CASL Single Lecture Notes

SLN - AIB - 07

2006-12-05

Cognitive Autonomous Systems Laboratory
Department of Computer and Information Science
Linköping university, Sweden

Erik Sandewall

Language Processing in Artificial Intelligence

Subject Area: Artificial Intelligence Basic

Date of lecture: 2006-12-06

"Single Lecture Notes" are notes corresponding to one lecture.

Related information can be obtained via the following WWW pages:

Linköping university:	http://www.liu.se/
This course:	http://www.ida.liu.se/~TDDA23/
CASL Group:	http://www.ida.liu.se/ext/casl/
The author:	http://www.ida.liu.se/~erisa/

1 Overview of Language Processing

The word 'language' in this memo refers to natural languages, such as English or Swedish, and not to any kind of programming language or other formal language. Furthermore we shall be concerned with both spoken and written language. For lack of unifying words we use 'text' to refer both to a written text and a period of speech. Similarly, 'dialogue' will refer both to an exchange of spoken phrases and to an analogous exchange of written phrases.

There are four main areas for language processing by computers:

- Search in large collections of textual documents in order to find information that is relevant to a particular query.
- Translation between languages, for example from English to Swedish. Traditionally one has only considered fully automatic translation without human intervention. More recently there has been an interest in computer assisted translation, where the work is shared between the operator-user and the system.
- Requests and queries to services, for example purchasing systems and travel reservation systems.
- Communication with a robot system, including both physical robots and simulated agents in computer games.

The last two of these require that the program in question achieves a fairly good understanding of the incoming sentences. In the first application, for search of large-scale bodies of text, one must instead use other methods, often statistical ones, that only scratch the surface of understanding the contents. The case of language translation is an intermediate one, where some argue that good language translation requires understanding the text and others claim that statistical methods are sufficient for practical purposes.

Artificial intelligence techniques come into play only for the last three applications, and not for text search (except possibly as a way of expressing the queries to the search). Both spoken and written text is of interest. For example, language translation is of interest both for longer texts such as articles and reports, for shorter texts such as email messages, and for speech for example as a component in a telephone system (so that the two parties can speak their respective languages and hear the other party in translation).

In addition there is of course those applications where text is processed without any reference whatever to its contents, for example in word processing and formatting systems, or in a web browser.

For the purpose of this lecture I shall focus on the robot dialogue applications. It is the most complete one and the requirements of the translation and service-request applications are largely a subset of those of the robotic application. For the same reason I shall also focus on the dialogue situation with interaction between the user or operator and the computer system. The language processing system will therefore be referred to as a dialogue system.

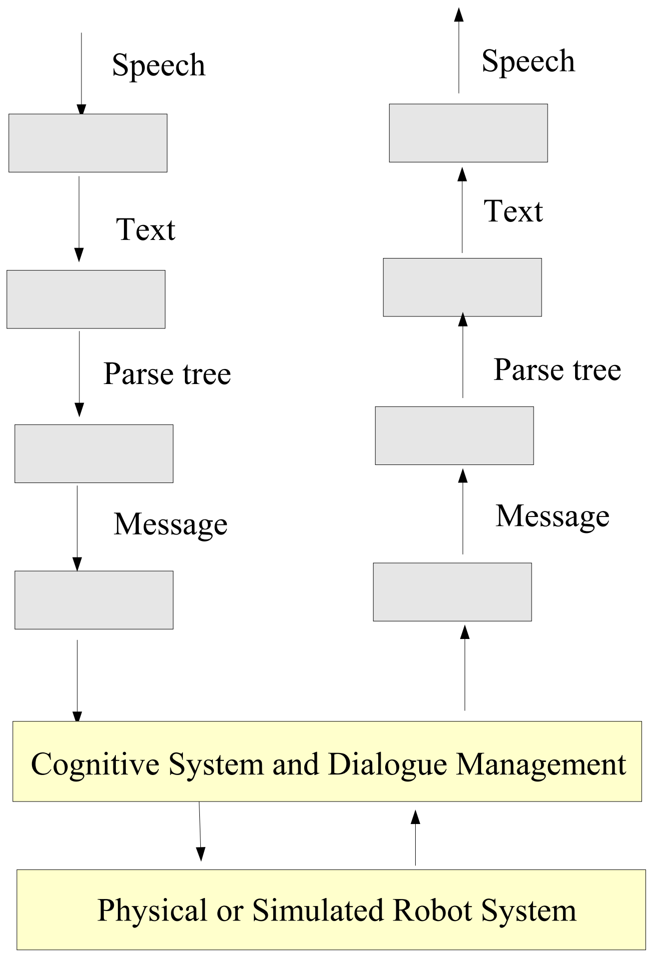

Figure 1: *Standard Architecture for Robotic Dialogue System*

2 Robotic Dialogue Systems

2.1 The Standard Architecture

Sentences that are input to, or output from the dialogue system have to pass through a number of operations that are performed in sequence. (There are some qualifications to 'in sequence' that will be described later on). The *architecture* of the dialogue system specifies what those modules are, and how they are connected by the flow of sentences and other messages.

Figure 1 shows the standard architecture for a robotic dialogue system. Each of the smaller boxes in the diagram represents a transformation from one representation of the sentence to another one, and they are as follows:

- Speech-to-text transformation
- Parsing, that is, transformation from a written sentence in the lan-

guage, to a tree structure that captures the *phrase structure* of the sentence, that is, what words belong together in meaningful groups

- Semantic transformation, where the parse tree is converted to a representation that more closely represents the 'meaning' of the sentence
- Integration with the knowledge base of the dialogue system.

The difference between semantic transformation and integration is somewhat fuzzy, but in general one expects the semantic transformation to have little if any use of the knowledge base, whereas for the very point with integration is to merge the message into that knowledge base. The functions that are naturally included in 'semantic transformation' include simple cases of the following:

- Identifying the *referents* of *anaphoric expressions*, such as pronouns and phrases referring to previously mentioned events, for example 'that' in 'why did you do that' or 'please do not do that'.
- In a multi-language system, converting from the vocabulary in the input language to corresponding words or concepts in an internal representation in the cognitive system.

More complex variants of these operations require the use of the cognitive system as well.

Referring back to the other types of language processing that were mentioned above, in the case of translation the 'robot' module must be omitted from the diagram. Furthermore there are mixed opinions about whether the 'cognitive system' module is needed or not. Some argue that it is sufficient to make a transformation from the parse tree in the source language (or some similar "deep structure") to the corresponding structure in the target language.

In the case of service queries the 'robot' module in the diagram has to be replaced by a 'database' or 'service' module. This tends to be a considerable simplification of the overall system since a robot may take dialogue initiatives whereas a database or service module is usually just reactive. More about this later on.

Furthermore, the speech-to-text and text-to-speech modules are of course omitted for the case of translation of written text and for queries and service requests that are expressed in written language.

2.2 State of the Art

Different modules in the standard architecture differ widely with respect to how easy or difficult it is to implement them. Let us consider it from the point of view of the person who wishes to build a robotic dialogue system for a particular application.

Every application has its own variety of language: what words will be understood by the system, and what kinds of phrases. These requirements can be met either by taking an existing, general purpose program and providing it with appropriate control information, or by implementing a new program for the module in question.

For the 'speech out' module there exists a range of software systems that perform the task for the full language and that come with the required databases of sounds and pronounciation information, and for a number of languages. Not only large languages such as English but also relatively small languages, in terms of numbers of speakers, such as Swedish are supported in this way.

For 'speech in' there also exist software products and as well as open-source systems for a number of languages, but in this case one can only expect to be able to cover a small fragment of the language in question. It is necessary to specify the input vocabulary, and in many cases one also specifies the input grammar in the same way as for the parser.

Both 'speech in' and 'speech out' are complex processes and it is not an easy undertaking to implement one's own. For the parser module, on the other hand, the technology is well known and not too complicated. Using the chart parser method described in the textbook, for example, a participant in this course should be able to implement her own parser in a fairly short time.

Sentence generation from a parse tree is trivial.

The remaining modules are: transformation from parse tree to formal input message, integration of input message into the knowledge base, generation of formal output messages, and converting them to output parse trees. These modules have to be implemented anew for every application.

2.3 Reservations about the Standard Architecture

The standard architecture is straightforward and easy to understand, and it makes it possible to make variations e.g. by omitting 'speech in' and 'speech out' obtaining a system for textual interaction. However there are a number of exceptions to the rule that must be mentioned here.

Integration of 'speech-in' and parser. In many cases the 'speech-in' module relies on the use of a parser. In such cases it is redundand to let it produce a linear sentence that is again going to be parsed; it makes more sense to let the 'speech-in' module deliver a parse tree directly to the semantic transformation.

Omit parse-tree in output chain. The range of output sentences in a robot dialogue system may be fairly restricted. In such cases it may be easier to have one module that generates sentences directly from output messages, rather than going via a parse tree in output.

Question the pipeline structure. The literal interpretation of the standard architecture is that there are a number of modules that communicate only in input-output fashion, that is, that the output sentence, parse tree, or message from one module becomes the input from another one. This makes it difficult to handle situations where an earlier module in the pipeline has use for information that is only available in a later module. One way of meeting this demand is to let modules generate more than one alternative, so that later modules can choose between them according to the information that they have.

A more systematic way of handling such situations is to allow modules to interact in arbitrary directions for the purpose of querying another. In this case they exchange messages that are not transforms of the input or output sentence, but for control purposes. This is a powerful extension of expressivity, but it also increases the complexity of the system and may make the software more difficult to manage. Also, when off-the-shelf modules are used, one can not take for granted that they will support this mode of operation.

Real-time considerations. In a robotic dialogue system where dialogue initiatives may come both from the operator and the robot, there may be situations where 'too many' individual output messages are generated. There may be too many compared to the time it takes for pronouncing them, resulting in a 'traffic jam' in the output channel. They may also be too many in the sense that several parts of the output software may independently generate messages with the same or similar contents. In both cases there is a need for coordination and for merging several messages into one. A practical way to meet this demand is to establish a buffer in the output channel: output messages are sent to the buffer, and a separate module picks messages from the buffer while also rank-ordering them for priority and integrating (or discarding) them when appropriate. This is illustrated in figure 2.

Software management considerations. Consider the case of a robot that has a certain reportoire of actions. For an autonomous UAV these may be actions such as 'take off', 'land', 'fly to d' with various ways of specifying the destination, 'fly along the trajectory t', 'direct the camera to position p', and so on. Each one of these actions will be represented by linguistic expressions and data constructs in several of the modules: in the vocabulary, in the grammar, in formal input and output messages, in the actual UAV control system and in the simulator(s) for it, and so on. When working with the system one often wants to compare and relate the information for a particular type of action in several of those modules.

This is a typical example of the problem one wishes to solve in so-called *aspect-oriented programming* which is a software development technique that has recently attracted much attention. The workflow techniques in office information systems are another potential use of aspect-oriented programming. The natural approach to solve this problem is to have a 'source file' that contains all the grammatical rules, program fragments, etc that pertain to a particular action, and to have a program generation program that redistributes this information to the respective module where it is to be used when the system executes.

3 The discourse level

The standard architecture that was described above focusses on the processing of individual sentences. There are also a number of considerations that arise because of the higher level structure *between* sentences. The word *discourse* refers to that structure and includes both the case of longer texts without interaction (that is, monologue) and the case of dialogue between two or even more partners.

The real-time considerations that were discussed above are a case where discourse-level issues force modifications of the standard architecture itself.

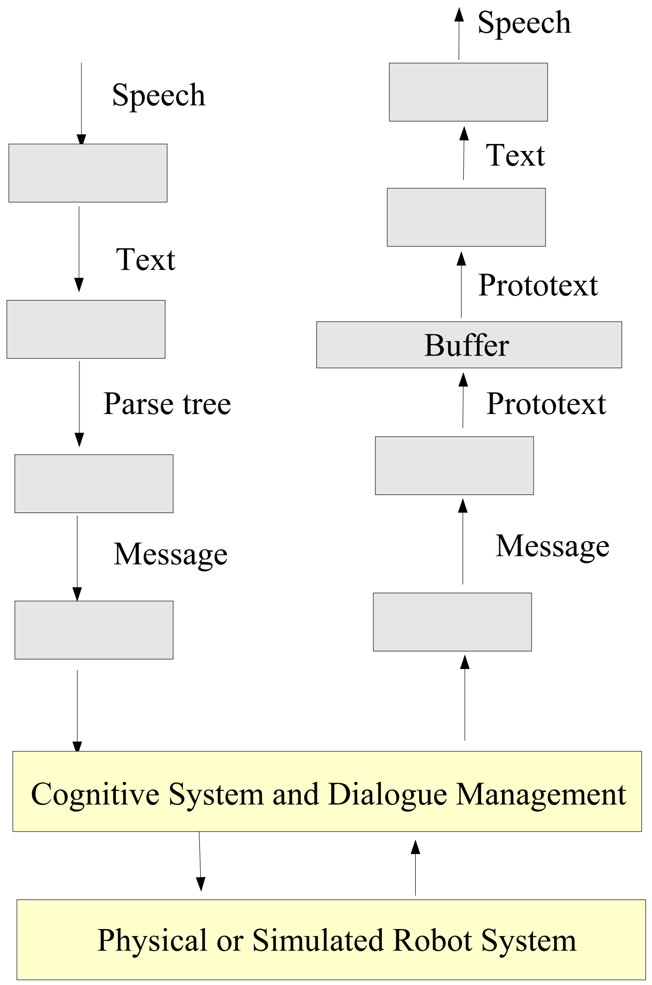

Figure 2: *Architecture with Buffered Output for Robotic Dialogue System*

There are also a number of other discourse-level considerations that are important but that do not modify the architecture; they can be accomodated within some of its modules.

Use of anaphora. Anaphora are expressions within natural-language sentences that refer to previously mentioned objects, events, or other phenomena. They include pronouns ('it'), noun phrases often in the definite form ('the house') or using determinative pronouns ('that house'), and a variety of other and more complex phrases ('the other house'). They apply to events and not only to objects: 'never do that again' and sometimes their anaphoric character is not very evident ('when he arrived the second time').

Relatively simple cases of anaphora are those where the anaphoric referent has previously been mentioned explicitly, and then it is 'only' a question of picking the right one. This often requires the use of semantic information in order to identify what is a meaningful referent; just using the most recently mentioned object is not sufficient.

More complex cases are those where the anaphoric referent is implicit: it has not been mentioned before, but its existence and some of its properties can be derived from available information in the knowledge base.

The proper dealing with anaphora is therefore an important and major function of the 'semantics' and 'integration' modules. Proper use of anaphora in output is likewise a nontrivial problem, at least if one has high requirements on the quality and naturalness of the output phrases.

Turntaking. Turntaking refers to the rules for which one of several participants in a dialogue is going to speak at a particular instance. It is constrained both by the contents of the dialogue, for example when a question is posed to one of the participants and that participant is the most likely one to answer, and by social rules and conventions.

Simple forms of turntaking issues arise already in a dialogue between one operator and one robotic system. In network situations where there are a number of robots and a number of human operators, turntaking becomes a major issue if one allows several of these dialogue agents to interact concurrently and not only locally in pairs.

Multiple threads. A 'thread' in the dialogue is a sequence of interactions and intervening events that belong together in a logical fashion. A question and the subsequent answer constitute a thread. A command to the robot and its successive progress reports about the execution of that action over a period of time also constitutes a thread. This example shows that threads may last over a period of time, and that there may be parts of that period where nothing is said in that particular thread. If several threads occur in parallel, for example if a question is posed while an action is being performed, then there will be points where an utterance in the dialogue belongs to another thread than the preceding utterance. The normal practices of language require us to mark the switch of thread in some way, for example by a 'padding' phrase at the beginning of a sentence that represents switch of thread.

The management of multiple threads applies both to input (understand the parts of input sentences that are motivated by switch of thread) and to output (produces those padding phrases so that the output makes sense). It becomes even more complex when several output sentences are merged into one, as was discussed above for real-time considerations. It also interacts with turntaking: since switch of thread requires a certain mental effort, some preference should be given in a dialogue system to stay with the same thread rather than switching threads.

4 Parsing methods

In addition to the material in this memo, chapter 22 sections 22.1 to 22.5 in the textbook (new edition) are included in the required reading for this course. Particular attention should be given to the chart parser methods and to the extension of that method using features.