based on the Advanced Language Engineering Platform (ALEP; cf. [6]), re-using e.g. its morphographemic analyser, parser and feature interpreter. In the second place, ALEP is being used as the development platform by the language engineers in charge of developing MELISSA linguistic resources.

A number of criteria motivates the choice of ALEP. ALEP provides a "lean" (based on term unification) typed feature structure formalism that has been used – in continuation of the grammar development effort of the LS-GRAM project (cf. [8]) – to design lexicons and grammars within the general paradigm of HPSG (cf. [4] [5]). Furthermore, ALEP provides an efficient head scheme based parser, rule indexation mechanisms and, above all, a number of devices that support a modular and, thus distributed design of linguistic resources. An interface format enables integration of SGML encoded data structures obtained from text processing modules (THM) with grammars supporting deep linguistic analysis. The ALEP refinement facility allows for enriching linguistic structures, as obtained from the parsing operation, with additional information; it is thus possible to shift the computational burden of accounting for a variety of grammatical constraints, grammar relaxations, and ambiguities (lexical, syntactic and semantic) from the computationally expensive parsing operation to the stage of refinement (which will be described below).

All these ALEP features have contributed to an implementation of the NL understanding capabilities of MELISSA with efficient run-time performance without sacrificing declarativity or generality of linguistic resources.

3 The Text Handling Module

The Text Handling Module (THM) is conceived to prepare the textual[3] input for higher level processing. It is designed as a set of modular components dealing with both language specific and application specific phenomena[4]. The latter are derived from the application specific corpus analysis. The THM is parameterizable for all the sub-modules defined.

[3]Note that speech input is first converted to written text.

[4]THM has been specifically designed for the ICAD application (cf. footnote 2) and KEWIS, a citizen service and information system with the requirement of multilinguality.

3.1 Functionality

The first task of THM is to delimit the basic textual units, and to organize the input into a (rather flat) tree structure. Tagging of textual units is done by means of SGML tags, which is integrated by means of a rule-based interface to the high-level linguistic analysis in ALEP.

The "top-level" unit of representation is the paragraph, marked with a <P> ... </P> tag pair. Below this, the input is organized into sentences (<S> ... </S>). Potential errors in sentence splitting introduced by abbreviations are as far as possible avoided since known abbreviations are checked in language dependent abbreviations databases. Language specific modules are also activated for dealing with punctuation marks.

Each sentence is, in turn, divided into word tags (<W> ... </W>). Within those tags, the words and the recognized language or application specific phenomena are classified by means of the "TYPE" feature. Special constructs such as fixed expressions, (dates, proper names, numbers, multiple word unit, etc.) and application specific constructs (codes or other items from an application specific ontology) are recognised, marked up with an appropriate TYPE value and parsed to produce a normalized value (e.g. an ISO representation of dates, etc.). For instance, consider how the following is marked up:

Propuesta tipo compras generales del 28 de abril de 1999

```
<P>
 <S
  <W TYPE="WORD" ORIG="propuesta"
     CONV="propuesta"...>propuesta</W>
  <W TYPE="PROPTYPE" ORIG="tipo compras generales"
     CONV="0801"...>tipo compras generale</W>
  <W TYPE="WORD" ORIG="de"
     CONV="de"...>de</W>
  <W TYPE="DATE" ORIG="el 28 de abril de 1999"
     CONV="28/04/1999"...>el 28 de abril de 1999</W>
 </S>
</P>
```

Note that string *tipo compras generales* has been recognised as a proposal type based on application-specific patterns. As it is a 'W' element, it will be treated as atomic for the rest of the processing chain, i.e. its internal structure is not inspected, as for the (language-specific) date *el 28 de abril de 1999*, the CONV value (created according to the application definitions) will be passed on for use in the SAM module.

The THM is also being used for "noise filtering". Sub-strings which have been identified as irrelevant

in the application specific corpus analysis are suppressed. This holds, for instance, for politeness markers such as as *por favor (please)*.

3.2 Implementation

The implementation of the THM is in the Perl programming language, ensuring a high level of portability across the operating systems on which the MELISSA system runs (Unix and NT). The THM comes in two versions, a stand-alone version and a server-client implementation. THM can be also used outside the context of MELISSA, being configurable with respect to its output, the default being an SGML type representation.

The phenomena under consideration are identified either by means of regular expressions or combinations thereof, or by means of fast lookup in tables. These resources are largely determined as a result of investigation of user needs (i.e. application specific corpora). In a number of specific areas, static resources in the form of data files are used. These include, for instance, abbreviations and codes, which are developer- and user-definable by means of an configuration interface. The format of these external resources is application dependent and include native Perl data structures, "foreign formats" as well as plain text files.

In the THM of MELISSA we have thus achieved a high degree of modularity, particularly with respect to the languages and applications under consideration. The experience acquired suggests that adaptation to other languages and applications would be relatively straightforward. The modularity achieved so far also provided support for the related proceesing steps in the architecture of MELISSA.

The THM component provides a high-level of robustness, efficiency and openness for the high-level processing, in that the LPM grammars described below, are reasonably static with respect to changing requirements of the application, since application-specific terms are represented with abstract lexical entries for subsequent processing.

4 The Linguistic Processing Module

Acceptance of an NLI (by the end-user) requires that the underlying NL processing capabilities cope with as broad a range of natural ways of accessing application functionality by language as possible. It further requires that response times do not exceed those of standard (command line or graphical) user interfaces.

¿From the point of view of grammar design, "natural ways of accessing application functionality" include sub-sentential input units (*Propuesta 87/78*), head-elliptical input units (*ahora las que ...*), functionally incomplete input units or constituents (e.g. lacking determiners (*tramitar propuesta 98/89*)), and input units exhibiting non-standard constituent ordering. Though the scope of the MELISSA project puts limits on the provision of robustness features, the coverage of the current linguistic resources goes far beyond "core" grammatical phenomena, to the above-mentioned types of construct. In that, generality of the core components of the linguistic resources is maintained on the basis of a strictly modular design. These components may thus be re-used across different applications, by supplementing them with add-on modules according to the demands of a given application and its specific user base.

4.1 Grammar Design

The process of designing and implementing a grammar in ALEP is divided into two steps: the type system declaration and the definition of lexical entries and grammar rules.

4.1.1 The Lexicon

The ALEP formalism supports a lexicalist treatment of linguistic phenomena such as subcategorization, control relations, subject-verb agreement, traditionally dealt with by means of specialised structure rules. This results in an increase in redundancy and complexity of lexical entries. For this reason, it is very important the way this information is encoded.

The ALEP formalism does not support multiple inheritance. However, a rich set of macros can be defined so that lexical information common to a subset of entries can be moved out of lexical entries into separate macro definitions. This ensures coherence of the linguistic code and ease of maintenance and updating of the lexical resources. The Spanish lexical resources include 90 macros encoding lexical templates for open class categories.

These macros may be defined in such a specific and exhaustive way that they account for a whole class of lexical entries or in a modular way so that macros can be nested. We followed the latter strategy, which allowed the grammar developers to adapt the generic linguistic resources to the application requirements in a straightforward way. Consider for instance relaxing functional completeness or subject-verb agreement, in this case, we only need to modify the modular macros encoding this information, lexical macros

that use these macros are automatically updated on compilation.

4.1.2 The Structure Rules

Grammar rules are reduced to a small set of binary-branching phrase structure rules.

Structural macros have been defined to implement the general principles governing the ID schemata, diversification of rules for the different categories allowed grammar developers to modularize the lingware, and by means of a system of rule sets, for an easy activation/deactivation of those rules which are appropriate for a given application. On the other hand, occasionally diversification of rules favoured the extension of a generic treatment of a given phenomenon to cope with its specific requirements. So for example, the rule dealing with adverbial modifiers was left aside in the ICAD application, whereas those dealing with clitics have been extended to cover "laismo"/"leismo".

In addition, special macros have been defined to cover those prototypical constructions of such a data handling application. An example of these is the macro defined to deal with other inputs than finite clauses, such as infinitival VPs (e.g., *Tramitar propuestas (Send proposals)* or NPs (e.g., *Propuesta 98/45 (Proposal 98/45)*. Again, multiplication of these special rules favoured their extension to other types of input expressions which, even though they may not be considered as prototypical structures, provide the LPM with robustness, something to bear in mind when developing NL interfaces.

4.2 Efficiency

ALEP grammars were being used for the first time in an industrial context. Therefore, a major requirement in the LPM was efficiency. To fulfil this requirement, grammar developers have deployed expensive ALEP facilities to control the process of analysis and to reduce spurious ambiguity, namely the *specifier features, head selection declarations*, and the *refinement*.

- **Specifier features.** The type system on the basis of which linguistic knowledge is defined can also be used to encode non-linguistic information, that may be used to activate only those lexical entries and structure rules appropriate for the different analysis steps, thus avoiding spurious ambiguity at analysis steps in which they cannot be resolved.
- **Head selection declaration.** ALEP provides a head schema parser; i.e., structure rules are accessed by a separate head-daughter key, which grammar developers can specify in a special type of declaration. Specifying functional categories (e.g., determiners, conjunctions) as being parsing heads rather than substantial categories, significantly improves the performance of the grammar, and allows the grammar to be extended with no significant effect on performance.
- **Refinement.** Spurious ambiguity has been successfully reduced by the use of the ALEP facility to distinguish between two different phases of the analysis process: 'parsing', which builds up the tree, and 'refinement', which adds information to the tree. So, for instance, a unique lexical entry for modifying prepositions has been encoded in the 'parsing' lexicon, leaving the semantic content unspecified or underspecified, different readings are encoded in different 'refinement' entries. This facility has also been used in phrase structure rule definitions, where for instance the distinction between different types of unbounded dependencies is postponed to the refinement phase. Consequently, relaxation on accents for interrogative pronouns, determiners and adverbs did not interfere in the processing performance.

To illustrate performance, in a testsuite containing 808 sentences with an average of 7.341 words/sentence, the analysis time (including both THM and LPM) is 0.83 seconds/word[5].

5 Meaning Representation

During the refinement stage of linguistic processing, compositional meaning representations are obtained by recursive embedding of semantic feature structures corresponding to the constituents of the processed utterance. In that, predications, argument relations, and modifier relations are encoded as individual semantic facts, marked by a unique wrapper data type, so called 'sf-terms' (SFs). Links between semantic facts are established through sharing of variables which mark eventive or non-eventive discourse referents. This is illustrated by the following example:

Elaborar nueva propuesta (Elaborate new proposal)

```
t_sem:{
      index => sf(index(event,E)),
      pred => sf(pred(elaborate,E,A,B)),
```

[5]Machine: Sun Ultrasparc

```
arg2 => t_sem:{
     arg => sf(rel(arg2,E,B)),
     index => sf(index(nevent,B)),
     pred => sf(pred(proposal,B)),
     mods => [ t_sem:{
          mod => sf(rel(quality,B,M)),
          index => sf(index(nevent,M)),
          pred => sf(pred(new,M,B))},
          arg1 => sf(rel(arg1,M,B)) }]
```

Borrowing from [1], a rather small set of argument relation labels is assumed, with external arguments and least oblique arguments being assigned *arg1* and *arg2* respectively. A number of semantically interpreted argument relation labels, such as *arg_place* or *arg_goal*, is reserved for arguments the meaning of which resembles that of adjuncts. As for adjuncts, a standard set of modifier relations is used including e.g. *place*, *goal*, *origin*, *time*, *beneficiary* or *instrument*.

Beyond encoding predicate-argument structure and modification, the MELISSA semantic model also accounts for functional semantic information relating to negation, determination, tense and aspect. The SF-encoding scheme carries over to these facets of semantic information, such that a negated predication as in *not approved*, for instance, would yield the following representation:

```
t_sem:{
     index => sf(index(event,E)),
     pol => sf(polarity('0',E)),
     pred => sf(pred(approve,E,A,B)) }
```

Based on the SF-encoding scheme, a flat list of all SFs representing the core linguistic meaning of an input expression is extracted from the output structures of the LPM and passed on to the Semantic Analysis Module (SAM) which is responsible for mapping the semantic graph constituted by this list of SFs onto the correponding representation of an application function.

6 Conclusions and Prospects

In this paper, we have seen how generic (large scale) grammars involving deep linguistic analysis can be made efficient for use in NL interfaces, and how a modularized design of the linguistic resources allows us to easily adapt them to application specific requirements.

The basic set up of the linguistic modules (THM, LPM), as well as the openness of the ALEP platform, support the integration of more powerful low-level processing tools with the deep linguistic analysis. In particular, we envisage externalising morphological resources (lexica), and integrating part of speech taggers and shallow parsers. By these means, it will be possible to further reduce the non-determinism of the high-level processing and to make the parsing predictions more accurate. The feasibility of the integration of such tools has already been proven on a small scale basis (cf. [2] [3]). Investigations as to how to adapt the LPM to optimize interaction with such low-level tools are currently under way. Areas of interest are, for example, a treatment using default lexical templates to obtain useful meaning representations without the need for explicit lexical entries.

References

[1] Badia, T. and Colominas, C. *Predicate-Argument Structure*, in Linguistic Specifications for Typed Feature Formalism, Studies in MT and NLP, volume 10

[2] Declerck, T. and Mass, D. *The integration of Part-of-Speech Tagger into the ALEP Platform.* In: Proceedings of the 3rd ALEP User Group Workshop. Saarbrücken 1997.

[3] Fouvry, F. and Bredenkamp, A. *Partial parsing in ALEP.* In: Proceedings of the 3rd ALEP User Group Workshop. Saarbrücken 1997.

[4] Pollard, C. and Sag, I. (1987) *Information-Based Syntax and Semantics*, CSLI Lecture Notes Series, Chicago, Chicago Univerisity Press.

[5] Pollard, C. and Sag, I. (1992) *Head Driven Phrase Structure Grammar*, Chicago University Press.

[6] Simpkins, N.K. (1994) *Linguistic Development and Processing, ALEP-2 User Guide*, CEC, Luxembourg

[7] Bredenkamp, A., Declerck, T., Groenendijk, M., Phelan, P., Rieder, S., Schmidt, P., Schulz, H., and Theofilidis, A., *Natural Language Access to Software Applications.* In: Proceedings of COLING-ACL, Montreal 1998.

[8] Schmidt, P., Theofilidis, A., Rieder, S. Declerck, T *Lean Formalism, Linguistic Theory, and Applications. Grammar Development in ALEP.* In: Proceedings of the 16th COLING, Copenhagen 1996.