The Syrinx Spoken Language System

to appear in the

International Journal of Speech Technology

vol 5(1), January 2002

Dominique Estival

sayso!

Level 8, 32 Arthur St

North Sydney 2060

AUSTRALIA

e-mail: Dominique.Estival@ sayso.com

phone: +61-02-9779-5555

fax: +61-02-9779-5477

Dominique Estival
Syrinx Speech Systems

Abstract

This paper describes the Syrinx Spoken Language System (*Sylan*), an automated dialogue system that is fully integrated with the Syrinx Large Vocabulary Speech Recogniser (*Sycon*) into the Syrinx *SpeechMast*er platform. This platform combines speech recognition, natural language processing, dialogue management, telephony and database integration into a robust and flexible Voice User Interface that permits the deployment of natural language dialogue systems in automated call centres. We first describe the architecture of *Sylan* which, being modular, allows us to build a system whose domain-independent components are reusable from application to application. We then present those components from the point of view of application developers, describing the data structures used by the system and the utilities to build them. The two prototypes which have already been developed using *Sylan* are briefly presented, and we conclude by drawing the lessons learned along the way and pointing to further research directions.

Keywords

automated dialogue system, spoken language processing, modular architecture

1. Introduction

There are severe constraints on automated call centre applications: unlike research systems which can be limited to a small domain, or conversely can be allowed greater freedom because they can afford to fail, commercial systems to be used at a call centre must work in real-time over the public telephone system and must be usable by the general public. Thus, by definition, they require a speaker-independent speech recognition system, and they must be extremely robust, allowing a wide range of users to access the system and successfully complete calls and transactions with it (Glass, 1999). To be commercially viable, the system also must be easily portable to different environments, and it must allow quick development of new applications in different domains. Due to these constraints, current commercial systems are usually limited to single token (this may be a phrasal token, rather than a single word) recognition and fairly directed dialogue structures.

Sylan is an automated dialogue system that was recently developed at Syrinx Speech Systems under the Natural Language Processing Project[1]. The aim was to confront these issues and to produce a framework upon which to build applications where the structure of the dialogue can be less constrained, allowing users to input more natural, multi-token utterances that are interpreted and processed in several stages. This framework was developed independently of any particular speech recogniser and can be used with other off-the-shelf speech recognisers, as well as with textual input. However, it is fully integrated with the latest Syrinx large vocabulary speech recogniser (*Sycon3*), and together these components have been integrated into the *SpeechMaster* platform.

The *SpeechMaster* system combines speech recognition, natural language processing, dialogue management, telephony and database integration. It provides a software architecture and an environment for the creation of dialogue systems in natural spoken language for automated call centres. The general system architecture we adopted is described in section 2. This architecture, where a dialogue module acts as the central component directing the conversation flow, deciding which actions should be taken and what responses should be generated on the basis of the interpretation of the input utterance into a set of attribute-value pairs, is similar in design to the familiar one presented by other researchers in similar projects (see Bernsen, Dybkjaer and Dybkjaer, 1998).

In section 3, we present the different components from the point of view of the application developers, in particular the Dialogue Flow Controller, the Analysis module and the Generator, and we also describe some of the tools and utilities needed for application development. We briefly describe in section 4 the development methodology followed and the evaluation metrics we devised, and we conclude by presenting in section 5 our first applications, the Olympics InfoLine and the HomeBanking system, and by pointing out future research directions in section 6.

2. Architecture of the Spoken Language Processing System

The general architecture of the spoken language processing system can be divided into two main components, the Utterance Processing component, which processes each utterance from the caller, and the Dialogue Processing component, which puts that utterance in the context of the conversation and determines the appropriate response. Figure 1 shows the general architecture of the spoken language processing system, abstracting from the specifics of the speech recognition and telephony components. Rectangular boxes represent software components, file icons represent the data structures to be provided by the application developers, and ellipses represent the data structures created and manipulated by the system during processing. We omit from this diagram the representation of the tools and utilities.

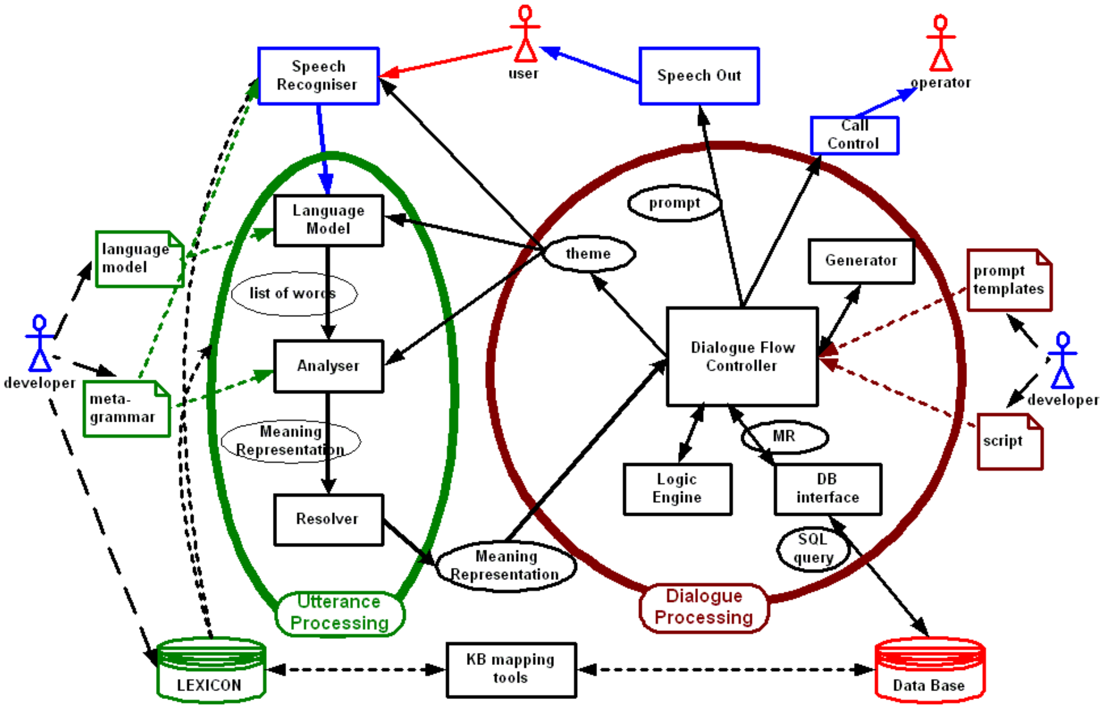

Figure 1: Architecture of Sylan, the Syrinx Spoken Language System.

As can be seen from Figure 1, the Utterance Processing component uses a pipe-line architecture, with the output of the Language Model feeding into the Syntactic Analyser and the output of the Syntactic Analyser feeding into the Semantic Resolver. The architecture of the Dialogue Processing component is centred around the Dialogue Flow Controller which directs the flow of information among the Logic Engine, the Database Interface and the Generator.

When a call comes in, the *SpeechMaster* telephony component initiates a session and hands over control to the Dialogue Flow Controller (DFC), which then sends the first prompt to be played and initiates recognition. The dialogue for a specific application is specified in a dialogue script. Typically, the system starts the call with a greeting and a message prompting the caller to ask a question or formulate a request for a transaction, as in (1), which starts the dialogue for an Olympics InfoLine session.

(1) S: This is the Syrinx Olympics InfoLine! I have information about the Olympics schedule. What would you like to know?

When the caller answers the prompt, the input is processed initially by the Speech Recogniser, which filters out irrelevant background noise from the caller's utterance and produces an N-Best list of results. The Language Model then rescores and selects one or more results from the N-best list.[2] In the current implementation, the Language Model is based on a backoff trigram model (Jelinek, 1997), incorporating both acoustic and linguistic scores.

The result chosen by the Language Model is passed on to the Syntactic Analyser as a list of words linked to their lexical entries in the Lexicon. Because our emphasis is on robust processing of the input, and not on producing a complete parse of the input, we have adopted a chunk parser approach (Abney, 1991; 1995) and the Analyser will ignore words or phrases in the input which do not contribute to the semantic meaning to be used by the application. The output of the Syntactic Analyser is passed on to the Semantic Resolver for further processing. At present, the Semantic Resolver mainly deals with the processing of date and time phrases, and it resolves these expressions as complete date and time objects.

When the caller's response to this first prompt has been processed by the Utterance Processing component, the output to the Dialogue Processing component is a Meaning Representation (*MR*), which consists of a structured list of attribute-value (AV) pairs, the minimal information elements in feature-based formalisms.[3] For instance, if the caller answers the initial prompt given in (1) with the information request in (2),[4] the resulting *MR* will be as shown in (3), where each attribute-value pair represents one piece of information extracted from the input.

(2) U: What time is the women's butterfly?

(3)
```
utterance-type = query
target = time
sub-discipline = butterfly
gender = womens
```

Not only is the *MR* the interface between the Utterance Processing and the Dialogue Processing components, it also serves as the general data structure holding the contents of the utterance during processing. The *MR* resulting from processing an utterance may be used directly by the DFC to construct a query to the customer database through the Database Interface. For instance, from (3), the SQL query in (4) will be derived automatically.

(4)
```
SELECT TIME FROM SWIMMING WHERE SUBDISCIPLINE = BUTTERFLY
AND GENDER = WOMENS
```

In other instances, the *MR* resulting from processing an utterance can be used by the Logic Engine to modify the *MR* that had been stored by the DFC from the previous utterance. For instance, if the system asked the caller to further specify the "women's butterfly" with the request in (5), and the caller gave (6) as an answer, the new *MR* would be as in (7).

(5) S: There are many answers about women's butterfly, please specify a heat

(6) U: The finals.

(7)

```
utterance-type = query
target = time
sub-discipline = butterfly
gender = womens
heats = finals
```

As in most attribute-value based systems, where a feature matrix can be specified to a greater or lesser extent, the *MR* is an under-specified structure: attribute-value pairs are not required to be present, and values may be left unspecified. The *MR* can be modified and updated throughout the dialogue. The Dialogue Processing component may replace an old *MR* with a new *MR*, modify the old *MR* with the new *MR*, or decide to keep the old *MR* and ignore the new one.

Whatever operations may have been performed on the *MR*, the output of the Dialogue Processing component consists of two elements:

a) the prompt to be played to the caller via the Speech-Out component; this could be a pre-recorded file, a text string sent to a Text-to-Speech (TTS) system, or a combination of audio and TTS, with appropriate mark-up;

b) a message to the Utterance Processing component, setting up the appropriate parameters for the next utterance, including pointers to the grammar and vocabulary to be used at the next dialogue state.

If the DFC encounters problems it cannot solve, the call is passed to a human operator via Call Control. When the call either is successfully completed or interrupted, control is regained by the telephony component.

3. Creating the application

In section 2, we described the general architecture of the spoken language system from the point of view of processing, showing what happens when a call is being processed. In this section, we present *Sylan* from the point of view of the application developer, and in particular the data structures that need to be specified for each application, and we briefly describe the tools and utilities used by the application developer to specify those data structures.

From the point of view of the application developer, the process starts with designing the dialogue and defining the attribute-value pairs that are relevant for the application. The first task is usually accomplished from the specifications and requirements provided by the customer, while the second task requires specific information from the customer's database and from the business rules that have been defined for the application. As seen in section 2, the architecture of the Syrinx spoken language system is domain-independent, and *Sylan* is designed to be easily adaptable to different domains for different applications. A spoken language application can be more or less sophisticated, depending on how large the domain of the application (and the associated lexicon) is, and how free the dialogue is allowed to be. However, the general architecture remains the same, whether for tourist information, bill payment, information about a specific event, travel reservation, etc.

To develop an application, developers need to:

a) design the dialogue script;
b) specify the prompt templates;
c) determine the attribute-value pairs for the *MR*, from the customer database and a corpus analysis;
d) enter the words in the Lexicon as lexical entries, with the Part-of-Speech (POS) tag for analysis (see section 3.3. and 3.4), and the semantic values if they are used to build the *MR*; and
e) specify the grammars for the input expected at the different dialogue states.

The dialogue script is designed from the specific application requirements (Balentine and Morgan, 1999). The lexicon and the grammars are built from an analysis of the customer's corpus (Young and Bloothooft, 1997). This corpus analysis, in conjunction with the requirements for the application tasks, will contribute to the specification of the attribute-value pairs for the *MR*. Other attribute-value pairs are derived from an analysis of the customer's database. In turn, this analysis also contributes to the task specifications (Rudnicky et al., 1999). From the corpus, the developer can also train application-specific language models.

The creation, development and maintenance of these data structures necessitate their own tools, and we have developed a number of tools and utilities for our in-house application developers. These utilities can also be made available to customers so they can modify their application. Modifications to the lexicon, the grammars, the dialogue script, and the prompts can be made without having to recompile the application, an important consideration when the application is already running at the customer's site.

3.1 Dialogue Script

The Dialogue Flow Controller manages the dialogue via a dialogue script, which embodies a state machine describing the possible paths through a call and the actions to be taken at each dialogue state. Current dialogue scripts handle 'protocol' interactions such as greetings, closures, interruptions, error handling, etc., as well as the actual question or transaction request from the caller. The dialogue scripts are developed through a GUI, and the scripts which are created are then interpreted at run-time by the Dialogue Flow Controller.

According to the specific dialogue script for an application, the Dialogue Flow Controller can query the customer or domain database to retrieve the answers to the caller's request. The database interface (DBI) is generic so the system can be adapted easily to different domains. The database itself is organised into concepts which are used by the Knowledge Base Mapping Tools to map the database elements in the customer database to lexical entries in the Lexicon.

The Dialogue Flow Controller may call on the Logic Engine to return an answer from the results returned by the database. The Logic Engine may in turn request further information to narrow the caller's request, until enough information has been gathered to provide an answer. Other functions of the Logic Engine are: 1) to sort and/or aggregate multiple answers from the database for use by the Generator in producing responses; 2) to narrow under-specified queries; and 3) to create a new *MR* from an incomplete one by using attribute-value pairs from the previous query.

Finally, the Generator produces the system's responses. The responses are partly specified by the developer in *templates* that are called from the dialogue script and instantiated during processing.

3.2 Prompt templates

The Generator generates the output prompts from *templates,* while the choice of *template* is specified in the script. This approach separates the work of the dialogue writer into two parts:

1) describe the logic in the dialog script: decide *what to say*;

2) specify response templates: decide *how to say it*.[5]

The calls to the templates are parameterised, to permit the use of predefined functions that can filter answers from the Database or reorganise the presentation of answers. These functions make use of attribute-value pairs in the Meaning Representation being processed. Example (10) would cause a call to the function "`makeProbe`", shown in a slightly simplified form in (11).[6] The function "`makeProbe`" provides: a) the template ID ("specify"), b) the current *MR* ("MRin") from which the variable "%$given" will be extracted, and c) the variable "%$discriminator". The template "specify", given in (12), produces the response shown in (13).

(10) U: When are the 100m swimming finals?

(11) makeProbe("specify", *MR*in, DBresult, "Gender")

(12)
```
$template specify
"There are many answers about %$given, please specify a
%$discriminator"
$end
```

(13) S: "There are many answers about 100 meters swimming finals, please specify a gender"

The prompts themselves may be pre-recorded, and the individual elements will then be concatenated by the SpeechOut component. Alternatively, an integrated off-the-shelf Text-To-Speech component provides a larger range of possibilities in the responses, allowing the dialogue designer greater flexibility to modify the prompts.

3.3 Lexicon

The Lexicon contains all the lexical items needed for an application. Several types of information are associated with each lexical entry. Of the fields shown in Table 1, (a) and (b) *phonemic transcription* are currently used by *Sycon*, while (a) *orthographic spelling* and (c-f) *domain, POS, Semantic Type,* and *Semantic Value* are used by *Sylan*. *Syntactic Type,* (g), has not been used in any of the applications developed so far, but is available for further specification (e.g., indication of the valency of predicates). *Domain,* (c), is used to distinguish between lexical items that have to be interpreted differently in different applications (e.g. “butterfly” and “heat” in the sports domain).

Type of information	**obligatory/optional**
a. orthographic spelling	obligatory for recognition
b. phonemic transcription	obligatory for recognition
c. Domain	optional (useful for reuse of lexicons)
d. POS (part of speech)	obligatory for analysis
e. Semantic Type	optional (useful for building AV pairs)
f. Semantic Value	optional (useful for building more fine-grained AV)
g. Syntactic Type	optional (useful for more detailed analysis)

Table 1: Types of information in lexical entries

The *Vocabulary Development Tool* provides an interface and utilities allowing both developers and customers to maintain the lexical database, add or modify lexical entries, extract sub-dictionaries for specific grammars, and compile the dictionaries for an application.

3.4 Grammars

As mentioned above, our emphasis is on robust processing of the input, and the Analyser processes phrasal chunks. The output data structure for these phrasal chunks consists of attribute-value (AV) pairs specifying the semantic meaning of the phrases that have been recognised and processed, i.e., the *MR* (see section 2). The phrasal chunks to be processed are specified in the analysis grammars developed for each application.

Analysis grammar rules consist of a LHS (left-hand side), which is a non-terminal element (i.e., a phrase), and a RHS (right-hand side), which is a list of POS (Part-Of-Speech) terminals and/or phrasal non-terminals. The POS labels are arbitrary, and those for content words will, by and large, correspond to their semantic interpretation for a given application, e.g., "DISCIPLINE" for the sports in the Olympics InfoLine system. The POS for grammatical and closed class items have been taken from the widely used University of Pennsylvania Treebank POS tagset (Marcus et al., 1993).[7] Some Penn Treebank POS labels, used in the rules (15-18) below, are shown in (14).

(14) CD = cardinal number (*1, 2*…)
WRB = question-word (*where, when, how, who, what*…)
TUNIT = time unit (e.g., *day, month*…)
NNS = plural noun (would be instantiated to the semantic labels for a particular application)

The body of a rule consists of conditions, used to check the semantic type and/or value of RHS elements, and actions, which create the attribute-value pairs to be stored in the *MR*. The variable "x.t" refers to the semantic type of the RHS element "x", while the variable "x.v" refers to its semantic value.

Phrasal chunks can be full or partial Noun Phrases (NP) for the entities and concepts from the application domain, for instance (15) and (16) specify measure phrases relevant for the sport domain.

```
(15) # for phrases such as "1 kilometre, 1500 metres"
DIST ->     CD UNIT
            if 2.v eq "m" or 2.v eq "km"
            then distance = 1.v + 2.v
            end.

(16) # for phrases such as "1 kg, 97 kg"
WEIGHT ->   CD UNIT
            if 2.v eq "kg"
            then weight = 1.v + 2.v
            end.
```

Phrasal chunks can also consist of general phrases for questions, or dates and time phrases, which are reusable across applications, as in (17) or (18).[8]

```
(17) # for phrases such as "how many times"
WHP ->      WRB MANY NNS
            then utterance-type = QUERY; target = number; entity = 3.v
            end.

(18) # for dates in the format "4th of July"
DATE ->     ORDINAL IN TUNIT
            if 3.t eq MONTH
            then create DATE; dayofmonth = 1.v; month = 3.v
            end.
```

In the first few iterations of the project, analysis grammars were developed separately from the grammars used by the recogniser. This made sense to the extent that a recognition grammar specifies the string of words to be recognised, while an analysis grammar extracts meaningful chunks from the input string. However, this was not optimal for the application developer who had to keep track of two grammars and make sure they remained synchronised in case of modifications. We now specify both the input string (the “recognition grammar”) and the phrasal chunks with their associated attribute-value pairs (the “analysis” grammar) in the same “meta-grammar”. Meta-grammars are written in the Java grammar script format, which is a convenient format for the integration of the notation used for phrasal chunks such as those shown in examples (15-18) and the BNF notation used for recognition grammars.[9] The meta-grammars are then directly compiled into the BNF file format used by the recogniser and the Syrinx proprietary format used by the Analyser. A simple example of a meta-grammar rule is given in (19.a), with the corresponding BNF recognition rule in (19.b) and the analysis rule in (19.c). Note that (19.b/c) are automatically produced, and the grammar writer only specifies (19.a).

(19.a) meta-grammar rule:

```
<how_much> {HOW_MUCH}=
how much {utterance-type = Query-Type;
Query-Type = Information;
Information = Balance}
```

(19.b) recognition rule:

```
$how_much =
how much;
```

(19.c) analysis rule:

```
Level 1
HOW_MUCH -> WRB MUCH
then utterance-type = Query-Type;
Query-Type = Information;
Information = Balance
end.
```

There is currently no morphological processing performed on the input, as this has not been found to be needed at this stage for current applications. For English, listing singular/plural nominal forms and the different verbal forms as separate lexical entries has been sufficient. In fact, it is an open question whether including a morphological processing stage would lead to an increase in either efficiency or accuracy. In most cases, morphological alternants will result in the same information, e.g., *women/women's* will both give "GENDER = WOMEN". Moreover, morphological alternations are often realised as suffixes, which are often difficult to recognise reliably in the first place. Even for languages other than English, where morphological information may be more easily recognised acoustically, it is not necessarily the case that an extra level of analysis would be warranted. We are leaving this as an area for further research.

3.5 Database Interface

The first prototype for *Sylan* was developed for the "Olympics InfoLine", a demonstration system for information about the schedule of the Sydney 2000 Olympic Games. Another application deployed by Syrinx is a financial trade system, "VoiceBroker", for the Commonwealth Bank, one of the major Australian banks. While the "Olympics InfoLine" application answers queries about time and location of Olympic events from the schedule published on the Web, the "VoiceBroker" application allows callers to obtain current stock prices and to trade (buy or sell) stocks on the Australian Stock Market (Berry and Estival, 2000). Thus, part of the challenge for "VoiceBroker" is the link to a dynamic database. We thus developed a generic database interface (DBI) allowing easy access to any customer database. One of the lessons learned during the development of the "Olympics InfoLine" was that the complexity of the domain database and the complexity of the domain itself contribute significantly to the difficulty of application development, and we are developing tools for the automatic extraction and mapping of database concepts to lexical information.

3.6 Other features and utilities

Although the spoken language processing component of the system is compatible with other speech recognition systems, we assume that the speech recogniser to be used is speaker independent and that it accepts continuous speech. In addition to these characteristics, the Syrinx Speech Recogniser also provides a barge-through facility to allow callers to interrupt system prompts with new queries or responses, and we take advantage of the barge-through facility in dialogue design. The Syrinx Speech Recogniser also gives as output an N-Best result list with confidence scoring, which is used by the Language Model in calculating the overall score (see section 2).

For applications that require it, in particular, financial applications, a Speaker Verification facility has been integrated into the overall system architecture, allowing confirmation of the callers' identity before they are permitted to continue.

4. Development methodology and Evaluation

The architecture of the Syrinx spoken language system is modular, and all components have been designed following Object Oriented methodology. The system is implemented in C++ to allow for easy deployment across different platforms and operating environments (including Windows NT and UNIX), internet and telephony networks, and Automatic Call Distribution (ACD) and Interactive

Voice Response (IVR) platforms. It is scalable, with installations ranging from a single PC with several ports to a clustered environment with several hundred ports.

Evaluation was conducted during development using the EAGLES methodology (EAGLES, 1995). We instantiated the EAGLES "7-step recipe" which lays out the "7 major steps necessary to carry out a successful evaluation of language technology systems or components" (King, 1999). One of these steps is the identification of evaluation criteria, another is of the identification of metrics for those criteria. To evaluate spoken dialogue systems beyond mere word recognition accuracy, we decided to focus on the six criteria or features given in (20):

(20) 1. Speed: Is the spoken language system capable of answering queries in real-time?

2. Access: How many people can call at the same time?

3. Accuracy: of recognition and analysis of input. The accuracy of the spoken language system depends in the first place on the accuracy of the front-end acoustic recogniser.

4. Correctness: Is the information provided to the caller correct and complete? Is it the information the caller was asking for?

5. Ease of use: Does the user need to be trained in using the system? How easy is it to get an answer to a query?

6. Robustness: How does the system handle system errors and how does it recover?

These 6 criteria are further detailed as shown in Table 2.

Speed The whole system must be quicker than the current procedure using touch-tone or IVR with word-by-word speech recognition. There could be a trade-off, if the improvement in ease of use and the flexibility of the dialogue consistently allows callers to get more information or perform more operations than with IVR. However, the system must run in real-time, where real-time is taken to mean that "its latency (time after utterance is complete) is independent of utterance duration" (Glass et al., 1999).
Access The system must be able to handle several calls in parallel.
Accuracy: The system must recognise all the words in the input that are relevant to the query, group them correctly, and formulate the appropriate SQL query. Accuracy must be divided into (a) accuracy of recognition: not part of the NLP system, but relevant for evaluation of the application; and (b) accuracy of analysis.
Correctness of information The information returned to the caller must be the complete correct information retrieved from the database, and must be the information relevant to the query.
Ease The system must be at least easier to use than IVR. The dialogue must be ”natural”. Ease of use may be divided into main aspects: • the results must be presented in a form usable by the caller: - quality of speech output (TTS or recording) - structure of response: presentation of each answer and presentation of several answers • interaction with the caller: - must allow caller to modify the query or make another query - must allow caller to adjust the query, e.g., specify more information to narrow the query - must handle recognition and understanding problems in a "user-friendly" manner; standard error handling: how many times do you ask the caller to repeat or rephrase a query before

	handing the call to a human operator
Robustness:	Monitor unrecognisable queries and system errors

Table 2: Criteria for the evaluation of a spoken language dialogue system

The complete set of measures for these criteria is listed in Appendix A. For each of the criteria, these measures provide three levels of evaluation: good, satisfactory and unsatisfactory. Results during development, and for the prototypes we are currently demonstrating, have been mostly satisfactory. Recognition is, of course, expected to improve once the systems have been operational and more training data have been collected.

5. Applications

Two prototype applications with full dialogue scripts, grammars, lexicons and database interfaces were built and fully integrated in the *SpeechMaster* platform; they are currently available as demonstration systems. While the Olympics InfoLine system has remained very much an in-house research prototype, the HomeBanking system is now being developed into a commercial application. The demonstration prototype can be used remotely over the public telephone to a server at Syrinx, or as a self-contained system on a stand-alone machine, with its own telephone and PABX.[10]

The Home Banking application allows callers to perform 3 types of operations:

§ Account Details, for various accounts (savings, cheque, credit card).

§ Funds Transfer from, and into, any selected account.

§ Bill Payment (BPay), from selected account to various billers.

The application allows the caller to say what they want to do in a fairly unconstrained manner, e.g., either giving all the information at once, or in different combinations. It confirms the transaction and asks for clarification, if necessary, and allows the caller to perform any number of transactions per call. There are a number of important issues concerning dialogue design (see Gorin, Ricardi and Wright, 1997 inter alia) that are outside the scope of this paper, but which must be addressed during commercial application development (see Berry and Estival, 2000).

From the customer point of view, *SpeechMaster* allows for a better use of human resources, as staff at the customer site can focus on providing better service for more complex transactions, rather than handling routine and mundane caller enquiries. Automation permits full-time operation for a call centre, with prompt answering of all incoming calls 24 hours a day, and allows for automatic and efficient peak load management.

6. Further research

While the applications developed have so far been in the English language, we are investigating multi-linguality issues, and are looking at the development of speech and language components for other languages, in particular Cantonese, as well as other European languages.

Another research direction concerns parallel processing. We are first looking at allowing parallel processing at the Utterance Processing level, allowing several of the N-best results to get analysed before a choice is made between the resulting *MR*s. Next, we can allow several *MR*s to be processed, with the best fit with the current dialogue chosen by the Dialogue Processing component. We are also pursuing the use of statistical grammars for analysis and, eventually, statistical dialogue processing.

In a separate research project, we are investigating the use of prosody to enhance automatic speech recognition and understanding of telephone dialogues, in the first instance by assisting in the determination of phrase boundaries and, in the longer term, to assist in dialogue processing.[11] In the mean time, we have been studying other ways in which linguistic information can be used to improve recognition by rescoring recognition results. This linguistic information includes word- and class-based language models (Samuelsson and Reichl, 1999), check digit sums, semantic constraints imposed by the Dialogue Flow Controller (Hagen and Popowich, 2000), knowledge of language repair strategies and rate-of-speech modelling. Under speech repair, we have been particularly interested in investigating disfluencies (e.g., fillers, hesitations, repetitions, corrections, editing terms) in a corpus of spontaneous telephone transactions in order to build models to improve automatic speech recognition and understanding.

Finally, we have been experimenting with ways to automate the grammar creation process, through automatic grammar induction from a corpus of utterances. This automatic grammar induction process eventually will become part of a set of automatic data creation utilities, to help the application developer easily create and manage the different types of data (e.g., lexical information, language modelling, attribute-value pairs (from both corpus and database) and grammars) that are needed for an application.

On another front, we are moving towards a VoiceXML implementation of the Dialogue Processing component.[12]

References

Abney, S. (1991). Parsing by chunks. in R. Berwick, S. Abney and C. Tenny, Eds., *Principle-Based Parsing.* pp. 257-278. Dordrecht: Kluwer Academic Publishers.

Abney, S. (1995). Chunks and dependencies: Bringing processing evidence to bear on syntax. in J. Cole, G.M. Green and J.L. Morgan, Eds., *Linguistics and Computation.* pp. 145-164. Stanford: CSLI Publications.

Balentine, B. and Morgan, D.P. (1999). *How to build a speech recognition application: A style guide for telephony dialogues.* San Ramon: Enterprise Integration Group Inc.

Bernsen, N.O., Dybkjaer, H. and Dybkjaer, L. (1998). *Designing interactive speech systems: from first ideas to user testing*. Berlin: Springer-Verlag.

Berry, L. and Estival, D. (2000). Moving on from IVR. Proceedings of OZCHI 2000. pp. 166-168. Sydney: HSIG.

EAGLES (1995). *Evaluation of natural language processing systems.* EAGLES Document EAG-EWG-PR.2. Geneva: EAGLES.

Glass, J.R. (1999). Challenges for Spoken Dialogue Systems. Proceedings of ICASSP '99. Phoenix, AZ: IEEE.

Glass, J.R., Hazen, T.J., and Hetherington, I.L. (1999). Real-time telephone-based speech recognition in the Jupiter domain. Proceedings of ICASSP '99. pp.61-64. Phoenix, AZ: IEEE.

Gorin, A.L., Riccardi, G. and Wright, J.H. (1997). How may I help you? *Speech Communication*, 23:113-127.

Hagen, E. and Popowich, F. (2000). Flexible Speech Act Based Dialogue Management. Proceedings of *First SIGdial Workshop on Discourse and Dialogue.* ACL 2000. pp. 131-140. Hong Kong: ACL.

Jelinek, F. (1997). *Statistical methods for speech recognition.* Cambridge: MIT Press.

Johnson, M. (1988). *Attribute-Value logic and the theory of grammar.* Stanford: CSLI Publications.

King, M. (1999). *The 7-step recipe*. EAGLES Evaluation Working Group. Geneva: EAGLES. http://issco-www.unige.ch/projects/eagles/ewg99/7steps.html

Reiter, E. and Dale, R. (2000). *Building natural language generation systems.* Studies in Natural Language Processing. Cambridge: Cambridge University Press.

Rudnicky, A.I., Thayer E., Constantinides P., Tchou C., Shern R., Lenzo K., Xu W. and Oh A. (1999). Creating Natural Dialogs in the Carnegie Mellon Communicator System. Proceedings of *Eurospeech '99*. vol.4, pp. 1531-1534. Budapest: Eurospeech.

Samuelsson, C. and Reichl, W. (1999). A class-based language model for large-vocabulary speech recognition extracted from part-of-speech statistics. Proceedings of *ICASSP '99*, pp. 537-540. Phoenix, AZ: IEEE.

Marcus, M.P., Santorini, B. and Marcinkiewicz, M.A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. Computational Linguistics, 19:2, pp. 313-330.

Young, S. and Bloothooft, G., Eds. (1997). *Corpus-based methods in Language and Speech Processing.* Boston: Kluwer Academic Publishers.

Appendix A: Criteria for the evaluation of a spoken language system

Speed: response time after utterance is complete	
good: satisfactory: unsatisfactory:	< 1 second < 5 seconds ≥ 5 seconds
Access: number of calls which can be handled at the same time	
good: satisfactory: unsatisfactory:	10 or more users > 5 users < 5 users
Accuracy	
a) accuracy of recognition is not part of the NLP system, but is relevant in evaluating the system: an NLP system must be able to handle bad recognition.	
good:	- system recognises all utterances; all words in lexicon are recognised - system accepts interruptions and makes corrections (i.e., allows barge-in and adds new info to MR) - system allows pauses in the input and still recognises whole utterances
satisfactory:	- when words are not recognised, or query cannot be constructed, system asks caller to repeat or clarify, then hands call to operator if query is still not recognised - system handles pauses as marking utterance boundaries, but can add the new information to the utterance being recognised
unsatisfactory:	- words in the lexicon are not recognised - system doesn't recognise or mis-recognises some words in the input, but - keeps asking caller to repeat (bad "standard error handling"), - doesn't hand call to operator, - doesn't accept corrections from the caller, - doesn't handle barge-in
b) accuracy of analysis:	
good:	- all words important to the query are in the lexicon - all relevant and meaningful utterances are turned into MRs and queries - rules in the grammar apply to create phrases - correct MR is built
satisfactory:	- system handles short utterances with one query, but doesn't handle long, complex or irrelevant utterances: system behaves as if recognition had failed

unsatisfactory:	- system doesn’t handle long, complex or irrelevant utterances, and behaviour is unsatisfactory (see accuracy of recognition above) - system recognises relevant words in the utterance, but doesn’t construct the appropriate query: either the grammar rules do not exist, or they are not applied, and the MR is not built.
Correctness of information: Check result of SQL query with information from Database	
good:	- information is complete and accurate - handles questions about time and/or location - if information is not available, system hands call to an operator
satisfactory:	- information is accurate but maybe incomplete (e.g., time but no location) - if information is not available, system says so
unsatisfactory:	- information is inaccurate - can’t give an answer for only time or location or for both time and location - information is not available, and system doesn’t say so, but gives wrong information or keeps asking for another query.
Ease: Compare with touch-tone, or IVR with word-by-word systems.	
good:	- speech output is clear and natural-sounding - all the information for one answer is presented clearly - when several answers are found, the system asks if the caller wants to hear all of them and, if not, tells the caller how to narrow the query - if the caller interrupts and gives more information, the system first tries to narrow the current query; if not possible, then builds another query - if the caller is misunderstood 3 times, the call is handed to a human operator
satisfactory:	- speech output is not very natural, but intelligible - when several answers are to be given, they are grouped for presentation - if the caller interrupts, the information is used to start another query
unsatisfactory:	- speech output is not intelligible, or too artificial to be acceptable - when several answers are found, the caller cannot narrow the query, or navigate through the answers - if the caller interrupts, the information is ignored - when the caller is misunderstood, the system asks for query to be repeated.
Robustness: Monitor unrecognisable queries and system errors	
good:	- unrecognisable queries: caller is asked to repeat - if the caller is misunderstood 3 times, the call is handed to a human operator - in case of system break-down, the call is handed to a human operator
satisfactory:	- unrecognisable queries: see “accuracy of recognition, satisfactory behaviour” - system break-down: message to caller before hanging up.

unsatisfactory:	- unrecognisable queries: see “accuracy of recognition, unsatisfactory behaviour” - system break-down: no warning or message to caller.

Endnotes

[1] The Natural Language Processing Project (1998-2000) was a Research & Development project at Syrinx Speech Systems, partly funded by a DIST R&D Start Grant (STG00217). I gratefully acknowledge here the contribution of all the members of the NLP group at Syrinx since the beginning of the NLP project. Thanks in particular to Hugo DeVries, Ben Hutchinson and Cécile Pereira, who gave me valuable comments on earlier drafts of this paper.

[2] The actual boundary between the Speech Recogniser and the Utterance Processing component is to some extent arbitrary: on the one hand, the Language Model could be incorporated into the Speech Recogniser, so the Utterance Processing component would take as input the disambiguated result; on the other hand, the Language Model takes into consideration linguistic information which is, strictly speaking, not used by the Speech Recogniser. Moreover, keeping the Language Model separate from the Speech Recogniser allows us to leave open the option of keeping several of the outputs alive and to process them in parallel until more information (whether syntactic, semantic, or even from the dialogue) allows us to disambiguate between them.

[3] Attribute-value pairs correspond to what Nuance calls “key-value” pairs. The term “attribute-value” is much more widely used in Language Processing and we use it to indicate that the structure which is being built (the *MR*) can, if necessary, be more complex and allow for the embedding of attribute-value pairs in a feature matrix (Johnson, 1988). The values themselves are not restricted to strings or integers, but can be complex objects, such as dates, times, or money amounts. The *MR* itself is an instance of a feature matrix.

[4] We follow the convention of prefixing system prompts with “S” and user’s utterances with “U”.

[5] These two tasks correspond to what has been called “microplanning” and “surface realisation”, and the dialogue script itself can be seen as an instance of “macroplanning” (Reiter and Dale, 2000).

[6] In fact, in this example, another function “getDiscriminator” could have been used to retrieve automatically the most likely discriminator from the list of results from the database.

[7] http://www.scs.leeds.ac.uk/amalgam/amalgam/amalghome.html

[8] In fact, application developers do not have to know about rule (15), as complete grammars for dates and times have already been developed and are available for inclusion in application grammars.

[9] The Syrinx recogniser grammars are in the familiar BNF (Backus-Naur Form) format.

[10] The first set-up was used in November 2000, when Optus made a demonstration of Home Banking system to the ANZ Bank, the second set-up was used at the RIAO’2000 conference in Paris (April 2000) and at the OZCHI’2000 conference in Sydney (December 2000).

[11] This is a “Strategic Partnership with Industry - Research and Training” (SPIRT) project (ARC: C00106858) with the University of Technology Sydney and Macquarie University, titled “*Modelling the melody of human speech: profiling intonation for automated telephone systems*".

[12] http://www.voicexml.org/