Natural Language Access to Software Applications

Paul Schmidt
University of Mainz, An der Hochschule 2,
D-76711 Germersheim
schmidtp@usun2.fask.uni-mainz.de

Sibylle Rieder , Axel Theofilidis
IAI, Martin-Luther-Str. 14
D-66111 Saarbrücken
{sibylle;axel}@iai.uni-sb.de

Marius Groenendijk, Peter Phelan, Henrik Schulz
Anite Systems, 13, rue Robert Stumper
L-2557 Luxembourg
{marius;peter;henrik}@anite-systems.lu

Thierry Declerck
Deutsches Forschungszentrum für KI
D-66123 Saarbrücken
declerck@dfki.uni-sb.de

Andrew Bredenkamp
University of Essex, Wivenhoe Park, Colchester, CO4 3SQ
andrewb@essex.ac.uk

Abstract

This paper reports on the ESPRIT project MELISSA (Methods and Tools for Natural-Language Interfacing with Standard Software Applications)[1]. MELISSA aims at developing the technology and tools enabling end users to interface with computer applications, using natural-language (NL), and to obtain a pre-competitive product validated in selected end-user applications. This paper gives an overview of the approach to solving (NL) interfacing problem and outlines some of the methods and software components developed in the project.

Introduction

The major goal of MELISSA is ***to provide the technology and tools enabling software developers to provide a Natural Language (NL) interface for new products, as well as for legacy applications***. The project is based on the conviction that NL is the most user friendly interface for specific software applications and for a specific kind of users. NL is 'generic' requiring little or no training. Integrated with speech recognition and speech generation the NL interface is optimally convenient and allows for easy access to software systems by all kinds of (non-expert) users as well as for users with specific disabilities (e.g. visual, motor).

MELISSA will deliver three main components: a core of linguistic processing machinery and generic linguistic resources for Spanish, English and German; a set of methods and tools for acquiring and representing the knowledge about the host application and specific linguistic resources required for this application; a set of methods and tools for integrating the MELISSA core, the application knowledge, and the host application using the CORBA interoperability standard. The overall architecture of a MELISSA-based NL interface consists of the following software modules:

- Speech Recognition Module (**SRM**), which is a commercial product, providing a continuous speech interface for the other NL modules
- Linguistic Processing Module (**LPM**) consisting of the linguistic processing machinery and the linguistic resources
- Semantic Analysis Module (**SAM**) interpreting LPM output using application knowledge
- Function Generator Module (**FGM**) converting SAM output into executable function calls
- Application Knowledge Repository (**AKR**) containing all the relevant application specific knowledge being used by SAM and FGM
- Front-End Module (**FEM**) responsible for invoking requested operations in the application
- Controller Module (**CTR**) co-ordinating the cooperation between the previous modules
- End-User Interface (**EUI**) in which the user types or dictates his NL queries to target application

The focus of MELISSA is on **understanding** NL. In that, MELISSA addresses problems from knowledge representation and linguistic processing. In the following we concentrate on the design and the interrelation of the linguistic and knowledge-based modules (SRM, LPM, SAM, AKR).

The MELISSA tools are designed to be generic such that they support development of NL interfaces for a broad range of software applications. This requires an application independent encoding of linguistic resources, and an elaborate modularization scheme supporting flexible configuration of these resources for different software applications.

[1] This project is sponsored by the Commission of the EU under ESPRIT-22252. Project partners are Software AG, Espana, SEMA, France/Spain, Anite-Systems, Luxembourg, IAI, Germany, ONCE, Spain and the City of Cologne.

Furthermore, successful NL interface must meet with user acceptance requirements regarding response time. This poses a major challenge on the deployment of sophisticated, competence-grammar based NLP technologies as envisaged in MELISSA. One aspect of ensuring efficient performance of a NL interface consists in limiting its capabilities in terms of linguistic coverage. To avoid false (positive or negative) expectations such restrictions must be obvious to the user. In addition, any restriction in terms of linguistic resources must warrant naturalness of expression.

1 The Speech Recognition Module

Speech is the most natural form of communication for people and is felt to greatly extend the range of potential applications suitable for an NL interface.

MELISSA currently adopts a 'black-box' approach to speech recognition, viz., speech is just an alternative to a keyboard. The results of speech recognition are stored and can be retrieved by sending a request to the component. The speech component itself can be controlled by voice commands. Before using the SRM, speakers have to 'train' it in order to adjust the general voice model to the specific speaker's voice characteristics.

The speech interface sends recognized utterances as strings to other MELISSA components, but is not able to interact on a higher level with those components. In a subsequent phase the feedback and co-operation between the MELISSA core components and the SRM will be addressed.

2 The Linguistic Processing Module

The core of the LPM is based on the Advanced Language Engineering Platform (ALEP), the EU Commission's standard NLP development platform [Simpkins 94]. ALEP provides the functionality for efficient NLP: a 'lean' linguistic formalism (with term unification) providing typed feature structures (TFSs), an efficient head scheme based parser, rule indexation mechanisms, a number of devices supporting modularization and configuration of linguistic resources, e.g. an interface format supporting information flow from SGML-encoded data structures to TFSs (thus enabling straightforward integration of 'low-level' processing with deep linguistic analysis), the refinement facility allowing for separating parsing and 'semantic decoration', and the specifier mechanism allowing for multi-dimensional partitioning of linguistic resources into specialized sub-modules.

For the first time ALEP is used in an industrial context. In the first place, core components of ALEP (parser, feature interpreter, linguistic formalism) are used as the basis of the MELISSA LPM. In the second place, ALEP is used as the development platform for the MELISSA lingware.

The coverage of the linguistic resources for the first MELISSA prototype was determined by a thorough user needs analysis. The application dealt with was an administrative purchase and acquisition handling system at the Spanish organization of blind people, ONCE.

The following is an outline of solutions realized in the LPM for text handling, linguistic analysis and semantic representation.

2.1 Text Handling

The TH modules for MELISSA (treating phenomena like dates, measures, codes (pro-nr. 123/98-a1-T4), abbreviations, but also multiple word units and fixed phrases come as independent Perl preprocessors for pattern recognition, resulting in a drastic improvement of efficiency and a dramatic expansion of coverage.

Within the general mark up strategy for words a module has been added which allows the treatment of specific sequences of words building units. Once those patterns have been recognized and concatenated into one single unit, it is easy to convert them to some code required by the application. Precisely this latter information is then delivered to the grammar for further processing. For one application in MELISSA it is, for example, required to recognize distinct types of proposals and to convert them into numeric codes (e.g. 'òrdenes de viaje' into the number '2019'.)

The TH components allow for an expansion of the coverage of the NLP components. Experiments have already been made in integrating simple POS-tagging components and in passing this information to the ALEP system [Declerck & Maas 97]. Unknown words predictable for their syntactic behaviour can be identified, marked and represented by a single default lexical entry in the ALEP lexicon. In one practical experiment, this meant the deletion of thousands of lexical entries.

The default mechanism in ALEP works as follows, during parsing ALEP applies the result of lexical look-up to each of the terminal nodes; if this fails then ALEP will look at lexical entries which contain a default specifier to see whether any of them matches (typically these are underspecifed for string value, but fully specified for syntactic category etc.). Clearly without valency information such an approach is limited (but nevertheless useful). Future work will focus on the (semi)-

automatic identification of this information in the pre-processing.

The modular design of the TH components (distinction of application specific TH phenomena and general ones) allows for a controlled extension to other languages and other applications.

2.2 Linguistic Analysis

Based on experiences from previous projects [Schmidt et al. 96], mainstream linguistic concepts such as HPSG are adopted and combined with strategies from the 'lean formalism paradigm'.

For MELISSA, a major issue is to design linguistic resources which are transparent, flexible and easily adaptable to specific applications. In order to minimize configuration and extension costs, lingware for different languages is designed according to the same strategies, guaranteeing maximal uniformity. This is realized in semantics. All language modules use the same type and feature system.

Macros provide an important means of supporting modularity and transparency. They are extensively used for encoding lexical entries as well as structural rules. Structural macros mostly encode HPSG-like ID schemes spelled out in category-specific grammar rules. Structural macros are largely language-independent, but also lexical macros will be 'standardized' in order to support transparency and easy maintenance.

The second major issue in linguistic analysis is efficiency of linguistic processing. Efficiency is achieved e.g. by exploiting the lingware partitioning mechanisms of ALEP. Specifier feature structures encode which subpart of the lingware a rule belongs to. Thus for each processing step, only the appropriate subset of rules is activated.

Efficient processing of NL input is also supported by separation of the 'analysis' stage and one or several 'refinement' stages. During the analysis stage, a structural representation of the NL input is built by a cf. grammar, while the refinement stage(s) enriches the representation with additional information. Currently, this is implemented as a two-step approach, where the analysis stage computes purely syntactic information, and the refinement adds semantic information (keeping syntactic and semantic ambiguities separate). In the future we will use further refinement steps for adding application-specific linguistic information.

2.3 Semantic Representation

During linguistic analysis, compositional semantic representations are simultaneously encoded by recursive embedding of semantic feature structures as well as by a number of features encoding distinct types of semantic facts (e.g. predications, argument relations) in terms of a unique wrapper data type, so called 'sf-terms' (SFs). Links between semantic facts are established through variable sharings as (2) shows:

```
(1) Elaborate new proposal
(2)  t_sem:{
     indx => sf(indx(event,E)),
     pred => sf(pred(elaborate,E,A,B)),
     arg2 => t_sem:{
       arg => sf(arg(theme,E,B)),
       pred => sf(pred(proposal,B)),
       mods => [ t_sem:{
         mod => sf(mod(quality,B,M)),
         pred => sf(pred(new,M))} ] }}
```

The flat list of all SFs representing the meaning of an NL input expression is the input data structure for the SAM.

Besides predicate argument structure and modification, the semantic model includes functional semantic information (negation, determination, quantification, tense and aspect) and lexical semantics. The SF-encoding scheme carries over to these facets of semantic information as well.

Special data types which are recognized and marked up during TH and which typically correspond to basic data types in the application functionality model, are diacritically encoded by the special wrapper-type 'type', as illustrated in (4) for an instance of a code expression:

```
(3) proposal of type 2019
(4) t_sem:{
     pred => sf(pred(proposal,P)),
     mods => [ t_sem:{
       mod => sf(mod(concern,P,M)),
       pred => sf(type(proptype(2019),M))} ]}
```

3 Modelling of Application Knowledge

Two distinct but related models of the host application are required within MELISSA. On the one hand, MELISSA has to understand which (if any) function the user is trying to execute. On the other hand, MELISSA needs to know whether such a functional request *can* be executed at that instant. The basic ontological assumption underpinning each model is that any application comprises a number of functions, each of which requires zero or more parameters.

3.1 The SAM Model

The output of the LPM is basically application independent. The SAM has to interpret the semantic output of the LPM in terms of a specific application. Fragments of NL are inherently ambiguous. Thus, in general, this LPM output will consist of a number of possible interpretations. The goal of the SAM is to identify a unique function call for the specific application. This is achieved by a (domain-independent) matching process, which attempts to unify each of the LPM results with one or more so-called mapping rules. Heuristic criteria, embodied within the SAM algorithm, enable the best interpretation to be identified. An example criterion is the principle of 'Maximal Consumption', by which rules matching a greater proportion of the SFs in an LPM result are preferred.

Analysis of the multiple, application-independent semantic interpretations depends on the matching procedure performed by the SAM, and on the mapping rules. (5) is a mapping rule:

```
(5) rule(elaborate(3),                      -- (a)
   [elaborate, elaboration, make, create,
      creation, introduce],                 -- (b)
   [arg(agent, elaborate, _ ),
    arg(theme, elaborate, proposal),
    mod(concern, proposal,
      type(proptype(PropType)))],           -- (c)
   [new_proposal_type(
      proptype(PropType))]).                -- (d)
```

Each mapping rule consists of an identifier (a), a list of normalised function-word synonyms (b), a list of SFs (c), and finally, a simple term representing the application function to be called, together with its parameters (d).

The SAM receives a list of SF lists from the LPM. Each list is considered in turn, and the best interpretation sought for each. All of the individual 'best results' are assessed, and the overall best result returned. This overall best is passed on to the FGM, which can either execute, or start a dialogue.

The SFs embody structural semantic information, but also very important constraint information, derived from the text-handling. Thus in the example rule above, it can clearly be seen that the value of 'PropType' must already have been identified (i.e. during text handling) as being of the type 'proptype'. In particular cases this allows for disambiguation.

3.2 The Application State Model

It is obvious that NL interfaces have to respond in a manner as intelligent as possible. Clearly, certain functions can only be called if the application is in a certain state (e.g. it is a precondition of the function call 'print_file' that the relevant file exists and is printable). These 'application states' provide a means for assessing whether or not a function call is currently permitted.

A standard application can reasonably be described as a deterministic finite state automaton. A state can only be changed by the execution of one of the functions of the application. This allows for modelling an application in a monotonic fashion and thus calls for a representation in terms of the predicate calculus. From amongst a number of alternatives, the New Event Calculus (NEC) was chosen [Sadri & Kowalski 95] as an appropriately powerful formalism for supporting this state modelling. NEC allows for the representation of events, preconditions, postconditions and time intervals between events. NEC is appropriate for modelling concurrent, event-driven transitions between states. However, for single-user applications, without concurrent functionality, a much simpler formalism, such as, for example, STRIPS-like operators, will be perfectly adequate.

In terms of implementation methodology, the work to be done is to specify the application specific predicates. The state model of the application contains as components a set of *functions* which comprise the application, a set of *preconditions* that must be fulfilled in order to allow the execution of each function, and a set of *consequences* that results from the execution of a function.

Both preconditions and consequences are composed of a subset of the set of *propositions* which comprise the current *application* state. There exists a set of *relations* between the components: A function must *fulfil* preconditions and *produces* a set of consequences. The set of preconditions *is-composed-of* facts. The same holds for the set of consequences and the application state. (6) gives a summary for a simple text editor. ('F' = some file).

```
(6) Preconditions:
   create(F),[not(exists(F))]).
   open(F),[exists(F),not(open(F))]).
   close(F),[exists(F),open(F)]).
   delete(F),[exists(F)]).
   edit(F),[exists(F),open(F)]).
   save(F),[exists(F),open(F),modified(F)]).
   spell_check(F),[exists(F),open(F)]).

   a) Postconditions: Facts to be added
   add(create(F),[exists(F)]).
   add(open(F),[open(F)]).
   add(close(F),[]).
   add(delete(F),[]).
   add(edit(F),[modified(F)]).
   add(save(F),[saved(F)]).
   add(spell_check(F),[modified(F)]).

   b) Postconditions: Facts to be deleted
   del(create(F),[]).
   del(open(F),[]).
   del(close(F),[open(F)]).
   del(delete(F),[exists(F)]).
   del(edit(F),[]).
   del(save(F),[modified(F)]).
   del(spell_check(F),[]).
```

A simple planner can be used to generate remedial suggestion to the user, in cases where the desired function is currently disabled.

4 Adopted Solutions

4.1 Standardisation and Methodologies

Throughout the design phase of the project an object oriented approach has been followed using the Unified Modelling Language [Booch et al. 97] as a suitable notation. It is equally foreseen to actually propose an extension to this standard notation with linguistic and knowledge related aspects. This activity covers part of the 'Methodology and Standards' aspects of the project.

Other activities related to this aspect are concerned with 'knowledge engineering', 'knowledge modelling', and 'language engineering' (e.g. linguistic coverage analysis). Methodologies are being developed that define the steps (and how to carry them out) from a systematic application analysis (a kind of reverse-engineering) to the implementation of a usable (logical and physical) model of the application. This model can be directly exploited by the MELISSA software components.

4.2 Interoperability

As stated in the introduction, CORBA [Ben-Natan 1995] is used as the interoperability standard in order for the different components to co-operate. The component approach, together with CORBA, allows a very flexible (e.g. distributed) deployment of the MELISSA system. CORBA allows software components to invoke methods (functionality) in remote objects (applications) regardless of the machine and architecture the called objects reside on. This is particularly relevant for calling functions in the 'hosting' application. The NL input processing by the MELISSA core components (themselves communicating through CORBA) must eventually lead to the invoking of some function in the targeted application. In many cases this can be achieved through CORBA interoperability techniques (e.g. object wrapping).

This approach will enable developers to provide existing (legacy) applications with an NL interface without having to re-implement or reverse engineer such applications. New applications, developed with components and distributed processing in mind, can integrate MELISSA components with little development effort.

4.3 Design and Implementation

The software design of all components has followed the object-oriented paradigm. The SRM for example is implemented based on a hierarchical collection of classes. These classes cover for instances software structures focused on speech recognition and distributed computing using CORBA. In particular the speech recognition classes were implemented to be independent of various speech recognition programming interfaces, and are expandable. Vocabularies, dictionaries and user specific settings are handled by specific classes to support the main speech application class. Commands can easily be mapped to the desired functionality. Speech recognition results are stored in conjunction with scores, confirmed words and their alternatives. Other MELISSA components can access these results through CORBA calls.

5 Conclusions

MELISSA represents a unique combination of high quality NLP and state-of-the-art software- and knowledge-engineering techniques. It potentially provides a solution to the problem of re-using legacy applications. The project realizes a systematic approach to solving the problems of NL interfacing: define a methodology, provide tools and apply them to build NL interfaces. The production of the first working prototype has proven the soundness of the concept.

MELISSA addresses a highly relevant area wrt. future developments in human-computer interaction, providing users with an intuitive way of accessing the functionalities of computers.

Future work will focus on refinement of methodologies, production of knowledge acquisition tools, improvement and extension of the SAM functionality, robustness and extension of the LPM output. Contonuous user assessment will guide the development.

References

[Ben-Natan 1995] Ben-Natan, Ron (1995) *CORBA : A guide to common object request broker architecture.* McGraw-Hill, ISBN 0-07-005427-4

[Booch et al. 97] Booch, G., Rumbaugh, J., Jacobson, I. (1997) *The Unified Modelling Language User Guide.* Addison Wesley, est. publication December 1997.

[Declerck & Maas 97] Declerck, T. and Maas, H.D. (1997) *The Integration of a Part-of-Speech Tagger into the* ALEP *Platform.* In: Proceedings of the 3rd ALEP User Group Workshop, Saarbrücken 1997.

[Sadri & Kowalski 95] Sadri, F. and Kowalski, R., (1995) *Variants of the Event Calculus.* Technical Note, Imperial College, London.

[Schmidt et al. 96] Schmidt, P., Theofilidis, A., Rieder, S., Declerck T. (1996) *Lean Formalisms, Linguistic Theory, and Applications. Grammar Development in ALEP.* In: Proceedings of the 16th COLING, Copenhagen 1996.

[Simpkins 94] Simpkins, N.K. (1994) *Linguistic Development and Processing.* ALEP-2 *User Guide.* CEC, Luxembourg