A METHODOLOGY FOR APPLICATION DEVELOPMENT FOR SPOKEN LANGUAGE SYSTEMS

Lewis M. Norton, Carl E. Weir**, K. W. Scholz*, Deborah A. Dahl* and Ahmed Bouzid**

*Unisys Corporation

**Lockheed-Martin Corporation

ABSTRACT

A major bottleneck in the development of practical spoken language (SL) applications is the interface between the SL system and back-end application software. In theory, the range of potential back-end software for an SL interface is unlimited, but integration of an SL system with other software requires adapting the SL system to conform to the input formats of the back end. This is typically accomplished by ad hoc, labor-intensive methods, which are highly application-specific. Some SL systems have addressed this problem by attempting to handle only relational database interface applications, but this approach limits the usefulness of such systems, since relational databases represent only a fraction of the applications which could benefit from an SL interface. This paper describes a general, flexible, rule-based approach to integrating SL applications to arbitrary back-end software.

1. SYSTEM OVERVIEW

This approach is implemented as part of the Unisys NL Assistant system.

In addition to the application module, the system also includes:

- A session manager which handles connectivity with external databases, speech recognizers, the telephone system, and also manages visual and auditory displays to the user.
- A dialog manager which accesses the natural language processor and the application module.
- A speech recognizer which provides n-best ASCII text to the natural language processor.
- A natural language engine [2] which interprets the n-best in the context of the current discourse history, and creates a semantic representation (the Integrated Discourse Representation, or IDR) [5]. The format of the IDR does not change from application to application. Thus, it provides a domain and application-independent data structure which is a starting point for the application-specific rules.

We focus in this paper on design of the application module.

Previous Approaches to Application development

Previous spoken language understanding systems have tended to either restrict themselves to particular types of applications, most commonly relational databases [1], or have required developers to create application mapping code from scratch for each type of application [6]. Relational database interfaces represent only a fraction of all the potential applications of spoken language processing technology; consequently it is highly desirable to provide the ability to interface to arbitrary applications. On the other hand, application development can be greatly accelerated with a structured approach to application mapping, since the possibility of code reuse is increased.

2. APPLICATION RULES AND DATA STRUCTURES

The application module is implemented as a set of seven major types of rules, each with a specific function in the application. Input to the application rule module is the IDR, the semantic representation created by the natural language processing component. Output from the application module is a set of one or more actions for the system to take in the next phase of the interaction between user and system. These actions are sent to the session manager as directives to execute, using the KQML protocol for communciation between intelligent agents [3]

Natural language translation and filler translation rules:

The first set of rules which act after the natural language processing is complete are the natural language translation rules and filler translation rules (‘nltrans' and ‘fillertrans', respectively). The function of these rules is to detect patterns in the IDR which correspond to significant information to the

application. The IDR is necessarily a very rich representation, since it must provide sufficient semantic information to support arbitrary applications. Consequently, it will in general contain information irrelevant to any particular application's needs. The function of the natural language translation rules is to detect application-specific information in the IDR. The nltrans and fillertrans rules are very similar in function. They differ in that the natural language translation rules detect patterns in clauses; the filler translation rules detect patterns in noun phrases. One nltrans rule will be fired for each clause in the input sentence; fillertrans rules are fired as needed to process arguments of the main verb of each clause.

Once a pattern has been detected in the IDR, the last step in the natural language translation rule or filler translation rule is to fire a preparation rule.

Preparation rules:

The preparation rules determine actions. The actions are primarily based on information detected by the natural language translation rules, but may also reference any arbitrary information available to the system. The most common type of information referenced in addition to the user's utterance is the state of a database. For example, in a retail banking application, whether the system offers the user an opportunity to make a funds transfer might depend on whether the user has at least two accounts, since a transfer requires at least two accounts. Another common type of information which influences the system's next action is some aspect of the dialog history. For example, abbreviated prompts can be used if the user has already heard a full prompt once [6].

The actions themselves can take several forms. From the user's perspective, an action most frequently manifests itself in the form of the playing of the next system prompt or response. The system's prompt or response may be a question asked by the system if it needs additional information or it may be information provided to the user, such as an account balance. Internal actions not visible to the user may include a decision to access information from an external or internal database or recording information which will be used in later stages of the interaction. Actions are taken by firing additional rules, as described below.

Send rules:

One of the major advantages of the approach to application development described in this paper is that it separates processing of the natural language input from the details of the interface to the back end software. This means that much of the software for applications developed with this approach can be reused even if different customers have dramatically different databases. The send rules are the only set of rules which contain the information about details of the formatting required by specific back-end software. Thus moving an application from one database to another would require primarily modifications to the send rules.

Reply rules:

Reply rules handle formatting of the back end software output for the user. In current applications, only natural language output is supported. In typical applications, this involves translation of tabular information such as that obtained from a database to a spoken format for transmission over an auditory-only modality such as the telephone. Full natural language generation as well as graphical output are potential directions for future versions of the system. Due to the modular nature of the system, addition of different types of multimedia display objects would involve little modification to the system over and above changes to reply rules.

Prompt rules:

Prompt rules are called by the reply rules to compose the actual natural language output requested by the reply rules. They are also called by preparation rules in order to request additional information from the user.

Dialog management rules:

These rules implement meta-level dialog actions such as ending or restarting the interaction, turn-taking and context managment.

Error rules:

Error rules handle various error conditions, for example cases where the system is unable to process an input or an external database becomes unavailable.

The Application Vector Structure:

All of the rules can read from and write to a general blackboard-style data structure, the Application Vector Structure (AVS), through which they can record information which must be globally available for other rules to reference as needed. In addition, the AVS is also used to record planned actions to tak, such as getting something from the database or saying something to the user. One example of how the AVS could be used would be to record an account number elicited from the user in an early part of the dialog for later use with database interactions to obtain account information in later parts of the dialog.

3. EXAMPLE

In this section we briefly illustrate the processing for one utterance in an airline reservations application. In response to a system prompt, "Please tell be what city you will be traveling from and what city you will be traveling to," assume the user has said, "I want to go from Philadelphia to Boston" and the speech recognizer has correctly recognized this utterance.

The natural language engine creates an IDR containing a full semantic representation of this utterance. An nltrans rule detects the 'from city' and 'to city' in the IDR, stores this information in the AVS, and fires a preparation rule to ask for the date when

the user wishes to travel, since that is the next stage of the dialog. A dm rule resets the dialog state to the 'date' state and a prompt is issued to request the date.

No error rules fire in this example. At this point the system waits for the user's reply stating the date (or a user-initiated meta-control utterance, such as a request for definition of a term). The system proceeds in a similar fashion to elicit the date and time of travel and the number of travelers from the user. When the 'from city', 'to city', time, date, and number of travelers have all been determined and are stored in the AVS, a send rule sends a request to the database to determine if a flight is available. A reply rule formats this information for the user, for example "Flight 123 from Philadelphia to Boston is available for four adults on October 15 at 8:00 a.m." The final prompt gives the user an opportunity to plan another trip.

4. METRICS AND EVALUATION

The most appropriate metrics on which to judge the value of an approach to application development are:

- how long it takes to develop an application,
- how long it takes to train a developer to use this methodology
- how accurate are the resulting applications?

Seven very diverse applications have been developed using this approach, four for external customers and three internal demonstrations. We have found that after two weeks of training, a new developer can put together an application module of moderate complexity in about six weeks. The development of the application module rules is the task which consumes the largest proportion of overall development time, followed by dialog design. Adaptation of the natural language processing system and development of speech recognizer grammars are minimal, consuming less than 10% of development time each. Factors which contribute to lengthening the development process in particular applications are the need for mainframe database connectivity (which can be very idiosyncratic) and the complexity of the dialog design.

Accuracy of the resulting systems is measured by an utterance by utterance correct judgment based on the logfile evaluation approach [4]. In this metric a human judge reviews each utterance and each system response and determines whether the system did the correct thing. The most recent formally evaluated system achieved an accuracy of 90% overall, and more than 95% when utterances involving digit sequences were excluded. Nearly all the failures were due to speech recognition errors.

5. NEXT STEPS

The architecture of the application module in the NL Assistant has been established as efficient and practical. The major area for improvement is to increase the level of automation of rule generation, as currently all of the rules except for the prompt rules are hand-coded in Prolog. As the application module development process becomes increasingly standardized, the potential for tools which automate various aspects of the rule eneration process increases.

6. REFERENCES

1. Bates, M. Bobrow, R. Ingria, R., and Stallard, D. "The Delphi natural language understanding system". *Proceedings of the 4th Conference on Applied Natural Language Processing*, 1994.

2. Dahl, D. "Pundit - Natural Language Interfaces'. G. Comyn, N. E. Fuchs, and M.J. Ratcliffe, eds. *Logic Programming in Action*. Springer-Verlag, 1992.

3. Finin, T. , *Frizson, R., Mckay, D., and McEntire, R.* "KQML- A language and protocol for knowledge and information exchange". *Proceedings of the 13th International Distributed Artificial Intelligence Workshop*, July 28-30, 1994.

4. Hirschman, L. Bates, M, Dahl, D, Fisher, W. Garofolo, J., Pallett, D., Hunicke-Smith, K.,Price, P., Rudnicky, A., and Tzoukermann, E. "Multi-Site data collection and evaluation in spoken language understanding". *Proceedings of the ARPA Human Language Technology Workshop*, 1993.

5. Palmer, M., Dahl, D., Schiffman, R., Hirschman, L., Linebarger, M., and Dowding, J. "Recovering implicit information". *Proceedings of the 24th Meeting of the Association for Computational Linguistics* , 1986.

6. Yankelovich, N., Levow, G., and Marx, M. "Designing Speech Acts: Issues in User Interfaces". *Proceedings of CHI-95*,1995.