Confidence-based Adaptivity in Response Generation for a Spoken Dialogue System

Kristiina Jokinen
University of Art and Design Helsinki
00560 Helsinki, Finland
kjokinen@uiah.fi

Graham Wilcock
University of Helsinki
00014 Helsinki, Finland
Graham.Wilcock@helsinki.fi

Abstract

The paper addresses the issue of how to increase adaptivity in response generation for a spoken dialogue system. Realization strategies for dialogue responses depend on communicative confidence levels and interaction management goals. We first describe a Java/XML-based generator which produces different realizations of system responses based on agendas specified by the dialogue manager. We then discuss how greater adaptivity can be achieved by using a set of distinct generator agents, each of which is specialized in its realization strategy (e.g. highly elliptical or highly explicit). This allows a simpler design of each generator agent, while increasing the overall system adaptivity to meet the requirements for flexible cooperation in incremental and immediate interactive situations.

1 Introduction

When describing the desired characteristics of human-computer interaction, the common keywords are cooperation and naturalness. Cooperation is used to refer to the participants' ability to collaborate with each other on a given task and to provide informative and helpful contributions in a given task context. Naturalness, however, describes the participants' reaction which is appropriate in the current communicative context, and usually presupposes reasoning through which the participants can adapt themselves to the requirements of the situation and to the knowledge level of their partner.

Naturalness in spoken interaction can be characterised by features such as incrementality and immediacy (Jokinen et al., 1998). Speakers exchange information and present the new information in a stepwise manner, constructing a common ground by providing new pieces of relevant information to complete the task that the interaction was initiated for. They monitor their own presentations and react to the partner's contributions immediately, often simultaneously, to prevent potential misunderstandings growing and causing problems to the interaction. Spoken dialogue systems that aim at natural interaction with the user should thus have capabilities for incremental and immediate management of interaction. In other words, they should be able to produce responses that take into account the requirements of an incremental and immediate interactive situation. In this paper we discuss how various types of system responses can be generated taking into account the special requirements of incremental and immediate interaction situations.

The paper is structured as follows. In Section 2 we discuss concrete examples from a spoken dialogue system, in which different forms of surface realization are required in order to achieve interaction management goals. This may involve, for example, deliberate repetition of old information to get implicit confirmation of speech recognition accuracy. When there is a high level of confidence in the effectiveness of the communication channel, however, the preferred form of response realization is to include only the new information in the response.

Section 3 describes an implementation of dialogue response generation based on Java, XML and XSL transformations. The implementation can generate the different realizations discussed in Section 2. The way in which the generator chooses between the different realizations is based on detailed specifications of the information status of different concepts, given in an agenda by the dialogue manager component.

In Section 4 we then discuss an alternative approach to the choice of realization, in which a set of distinct generation agents are used, each agent being specialized in its realization strategy (e.g. highly elliptical or highly explicit). The design of the individual generation agents is therefore simpler, but the system as a whole increases in adaptivity by choosing which agent to use according to the wider dialogue context. We describe a Java and XML-based framework which supports this architecture, and discuss a strategy for adaptivity based on communicative confidence.

2 Interaction Management

In this section we discuss concrete examples from a spoken dialogue system. The domain is public transportation in Helsinki.

2.1 Agenda, NewInfo and Topic

To enable incremental presentation and immediate reaction, Jokinen et al. (1998) structure the context with the help of NewInfo and Topic, so that the generator can distinguish the new information that is meant to be put across in the current dialogue situation, and the background information that the new information is linked to. The pool of contextual information includes an *agenda*, a set of concepts marked with the help of 'topic' and 'newinfo' tags, the tags being determined by the dialogue manager which decides how the system is to react next.

The generator can freely use the tagged pieces of information in order to realise the system's intention as a surface string, but it is not forced to include in the response all the concepts that the dialogue manager has designated as relevant in the agenda. Thus the dialogue manager and the generator communicate via the specifically marked conceptual items in the shared knowledge-base, but they both make their own decisions on the basis of their own reasoning and task management. The dialogue manager need not know about particular rules of surface realisation while the generator need not know how to decide the information status of the concepts in the current dialogue situation.

While the notions of NewInfo and Topic are often used to illustrate the characteristics of word-order variation, their importance in spoken dialogue systems can be mostly shown in the planning process that lies at the border of dialogue processing and tactical generation. Consider first the following questions by the user:

(1) Which bus goes to Malmi?

(2) How do I get to Malmi?

The dialogue manager has analysed them as timetable requests related to the user's going to Malmi. It has also recognized that NewInfo in (1) is the information concerning bus numbers while NewInfo in (2) concerns means of transportation.

In the case of (1), the information that the dialogue manager gets from the task manager which consults the timetable database, is that there is bus 74 that goes to Malmi. The dialogue manager decides to put the following concepts into the agenda in the shared knowledge pool (using XML as discussed in Section 3):

```
<agenda>
  <concept info="Topic">
    <type>means-of-transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
  </concept>
  <concept info="Topic">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>busnumber</type>
    <value>74</value>
  </concept>
</agenda>
```

The dialogue manager also tags the concepts as NewInfo or Topic, reflecting its knowledge of how the concepts relate to the current dialogue situation. The concept 'busnumber' is tagged as NewInfo, and the other three as Topic, since this is a match with the new information asked in the previous utterance, and the user will be likely to link the response correctly.

The generator can then select the concepts and decide the surface realisation as described in Section 4. The simplest realisation is:

(1a) Number 74.

2.2 Indirect Requests

For example (2), however, the situation is somewhat more complicated since the user question can be understood either as a literal question about public transportation to Malmi, or as an indirect request for buses that go to Malmi. Information about the previous dialogue situation must thus be used, and we relate the difference to the Topic of the conversation: in the ’literal’ case, the Topic is Malmi, the destination where the speaker wants to find out transportation for, whereas in the indirect request, the Topic is the bus as a means of transportation to Malmi. Consequently, in the ’literal’ case, the dialogue manager consults the task manager by requesting a value for the concept ’means of transportation’, while in the indirect request, the dialogue manager requests task manager to give a value for the busnumber. In both cases, however, the information that the dialogue manager gets from the task manager is the same: that there exists a means of transportation to Malmi, namely a bus and the busnumber is 74.[1]

When deciding on the response to the ’literal’ case, the dialogue manager regards the means of transportation as NewInfo and the destination to Malmi as Topic, continuing the information structure of the previous question. The two other concepts, ’bus’ and ’busnumber’, are also tagged as new but since the NewInfo that matches the dialogue situation concerns the public transportation to Malmi, the piece of information of the bus number is extra information that can be seen as a sign of cooperation on the system’s side, rather than a necessary new information to be told to the user, i.e. they can be added in the response if the time constraints allow this and the level of cooperation so requests.

The agenda in the shared knowledge pool in case (2a) is as follows:

```
<agenda>
  <concept info="NewInfo">
    <type>means-of-transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
  </concept>
  <concept info="NewInfo">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>busnumber</type>
    <value>74</value>
  </concept>
</agenda>
```

In the case of an indirect request, the dialogue manager again relies on the dialogue context when tagging the concepts for the agenda. The Topic is the means of transportation to Malmi, whereas the NewInfo concerns the busnumber, and so only the ’bus’ and ’busnumber’ are tagged as new. For (2b), the shared knowledge is thus as follows:

```
<agenda>
  <concept info="Topic">
    <type>means-of-transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
  </concept>
  <concept info="NewInfo">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>busnumber</type>
    <value>74</value>
  </concept>
</agenda>
```

The difference in the system responses is reflected in the alternatives (2a) and (2b):

(2a) By bus - number 74.

(2b) Bus 74 goes there.

2.3 Confidence Levels

The next example is related to a different aspect of spoken dialogue systems: confidence in speech recognition results. The dialogue manager gets the recognized words together with their recognition scores, and decides on the appropriate action depending on the confidence levels.

[1]The dialogue manager and task manager communicate with each other via a particular task-form which has as its parameters the concepts important for the task manager to fetch information from the database. If the form is filled in so that a database query can be performed, the task manager returns the form with all the appropriate parameters filled in, and thus lets the dialogue manager decide on the status of the parameters and their values with respect to the dialogue situation.

(3) When will the next bus leave for Malmi?

(a) 2.20pm
(b) It will leave at 2.20pm
(c) The next bus to Malmi leaves at 2.20pm

As is common, we assume that if recognition is above a certain confidence level, the system will use the simplest and most straightforward answer, while if the recognition error becomes bigger, a confirmation strategy has to be used. Thus response (3a) is used when the system has confidence that the user has indeed asked about new information concerning the next bus leaving for Malmi (cf. 1a). Response (3b) is also used in the similar situation where the system is confident about its recognition, but the dialogue situation differs from the one in (3a) in that now the system assumes that the user expects a polite full response, instead of an elliptical answer as in (3a) where the user has talked about the buses to Malmi and just wants to check the next one leaving. The alternative (3c) is used when the system explicitly wants to confirm the Topic (= next bus to Malmi), so as not to allow user to draw false implicatures about which bus timetable the answer concerns.

The agendas for the alternatives (3a) and (3b) are similar, and the difference in the surface realizations is due to the different interaction history: in the former case the Topic continues and the dialogue history contains the concepts *destination* and *bus* as previous Topics, whereas in the latter case, the previous Topics may be different concepts or there may me no previous Topic at all (beginning of the dialogue).

(3a,b)

```
<agenda>
  <concept info="Topic">
    <type>means-of-transportation</type>
    <value>bus</value>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
  </concept>
  <concept info="Topic">
    <type>bus</type>
    <value>exists</value>
  </concept>
  <concept info="NewInfo">
    <type>bustime</type>
    <value>2.20pm</value>
  </concept>
</agenda>
```

Also the agenda for the alternative (3c) looks the same, except for the feature <confidence> which characterizes the system's own evaluation of how confident it is of the correctness of the recognized concepts. The value 1 marks certainty as in the case of *bustime* whose value is retrieved from the database. This feature has been left out in the other representations for the sake of clarity: if the confidence is above the threshold, the concept is treated according to its information status in the shared pool.

(3c)

```
<agenda>
  <concept info="Topic">
    <type>means-of-transportation</type>
    <value>bus</value>
    <confidence>0.6</confidence>
  </concept>
  <concept info="Topic">
    <type>destination</type>
    <value>malmi</value>
    <confidence>0.2</confidence>
  </concept>
  <concept info="Topic">
    <type>bus</type>
    <value>exists</value>
    <confidence>0.6</confidence>
  </concept>
  <concept info="NewInfo">
    <type>bustime</type>
    <value>2.20pm</value>
    <confidence>1.0</confidence>
  </concept>
</agenda>
```

3 Dialogue Response Generation

The system's competence in dialogue management is shown in the two tasks that the system must perform: evaluating the user goal, and response generation. The former results in strategic decision about operationally appropriate goals, while the latter concerns how the same goal can be realised in different ways in different contexts.

We now describe the framework which performs the dialogue response generation. The content determination has been done by the dialogue manager, which has selected the relevant concepts to put on the agenda. The discourse planning is based closely on the specification of Topic and NewInfo by the dialogue manager, but also includes specific decisions by the generator described in Section 3.2. The response generation continues with an aggregation stage, described in Section 3.3, followed by a stage which combines lexicalization and generation of referring expressions, described in Section 3.4. Morphological generation, which is very important for one of the languages generated (Finnish), is done in a separate stage.

The framework is based on Java, XML and XSL transformations. The implemented system can generate the responses which we have discussed. In the next section, we will describe an extension to this framework, suitable for adaptive and flexible response generation.

3.1 A Pipeline Architecture

The generator starts from an agenda of concepts specified in XML, set up by the dialogue manager as shown in the examples in Section 2. The generator produces linguistic output which is also specified in XML, to be passed to the speech synthesizer. We are therefore generating XML from XML. The simplest way to do this is to apply a set of XML transformations specified in XSL (XML Stylesheet Language). We do this using the Xalan XSL Processor (Apache XML Project, 2001) which is open-source software written entirely in Java.

With the Xalan XSL Processor it is easy to set up a sequence of transformations, in which the output of one transformation becomes the input to the next transformation. This kind of "piping" is a natural way to implement the standard pipeline architecture regularly used in natural language generation systems (Reiter and Dale, 2000).

The ease of setting up a pipeline architecture with XSL raises the general question of whether XSL transformations are suitable for wider use in NLG systems. This is discussed by Cawsey (2000), who concludes that relatively simple XSL transformations can be used for generation when the input is fairly constrained, but XSL is not suitable for less constrained input, when we need to turn to general purpose programming languages or NLG tools.

However, XSL can be combined with general purpose programming languages by embedding *extension functions* in the XSL templates. These functions can be written in Java (Apache XML Project, 2001). This means that even where general purpose programming languages are required for specific purposes, such as complex morphology, a pipeline of XSL transformations can still be used as a general framework.

3.2 Focus-based Generation

The model of dialogue response generation which we use is based on generation from the new information focus, as proposed by Jokinen et al. (1998). In this model, response planning starts from the new information focus, called *NewInfo*. One of the tasks of the generator is to decide how to present the NewInfo to the user: whether it should be presented by itself or whether it should be *wrapped* in appropriate linking information.

The wrapping of the NewInfo depends on the pragmatic requirements of the dynamic dialogue context. When the context permits a fluent exchange of contributions, wrapping will be avoided and the response will be based on new information only. When the context requires more clarity and explicitness, the new information will be wrapped by Topic information in order to avoid ambiguity and misunderstanding. When the communication channel is working well, wrapping will be reduced, and when there are uncertainties about what was actually said, wrapping will be increased to provide implicit confirmation.

Typically, XSL transformations are used to convert information content represented in XML into a desired presentation format, for example in HTML. There is usually no need for complex re-ordering of the content. Here however, the generator must convert the unordered bag of concepts in the agenda into a syntactically correct ordered sequence of words to be passed to the speech synthesizer. Also, the new information focus tends to come last in surface order, so the linking information (if any) will generally precede the new information in the surface realization.

Simple reordering can be performed in XSL, for example by using XSL *modes*. We have experimented with applying XSL templates first with a Topic mode (if required), followed by a NewInfo mode. The usefulness of XSL modes for such purposes is noted by Cawsey (2000).

However, as we must also handle detailed syntactic ordering, we use aggregation templates as described in Section 3.3.

If the real-time requirements of the system allow sufficient time, the generator can decide on the optimum way to wrap the new information, but if there is extreme urgency to produce a response, the generator can simply give the new information without wrapping it. If this leads to misunderstanding, the system can attempt to repair this in subsequent turns. In this sense, the model offers an *any-time* algorithm, important for providing incremental and immediate responses for spoken interactive situations.

3.3 Aggregation

The aggregation stage selects those concepts marked as NewInfo as the basis for generation, and also decides whether NewInfo will be the only output, or whether it will be preceded by the Topic linking concepts.

In order to implement detailed syntactic ordering, we use *aggregation templates* as a form of sentence plan specification. The aggregation templates are implemented by means of XSL named templates, as in the following simplified example:

```
<xsl:template name="NUM-DEST-TIME">
<aggregation>
  <tree><node>S</node>
    <tree><node>NP</node>
      <xsl:copy-of select=".
           /concept[type='busnumber']"/>
    </tree>
    <tree><node>V</node>
      <xsl:copy-of select=".
           /concept[type='bus']"/>
    </tree>
    <tree><node>PP</node>
      <xsl:copy-of select=".
           /concept[type='destination']"/>
    </tree>
    <tree><node>PP</node>
      <xsl:copy-of select=".
           /concept[type='bustime']"/>
    </tree>
  </tree>
</aggregation>
</xsl:template>
```

The selected aggregation template creates a new XML document instance, with root node `<aggregation>`. Its child nodes are one or more `<tree>` nodes, containing syntactic categories and other features. The trees contain variable slots, which will be filled in later by the lexicalization and referring expression stages. In the aggregation stage, the concepts from the agenda are copied directly into the appropriate slots by means of `<xsl:copy-of>` statements.

Our aggregation templates are quite similar to the syntactic templates described by Theune (2000). As argued by van Deemter et al. (1999), this kind of syntactic template-based approach, which rather resembles TAG-based generation, is fundamentally well-founded.

The selection of an appropriate aggregation template is based on which concept types are in the agenda and on their information status as Topic or NewInfo. The logic is implemented by means of nested `<xsl:choose>` statements, as in the following example:

```
<!-- CHOOSE TEMPLATE BASED ON AGENDA -->
<xsl:template match="agenda">
<xsl:choose>
<xsl:when test="concept[@info='NewInfo']
           /type='means-of-transportation'">
  <xsl:call-template name="BY-TRANSPORT"/>
</xsl:when>
<xsl:when test="concept[@info='NewInfo']
                /type='bus'">
  <xsl:choose>
  <xsl:when test="concept[@info='NewInfo']
                  /type='busnumber'">
    <xsl:call-template name="NUM-DEST-TIME"/>
  </xsl:when>
  ...
  </xsl:choose>
</xsl:when>
<xsl:when test="concept[@info='NewInfo']
                /type='busnumber'">
  <xsl:call-template name="NUMBER-ONLY"/>
</xsl:when>
...
</xsl:choose>
</xsl:template>
```

Here, if means-of-transportation is NewInfo as in (2a), the template named `BY-TRANSPORT` is selected. If means-of-transportation is not NewInfo, but bus and busnumber are NewInfo as in (2b), template `NUM-DEST-TIME` is selected. If only busnumber is NewInfo, as in (1a), the template `NUMBER-ONLY` is selected.

3.4 Referring Expressions

In the lexicalization and referring expression stages of the response generation pipeline, the concepts in the aggregation templates are replaced by lexical items and referring expressions. In general, concepts which are marked as Topic are realized as pronouns, as shown by the following simplified examples:

```
<!-- REFERRING EXPRESSIONS: PRONOUNS -->
<xsl:template
   match="concept[@info='Topic']"
   mode="referring-expression">
  <xsl:choose>
  <xsl:when test="type='busnumber'">
    <xsl:text> it </xsl:text>
  </xsl:when>
  <xsl:when test="type='destination'">
    <xsl:text> there </xsl:text>
  </xsl:when>
  <xsl:when test="type='bustime'">
    <xsl:text> then </xsl:text>
  </xsl:when>
  </xsl:choose>
</xsl:template>
```

Here, a destination concept marked as Topic is pronominalized as *there*, as in (2b). By contrast, concepts which are marked as NewInfo are realized as full descriptions. In the following template, the destination concept is realized by a prepositional phrase, *to* followed by the text value of the destination placename, obtained by the `<xsl:value-of>` statement.

```
<!-- REFERRING EXPRESSIONS: DESCRIPTIONS -->
<xsl:template
   match="concept[@info='NewInfo']"
   mode="referring-expression">
  <xsl:choose>
  <xsl:when test="type='busnumber'">
    <xsl:text> number </xsl:text>
    <xsl:value-of select="value/text()"/>
  </xsl:when>
  <xsl:when test="type='destination'">
    <xsl:text> to </xsl:text>
    <xsl:value-of select="value/text()"/>
  </xsl:when>
  <xsl:when test="type='bustime'">
    <xsl:text> at </xsl:text>
    <xsl:value-of select="value/text()"/>
  </xsl:when>
  </xsl:choose>
</xsl:template>
```

The above examples are simplified to show simple text output. The final stages of response generation actually perform syntactic and morphological realization producing XML output (SABLE or VoiceXML) which is passed to the speech synthesizer.

4 Confidence-based Adaptivity

In general, when confidence in speech recognition accuracy goes down, the dialogue system needs to adapt by increasing the repetition of old information to check that it is correct. When confidence in speech recognition accuracy is high, the system should adapt by reducing the repetition of old information, given the dialogue context itself does not require this. Normally, with high speech recognition confidence, a fluent dialogue will be made up of responses with only new information.

4.1 A Development Framework

In order to allow this kind of variation in the responses produced, the framework in which the dialogue management is embedded must itself be designed specifically to support adaptivity. One such system is the Jaspis adaptive speech application framework (Turunen and Hakulinen, 2000). Jaspis is a general agent-based development architecture, and on the most general level it contains *managers* which handle general coordination between the system components and functional modules (such as the Input Manager, the Dialogue Manager and the Presentation Manager). Within each manager there are several *agents* which handle various interaction situations, as well as a set of *evaluators* which try to choose the best possible agent to handle each situation. The architecture also exploits a shared knowledge-base called the Information Storage, where the information about the current state of the system is kept and which each of the agents can read and update.

The adaptivity-oriented architecture of our dialogue system is shown in Figure 1 (the Input Manager is left out). The Dialogue Manager consists of a number of *dialogue agents* that are experts on one specific aspect of dialogue management and whose activities are controlled and coordinated by a particualr dialogue controller (which thus currently acts as the central evaluator for all the dialogue agents). The Dialogue Manager decides what to say next on

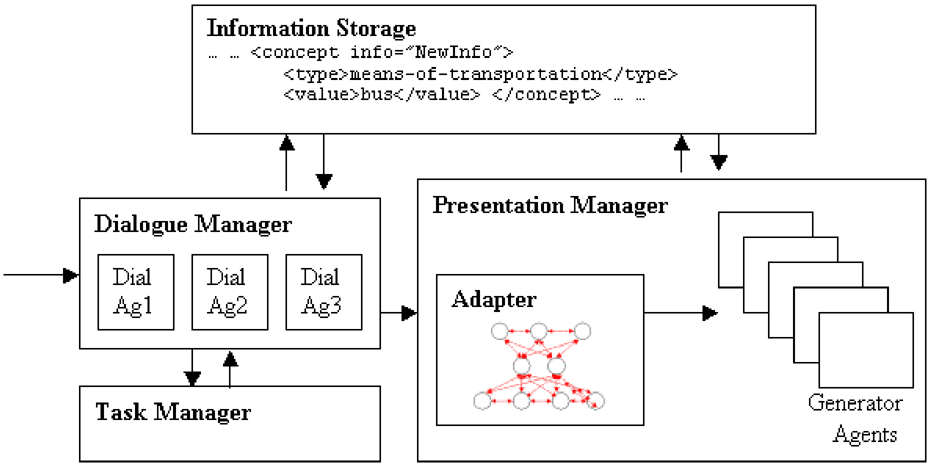

Figure 1: Part of the system architecture

the basis of the dialogue context recorded in the Information Storage by the various agents, and by consulting the Task Manager, whenever the requested information requires factual information about the task itself. The output of this reasoning is expressed in terms of concepts marked as NewInfo and Topic, and stored in the shared Information Storage in the XML format as shown in the examples in Section 2.

The response generation takes place in the *presentation management* module, which contains several *generator agents*, each of which specialized in one particular type of generation. The agents may, for example, generate in different languages (we are developing generators for English and Finnish). In the current implementation, we mainly consider agents for pronominalization, explicitness and politeness.

We are developing generation agents at three distinct explicitness levels. The first agent generates NewInfo only, and is suitable for quick informal interactions with high speech recognition confidence like example (3a). The second agent generates NewInfo wrapped by Topic, where Topic is normally realized as a minimal referring expression such as a pronoun. This agent is suitable for more polite interactions with good speech recognition, as in example (3b). The third agent generates a fully explicit Topic, and is suitable for situations where speech recognition confidence is low, and confirmation of the topic is required, as in example (3c). One advantage of this approach is that the design of the individual generators is simplified, as they can follow a fixed realization strategy, but the overall adaptivity of the system is increased.

The selection of the appropriate generator agent is the task of the component called the Adapter in Figure 1. The Adapter is a particular kind of evaluator based on a neural network implementation. Input consists of a number of features which are encoded as binary features, and output consists of categories that represent the different generators. The features are extracted from the shared information storage and concern e.g. the content of the planned utterance (Topic, NewInfo), recognition confidence of the previous user utterance, and general requirements for cooperative, natural responses.

4.2 Adaptivity

Adaptivity is one of the desirable properties for learning dialogue systems (Jokinen, 2000). It is linked to the system's cooperation with the user, i.e. its capability to provide informative and helpful responses but also its capability to tailor responses according to various situations the users find themselves in.

In the above framework, one approach to providing the desired adaptivity is to have generator agents with different levels of explicitness. Changing levels of confidence in speech recognition accuracy can then lead to selecting generator agents with more or less explicitness.

The detailed mechanisms for switching between these different agents by means of the evaluators in the Jaspis framework, including a soft computing approach based on neural networks, are being evaluated.

Related research has studied adaptivity with respect to system strategies, and identified confirmation as one of the helpful strategies that spoken dialogue systems can use in order to show cooperation and allow the user to correct misrecognized words (Danieli and Gerbino, 1995). The use of system initiative also helps reduce misrecognition errors and thus contributes to user satisfaction (Walker et al., 1998). However, a fixed dialogue strategy may not suit all users, whose knowledge of the system's capabilities may differ. Adaptivity can thus be related to the system's ability to change from a user initiative strategy to a system initiative one, or to use varied confirmation strategies, in response to circumstances and the user model. Empirical evaluation of one such system shows that an adaptable system outperforms a non-adaptable one (Litman and Pan, 2000).

We have widened the notion of adaptivity to concern also the system's generation strategies in maintaining natural interaction with the user. The dialogue manager can be said to select among dialogue strategies, such as confirmation or initiative, and the choice is implicitly shown in the selection of the concepts in the agenda. The presentation manager then selects a generator agent to realise the agenda, and can be said to further extend the system's adaptivity as an aspect of the realization possibilities available for the system. The same agenda can be realised differently (as in examples 3a and 3b) by different generator agents, and thus the system can adapt on different levels. The selection of confirmation or non-confirmation strategy can also depend on the system's other capabilities.

4.3 Confidence

The aim of the adaptivity-oriented architecture is to enable the spoken dialogue system to adapt its responses to the changing confidence levels concerning the system's knowledge of the current dialogue situation. At the start of a new dialogue, when there is no previous history on which to establish any general communicative confidence, the system should start with a highly explicit response generator, and gradually, if some level of confidence is established, switch to a less explicit generator.

The high level of explicitness in the system responses has several aspects. Simply by providing a quantity of words to be recognised and acknowledged, the system can verify its understanding of the relevant concepts. Explicitness thus enables speech recognition confidence to be established, and is related to the system's confirmation strategy as in example (3c).

It is also associated with politeness: a high level of explicitness is more polite, and a high level of elliptical expression is more friendly. Since politeness is expected with strangers, more explicitness is therefore appropriate at the start of a dialogue and less appropriate as the dialogue proceeds: it is thus inversely related to the confidence of the partner which gets established in the shared situation. This pattern of gradual change from a more formal initial register to a more informal register as the dialogue progresses is well known, at least in cultures in which register is not dictated strictly by social hierarchy. Differences between English dialogues (dynamic register adaptivity) and Japanese dialogues (fixed register throughout) have been studied (Kume et al., 1989).

Generation of referring expressions is mainly concerned with enabling successful discrimination of the correct referent. However, referring expressions in dialogue systems are also strongly affected by the level of confidence. When there is some doubt, it is safer to use highly explicit referring expressions. When there is a high level of confidence, it is normal to take certain risks for the sake of fluent interaction. In fact the difference between definite descriptions and pronouns is based on confidence: if an entity has not been mentioned previously, it has no history on which any confidence can be established, so a high level of explicitness is required.

4.4 Communicative Obligations

As argued by Allwood (1976), communication creates normative social obligations which concern the speaker's rational coordinated interaction. Obligations are connected to a particular activity and a role in the activity, varying also according to the speakers' familiarity and relative status with each other.

In dialogue systems, communicative obligations are usually part of the system's control structure. The system can take the initiative, give helpful information in anticipation of the user's questions or to resolve problematic situations (misheard words, ambiguous referents, etc.), or simply react to the user input as best as it can. Obligations are thus used as a basic motivation for action (Traum and Allen, 1994).

In our framework communicative obligations are dispersed among the agents and evaluator control. This allows us to make the obligations overt, since they can be implemented as simple dialogue agents. However, as their application order is not fixed, the overall architecture supports flexible interaction where the basic communicative ability of the system is shown in the functioning of the system itself. The systems's cooperation is not only a pre-assigned disposition to act in a helpful way, but involves reasoning about the appropriate act in the context.

5 Conclusion

We have described our work on adaptivity in response generation for a spoken dialogue system, and have argued in favour of a system architecture using highly specialized agents. The system adapts its responses to dialogue situations by means of the detailed agendas specified by the dialogue manager and the selection of the generator agent by the presentation manager. Further evaluation of a larger demonstrator system is planned.

References

Jens Allwood. 1976. *Linguistic Communication as Action and Cooperation.* Department of Linguistics, University of Göteborg. Gothenburg Monographs in Linguistics 2.

Apache XML Project. 2001. Xalan-Java version 2.1.0. http://xml.apache.org/xalan-j/index.html.

Alison Cawsey. 2000. Presenting tailored resource descriptions: Will XSLT do the job? In *9th International Conference on the World Wide Web.*

Morena Danieli and Elisabetta Gerbino. 1995. Metrics for evaluating dialogue strategies in a spoken language system. In *Proceedings of the AAAI Spring Symposium on Empirical Methods in Discourse Interpretation and Generation*, pages 34–39.

Kristiina Jokinen, Hideki Tanaka, and Akio Yokoo. 1998. Planning dialogue contributions with new information. In *Proceedings of the Ninth International Workshop on Natural Language Generation*, pages 158–167, Niagara-on-the-Lake, Ontario.

Kristiina Jokinen. 2000. Learning dialogue systems. In L. Dybkjaer, editor, *LREC 2000 Workshop: From Spoken Dialogue to Full Natural Interactive Dialogue - Theory, Empirical Analysis and Evaluation*, pages 13–17, Athens.

Masako Kume, Gayle K. Sato, and Kei Yoshimoto. 1989. A descriptive framework for translating speaker's meaning: Towards a dialogue translation system between Japanese and English. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 264–271, Manchester.

Diane Litman and Shimei Pan. 2000. Predicting and adapting to poor speech recognition in a spoken dialogue system. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 722–728, Austin, TX.

Ehud Reiter and Robert Dale. 2000. *Building Natural Language Generation Systems.* Cambridge University Press.

Mariët Theune. 2000. *From Data to Speech: Language Generation in Context.* Ph.D. thesis, Eindhoven University of Technology.

David Traum and James F. Allen. 1994. Discourse obligations in dialogue processing. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Las Cruces.

Markku Turunen and Jaakko Hakulinen. 2000. Jaspis - a framework for multilingual adaptive speech applications. In *Proceedings of 6th International Conference on Spoken Language Processing*, Beijing.

Kees van Deemter, Emiel Krahmer, and Mariët Theune. 1999. Plan-based vs. template-based NLG: A false opposition? In *Proceedings of the KI'99 Workshop: May I Speak Freely?*, pages 1–5, Saarbrücken.

Marilyn Walker, Diane Litman, Candace Kamm, and Alicia Abella. 1998. Evaluating spoken dialogue agents with PARADISE: Two case studies. *Computer Speech and Language*, 12-3.