A Generic Architecture and Dialogue Model for Multimodal Interaction

D. Hofs, R. op den Akker & A. Nijholt
Parlevink Research Group
Centre of Telematics and Information Technology
University of Twente, Enschede, the Netherlands
{infrieks,anijholt}@cs.utwente.nl

Abstract

This paper presents a generic architecture and a dialogue model for multimodal interaction. Architecture and model are transparent and have been used for different task domains. In this paper the emphasis is on their use for the navigation task in a virtual environment. The dialogue model is based on the information state approach and the recognition of dialogue acts. We explain how pairs of backward and forward looking tags and the preference rules of the dialogue act determiner together determine the structure of the dialogues that can be handled by the system. The system action selection mechanism and the problem of reference resolution are discussed in detail.

1 Introduction

To help users find their way in a virtual theatre we developed a navigation agent. In natural language dialogues in Dutch the agent can assist users, when they are looking for the location of an object or room, and he can show them the shortest route between any two locations in the theatre. The speech-based dialogue system allows users to ask questions such as “Where is the coffee bar?” and “How do I get to the great hall?” In addition to that the navigation agent has a map, on which he can mark locations and routes. Users can click on locations in the map and ask questions about them like “What is this?”

We introduce a rather generic architecture and dialogue model allowing speech recognition and multimodal interactions, including reference resolution modelling and subdialogue modelling. Backward and forward looking tags determine the structure of the dialogues.

With different grammars for the speech recogniser and different implementations of the interpreters and clients for dialogue input and output, the same architecture was used for two different dialogue systems. In addition to the navigation application we used the speech architecture for Jacob, a tutor for the Towers of Hanoi, that earlier did not allow speech recognition (Evers et al., 2000).

In section 2 we describe the architecture of the system. In section 3 we describe the dialogue manager and related components in the navigation agent. In section 4 we have a short discussion about the information state approach versus the dialogue state approach in dialogue modelling and we present some conclusions.

2 A Multimodal Dialogue System

We present a multimodal dialogue system that the user can communicate with through speech and mouse input. The system can communicate with the user through speech and images. Initially the architecture can be conceived as a box containing a dialogue manager and world knowledge, which can be connected with input processors and output generators.

The processors and generators can easily be plugged into different dialogue systems. We have shown this by using the same speech processor and generator with some adjustments for both the navigation agent and a tutor.

The current implementation of the system supports streaming at the recording side as well as the playback side to decrease the delay between user utterances and system responses. Together with multithreading this architecture also enables the user to interrupt when the system is speaking. This appeared useful for the tutor, which often gives lengthy explanations that the user heard before. When that happens, the user can say something like, “I already know that.” Then the system stops speaking.

3 Dialogue Manager for the Navigation Agent

Figure 1 shows the architecture of the dialogue manager’s implementation. The input queue and output queue are connected to the dialogue input and output clients in the speech components. The mouse input reaches the dialogue manager through a map, which is a visual 2D representation of the world (the virtual theatre). The user can point at objects or locations on the map and the world notifies the dialogue manager of these events.

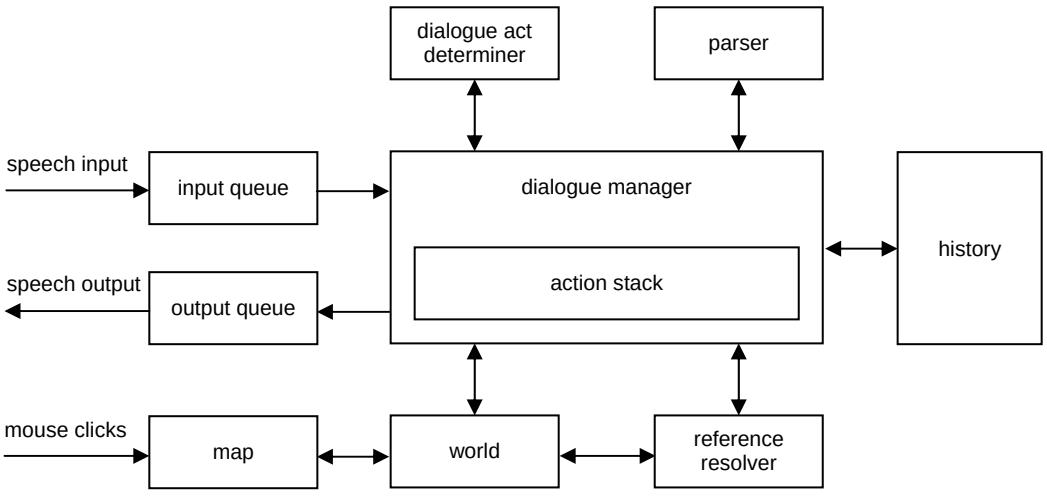

Figure 1: Architecture of the dialogue manager

The various components will be described in the following sections, but first we present the dialogue manager's main algorithm in Figure 2. It runs in a thread as long as the dialogue manager is opened.

```
if (turn == USER)
{
	possibleDlgActs = inputQueue.getDialogueInput();
	dact = dad.selectDialogueAct(possibleDlgActs,history);
	parser.parseParameters(dact);
	resolver.resolveReferences(dact.getArgument());
	actions.createActions(dact);
	history.add(dact);
	turn = SYSTEM;
}
else
{
	act = actions.pop();
	if (act != null)
		act.execute(this);
	else
		turn = USER;
}
```

Figure 2: Dialogue manager's algorithm

This algorithm shows that speech input is sequentially processed by the dialogue act determiner (dad) that selects a dialogue act (dact), the natural language parser (parser), the reference resolver (resolver) and the action stack (actions).

The dialogue management module itself is not implemented as a system of asynchronous distributed agents: there is a strict logical order in the execution of the updates of dialogue information, the selection of goals and the execution of actions after the system has received user input.

This sequential approach has some advantages and some disadvantages if we compare it with more integral approaches such as in (Kerminen & Jokinen, 2003) and (Catizone et al., 2003). In the former approach speech recognition, dialogue manager and speech production each have their own asynchronous processing. In the latter paper the authors mention and discuss the functionalities of a multimodal dialogue and action management module. They have chosen a stack of ATN's, but because during the dialogue the user can return to a previously closed dialogue topic it should be possible to return in the history and a previously popped ATN should remain accessible.

In our dialogue and action management module, to be discussed below, we allow mixed-initiative dialogues – one of the basic minimal functionalities required (Catizone et al., 2003) – and several types of subdialogues. Either the system or the user can take the initiative by asking for clarification instead of directly answering a question. An action stack stores the system's actions that are

planned and a subdialogue stack (the stack of “questions under discussion”) keeps track of the current dialogue structure. Besides, all dialogue acts are kept in a history list of dialogue acts, so they can be retrieved for later use.

Turn-taking is a problem in dialogue modelling. The algorithm shows alternating user/system turns. However, the system may decide to perform two or more dialogue acts or none at all in its actions. In the latter case it is perceived as the user being allowed to perform more than one dialogue act in one turn. The current system assumes that the user only reacts to the last dialogue act performed in a system turn, which need not be its most recent turn. To enable interruptions, the system returns the turn to the user as soon as it has planned a dialogue act and updated the dialogue state as if the act has already been performed. To keep the dialogue state consistent, the speech generator should provide feedback to the dialogue manager about the currently pronounced utterance, so the dialogue manager is able to update its dialogue state appropriately.

3.1 Dialogue acts

The input queue of the dialogue manager receives the user’s utterances in the form of lists of possible dialogue acts. The relation between word sequences and dialogue acts is specified in the grammar for the speech recogniser. We will show how a recognition result is converted into dialogue input with an example. Suppose that the user asks, “Where is the great hall?” The speech recognition grammar has been designed to return:

```
where {/where/location} is {null} the great hall {object}
{FWD_WHQ#<object>./where/} {BWD_NULL} {BWD_ACK} {BWD_HOLD}
```

The recognition interpreter parses this recognition result and returns a list of possible dialogue acts. The recognition result could also be empty, if the speech recogniser failed to recognise an utterance. We will get back to that later. A dialogue act contains the original sentence and a forward tag and backward tag, based on the DAMSL (Dialog Act Markup in Several Layers) scheme (Allen & Core, 1997). In addition to that, a dialogue act may have a domain-specific argument. The example recognition result contains one forward tag (WHQ), three backward tags (NULL, ACK and HOLD) and one argument associated with the WHQ tag (<object>./where/). The set of possible tags assigned to an utterance is motivated by the possible dialogue acts performed by the speaker when he uses this utterance. The sequence /where/ is a variable whose value is specified in the {/where/location} tag. Before parsing the argument, the variable is substituted resulting into <object>.location. The sequence <object> represents a parameter whose value is marked by the {object} tag. The {null} tag serves to mark the start of the parameter value, so the parameter value is “the great hall”. After parsing the argument, it will be clear that the user asked for the “location” property of a parameter of type “object” whose value is “the great hall”. In general an argument, after resolving /xxx/ variables, should be the product of this grammar:

```
EXPRESSION → CONSTANT | PARAMETER | EXPRESSION
CONSTANT → constant_name[.property_name]
PARAMETER → <parameter_name>[.property_name]
FUNCTION → function_name([ARGLIST])
ARGLIST → EXPRESSION | EXPRESSION,ARGLIST
```

To create the list of possible dialogue acts, every single forward tag is paired up with every single backward tag. So there will be three dialogue acts in the list for the example above. Both consist of the forward tag WHQ and the argument <object>.location, but the first dialogue act will have backward tag NULL and the second one will have backward tag ACK. A forward/backward tag pair may also be specified explicitly in the recognition result. This is shown in another example:

```
the great hall {object} {FWD_WHQ_ELLIPSIS#<object>:BWD_ACK}
{FWD_ASSERT#<object>:BWD_ANSWER}
```

In this case there will be two dialogue acts: the WHQ/ACK pair and the ASSERT/ANSWER pair. If the speech recogniser failed to recognise an utterance, the recognition interpreter will create a list with one dialogue act, which has the special forward and backward tag UNKNOWN.

A forward tag is in direct relation with the user's utterance, whereas the backward tag says how the utterance relates to the last utterance in the current subdialogue. The dialogue act determiner uses the backward tag to select one dialogue act from the provided list. The forward and backward tags we currently distinguish are listed in Table 1 and Table 2 respectively.

Forward tag	Description
OPEN	Speaker opens the dialogue.
CLOSE	Speaker closes the dialogue.
THANK	Speaker says thank you.
ASSERT	Speaker asserts something, usually the answer to a question
WHQ	Speaker asks an open question.
YNQ	Speaker asks a yes/no question.
REQ	Speaker requests to do something.
OFFER	Speaker offers to do something.
COMMIT	Speaker promises to do something.
CHECK	Speaker checks information that he thinks is true.
WHQ_ELLIPSIS	Speaker intends to repeat the last open question but only expresses the new content.
UNKNOWN	The speech recogniser failed to recognise an utterance.

Table 1: Forward tags

Backward tag	Description
NULL	Speaker starts a dialogue.
ACK	Speaker understood the last utterance and continues the dialogue.
REPEAT	Speaker repeats (part of) the last utterance.
HOLD	Speaker starts a subdialogue instead of replying to the last utterance.
ANSWER	Speaker answers a question.
ACCEPT	Speaker accepts a request or offer or answers affirmatively to a yes/no question.
MAYBE	Speaker reacts to a request or offer by neither accepting nor rejecting it.
REJECT	Speaker rejects a request or offer or answers negatively to a yes/no question.
UNKNOWN	The speech recogniser failed to recognise an utterance.

Table 2: Backward tags

3.2 Dialogue act determiner

When the dialogue manager gets a list of dialogue acts from the input queue, it passes this list to the dialogue act determiner together with the history. The only information in the history that the dialogue act determiner uses, are the forward tags of the last utterance in the current subdialogue and the last utterance in the enclosing subdialogue, if present. Research by Keizer et al. (2002) on the use of Bayesian networks and other classifiers for classifying user utterances in navigation and information dialogues has shown that there is no obvious reason to use a dialogue history of size greater than one in predicting the current dialogue act. Similar results are reported in (Black et al., 2003). That is, the forward tag of the last dialogue act is indeed a good predictor of the next dialogue act.

For every possible forward tag, the dialogue act determiner holds an ordered list of preferred backward tags that can follow it. For example after a WHQ forward tag (a question), the preferred backward tag is ANSWER. A special backward tag is NULL, which is used for the first utterance in the dialogue. The dialogue act determiner selects the preferred dialogue act and returns it to the dialogue manager.

The dialogue act determiner also determines the dialogue structure: a user dialogue act can be a continuation of the current subdialogue as well as a continuation of the enclosing dialogue. If the user could end the current subdialogue – that is if the last dialogue act in the enclosing dialogue was performed by the system and the user started the current subdialogue – the dialogue act determiner will always try to end the current subdialogue by connecting the user's dialogue act to the enclosing dialogue. So it first considers

the forward tag of the last dialogue act in the enclosing dialogue. Only if none of the preferred backward tags for this forward tag, is available, the forward tag of the last dialogue act in the current subdialogue is considered. If still none of the preferred backward tags is available, the dialogue act determiner returns the first dialogue act in the input list. Section 3.6 will illustrate this with an example.

3.3 Reference resolver

The dialogue manager can start processing the selected dialogue act. It starts with parsing the phrases that occur in parameters in the dialogue act's argument. The parser returns a feature structure, which is stored together with the original parameters in the dialogue act.

The next step is to bind the parameter to a real object in the navigation agent's world. That is where the reference resolver comes in. The reference resolver is used for all references to objects in the world. References can be made by the user or the system, by talking about objects or by pointing at them. An example dialogue should illustrate this.

U1: Is *this the cloak room*? (*pointing at the coffee bar*)
S1a: No, *that*'s *the coffee bar*.
S1b: *The cloak room* is over there.
U2: Could you take me *there*?

Experience teaches that usually users' pointing acts precede the accompanying verbal action (Oviatt, 1999). In that case the first reference to the coffee bar is made by the user pointing at it. This evokes a new referent into the reference resolver's dialogue model. All references are bound to their referent in the dialogue model. In this example the first reference bound to the coffee bar is the user's pointing action. Then the user uses "this" to refer to the coffee bar, which should be determined by the reference resolver. Because "this" is a demonstrative, it will try to find a referent among earlier referents. The referring expression "the cloak room" however can be resolved by looking at the world. It evokes another referent. Then the system refers to the coffee bar twice and in the next utterance it refers to the cloak room. Finally the reference resolver should be able to determine that "there" refers to the cloak room.

The reference resolution algorithm is a modified version of Lappin and Leass's algorithm (Lappin & Leass, 1994) that assigns weights to references, based on a set of salience factors. Although language used in the dialogue system is rather simple, compared to complex structures in written texts, resolving referring expressions in multimodal interaction is far from trivial. The modification makes the algorithm suitable for multimodal dialogues.

The algorithm has two tasks: resolving references and updating the dialogue state. The dialogue state contains a list of referents. Each referent is a collection of objects in the world. All references in the dialogue are bound to a referent and they are grouped by utterances. A reference consists of a feature structure (for

verbal references as opposed to pointing references) and a set of salience factors that apply to the reference. The salience factors define certain preferences. Each factor has a fixed weight value. They are listed in Table 3.

Salience factor	Value
1. recency	100
2. point	80
3. head noun	50
4. no prepositional phrase	20

Table 3: Salience factors and values

A salience value is the sum of the values associated with a set of salience factors. At each new utterance, all salience values are cut in half, so more recent references get a relatively high salience value. The recency factor is always assigned to make sure that a salience value is always positive and it decreases when it is cut in half. The point factor is assigned if a reference is made by pointing at an object. The head noun factor is assigned if a reference is the head noun in a phrase. And the last factor is assigned if the reference did not occur in a prepositional phrase.

The references that occurred within one utterance share one salience value. This value is computed from the salience factors that are assigned to any reference in that utterance. The salience value of a referent is the sum of the salience values of all utterances. The dialogue model also stores who made an utterance: the user or the system.

In addition to the *preferences* defined by the salience factors, the algorithm can apply *constraints* on referents. There are syntactic constraints related to the feature structures of referring expressions. For example a plural expression cannot refer to a referent that has only been referred to with singular expressions. Semantic constraints are used for mutually exclusive referring expressions. An example is the pair “here” and “there” in the utterance, “How do I get from here to there?” In this utterance the referring expressions “here” and “there” cannot have the same referent. We currently use the constraint that any two demonstratives used within one utterance, cannot have the same referent. This is obviously too strict.

The dialogue model is used to resolve (verbal) references. Verbal references are represented by a feature structure and two kinds of references can be distinguished: those that only consist of a demonstrative or pronoun (like “this”, “there” or “it”) and those that contain a content word (a noun or adjective). If there is only a demonstrative or pronoun, the reference resolver will ignore the world and only look at earlier referents. First it will create a set of candidates by checking syntactic constraints on earlier referents, based on the feature structures of the referring expressions. Semantic constraints may also be applied and then

the reference resolver will select the referent with the highest salience value. If there are more referents with the same salience value, the referent most recently referred to is selected. If no referent is found, the reference resolver will return an empty set of objects.

To resolve references with a content word, the reference resolver will first make use of the world. This will result into a set of objects. If there was a definite article or a demonstrative, the reference resolver will subsequently consider earlier referents. The candidates are restricted to referents that are a subset of the set returned by the world. If no referents were found, the original set is returned.

We will show how that works with the example dialogue, which is repeated here:

U1: Is *this the cloak room*? (*pointing at the coffee bar*)
S1a: No, *that*'s *the coffee bar*.
S1b: *The cloak room* is over there.
U2: Could you take me *there*?

First the user makes a reference to the coffee bar by pointing at it. The assigned factors are 1 and 2 for recency and pointing (see Table 3) and the salience value is 100 + 80 = 180. The dialogue model consists of one referent with one reference bound to it.

Then the user makes two verbal references: "this" and "the cloak room". The first reference consists of a demonstrative and therefore the reference resolver considers earlier referents. There is only one candidate: the coffee bar. The salience factors 3 and 4 are added, so the salience value becomes 250. The second reference contains a content word, so the world is used to find the cloak room object. Because of the definite article, earlier referents are considered too, but there is no referent that is a subset of the singleton set containing the cloak room. So a new referent is introduced for the cloak room. The dialogue model now contains two referents. The coffee bar has two references and its salience value is 250. The cloak room has one reference and salience value 170.

A new utterance is started, so the salience values are now cut in half. The system makes two references: "that" and "the coffee bar". Now the coffee bar has four references in two utterances. The user's utterance has salience value 125 and the system's utterance has salience value 170, so the referent's salience value is 295. The cloak room's salience value is 85 (half of 170).

The salience values are cut in half again and in the new utterance the system refers to the cloak room. The updated dialogue model is shown in Figure 3.

Note that the coffee bar's salience value is 147.5 and the cloak room's is 212.5. When the user's next reference "there" needs to be resolved, it will result into the cloak room, because it has the highest salience value.

After resolving references, the set of objects found is added to the parameter in the dialogue act where the reference occurred. So now we have a dialogue act that consists of a forward tag, a backward tag and an argument, and the

parameters in the argument have a value that was taken directly from the recognition result as well as a feature structure received from the parser and a set of objects received from the reference resolver.

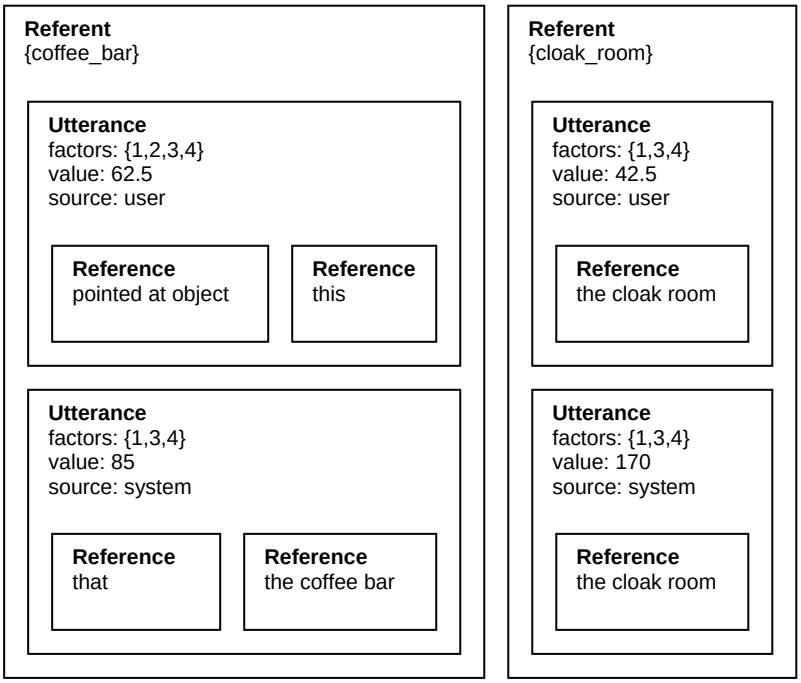

Figure 3: Reference resolver's dialogue model

3.4 Action stack

The history contains all dialogue acts that occurred during the dialogue, including the system's dialogue acts. They have a forward tag and a backward tag, but they differ from the user's dialogue acts in that they contain an action instead of an argument. A subdialogue stack is used for the currently running subdialogues. The height of the stack determines the subdialogue level. On top of the stack is the current subdialogue.

The action stack contains the actions that the system still needs to execute, but the stack also creates those actions, when it is provided with the user's dialogue acts after the parameters have been parsed and references were resolved. The rules that tell which actions should be created, are specified in a simple text file. An example line from that text file is:

```
WHQ  <object>[1].location  TELL_LOCATION(1)
```

This rule tells that when the user asks for the location of an object, the system should plan to tell the location of that object. More precisely, the user's dialogue act should have a WHQ forward tag and its argument should be <object>.location. In that case the action stack should create a TELL_LOCATION action with one

argument, which is the parameter marked with 1. A rule may also specify a list of actions separated by commas. Note the difference between a dialogue act and an action. The action name TELL_LOCATION suggests that it is a dialogue act, but it is not. A dialogue act may be performed from within an action however. In fact the TELL_LOCATION action is merely an intention to tell the location of an object, i.e. to perform the particular dialogue act. In certain dialogue states the system might not be able to tell the location of an object and the action could produce a different dialogue act.

The action rules from the text file are compiled into action templates and they are part of the action stack. The action that is specified in an action template, should be mapped to an action class that has an execute method. This way the system can easily be extended with new actions, by adding a rule to the text file, creating an action class with an execute method, and mapping the action name in the template to the new action class. Although the system's actions seem fixed by the user's dialogue acts, the actions are quite flexible as they have access to the dialogue manager and the world. The execute methods can be implemented to act differently depending on the state of the world and the dialogue. We will illustrate this with a few examples later.

When the action stack receives a dialogue act from the user, it will try to find an action template with matching forward tag and argument. Then it will create actions based on the action names specified in the template and it will extract the action arguments from the argument in the user's dialogue act. The new actions are put on top of the stack and when it is the system's turn, it will take an action from the stack and execute it. If a dialogue act is performed within the action, the reference resolver's dialogue model should be updated and the dialogue act should be added to the history. Like a user dialogue act, a system dialogue act consists of a forward tag and a backward tag, but a system dialogue act does not have an argument. Instead it contains the action within which the dialogue act was performed.

Now we will look at an example. Consider the following dialogue:

U1: Where is the cloak room?
S1: Which cloak room are you looking for?
U2: Are there more cloak rooms?
S2: Yes, there's one upstairs and one downstairs.
U3: I meant the cloak room downstairs.
S3: It is over there.

The first user utterance is a dialogue act with forward tag WHQ, backward tag NULL and argument <object>.location, where the object parameter is "the cloak room". This parameter is parsed and the reference resolver binds it with the two cloak room objects in the building. The information is stored in the dialogue act, which can now be passed to the action stack. The action stack will find the template we mentioned before:

```
WHQ  <object>[1].location  TELL_LOCATION(1)
```

The TELL_LOCATION action is created and it has one argument, which is the set with both cloak rooms. If the system cannot tell the location of two objects, it could start a subdialogue by asking: “Which cloak room are you looking for?” This is a dialogue act with forward tag WHQ and backward tag HOLD. The dialogue model of the reference resolver is updated in accordance with the reference to both cloak rooms and the dialogue act is added to the history. Because a subdialogue was started, the history’s subdialogue stack now contains two subdialogues. The current subdialogue (containing S1) is on top of the stack and the main dialogue (containing U1) is at the bottom. Since the system could not really tell a location, the TELL_LOCATION action is put on the action stack again, but this time a flag is set that says that the action is not yet executable.

Then the user starts another subdialogue and asks, “Are there more cloak rooms?” This is a dialogue act with forward tag YNQ, backward tag HOLD and argument exist(<object>), where the object parameter is “more cloak rooms”. Again the parameter is bound to both cloak rooms. The action stack could find this template:

```
YNQ  exist(<object>[1])  TELL_IF_OBJECT_EXISTS(1)
```

The action is created and put on the stack, so the action stack now contains two actions: TELL_IF_OBJECT_EXISTS and TELL_LOCATION. The history has three subdialogues on its own stack now. The new subdialogue with U2 was pushed on top of the stack. The TELL_IF_OBJECT_EXISTS action is executed. The system says, “Yes, there’s one upstairs and one downstairs.” This is a dialogue act with forward tag ASSERT and backward tag ACCEPT. The TELL_LOCATION action is still on the action stack.

Finally the user chooses the cloak room downstairs. The dialogue act determiner tries to find a dialogue act that ends the current subdialogue and it succeeds when it determines that the user’s utterance is a dialogue act with forward tag ASSERT, backward tag ANSWER and argument <object>. Note that the backward tag refers to S1, the last dialogue act in the enclosing subdialogue. The subdialogue U2-S2 is taken off the stack and on top of the stack is now the subdialogue S1-U3. The object parameter is now “the cloak room downstairs”, which the reference resolver will bind to the correct object. The action stack will find this template:

```
ASSERT  <object>[1]  FILL_OBJECT(1)
```

The FILL_OBJECT action tries to substitute a parameter of type “object” in the action on top of the action stack. In our example this is the TELL_LOCATION action. With the parameter substitution the action is also made executable again,

so the dialogue manager can take it from the stack and execute it. Because its parameter is now only one object, the system can tell the location of it. The subdialogue is also ended, so the subdialogue stack now only contains the main dialogue. In a similar way this model can be used to resolve ellipsis.

4 Conclusion and Future Research

The dialogue model presented is based on dialogue acts and the information state approach to dialogue modelling. The current implementation has no dynamic planning mechanism, but static plan rules.

As such it is comparable with GoDiS, the Gotenburg dialogue system that is proposed as an implementation of the TRINDI architecture (Bohlin et al., 1999).

The information state approach is sometimes opposed to the dialogue state approach. However, notice that the specification of the dialogue acts by the recognition grammar in terms of pairs of backward and forward looking functions together with the rules of preferences used by the dialogue act determiner for deciding whether a user act should be understood as a subdialogue closing act or open act implicitly pose a grammatical structure on the set of possible dialogues the system can handle.

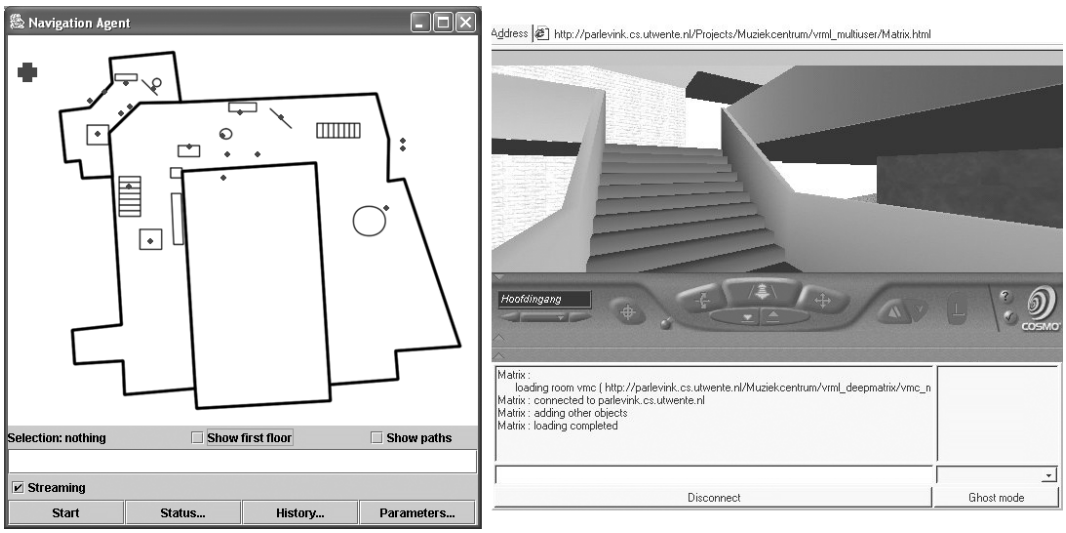

Figure 4: 2D navigation map and 3D (multi-user) virtual environment

We presented a dialogue model for spoken multimodal interaction and explained dialogue act determination, action handling and reference resolution using examples from a navigation dialogue system that interacts with the user through speech and a 2D map. In the complete system the user has two coordinated views of the situation: a 2D map and a view in the virtual environment (see Figure 4). Obviously, a next step is to extend our multimodal dialogue system to include references, pointing actions and subdialogues that are more directly linked to the

virtual world. We have not done so yet in order not to complicate and obscure the design of our dialogue system. That is, we assume that the current design allows us extensions to the more complicated situation.

Moreover in the current system generation of system utterances is done in a rather primitive way. A more principled approach to multimodal generation is investigated.

Acknowledgement

We are grateful to the referees for their critical remarks that helped us to improve this paper.

5 References

Allen, J. & M. Core, 1997. *Draft of DAMSL: Dialog Act Markup in Several Layers*. University of Rochester.

Black, W., P. Thompson, A. Funk & A. Conroy, 2003. Learning to classify utterances in a task-oriented dialogue. Proc. Workshop on Dialogue Systems: Interaction, Adaptation and Styles of Management. 10th Conf. of the EACL 2003, Budapest.

Bohlin, P., R. Cooper, E. Engdahl & S. Larsson, 1999. Information states and dialogue move engines, in: J. Alexandersson (ed.) *IJCAI-99 Workshop on knowledge and reasoning in practical dialogue systems*.

Catizone, R., A. Setzer & Y. Wilks, 2003. Multimodal Dialogue Management in the COMIC project. Proc. Workshop on Dialogue Systems: Interaction, Adaptation and Styles of Management. 10th Conf. of the EACL 2003, Budapest.

Evers, M. & A. Nijholt, 2000. Jacob - an animated instruction agent for virtual reality, in: T. Tan et al. (eds.) *Advances in Multimodal Interfaces - ICMI 2000*, LNCS 1948, Springer, Berlin, 526-533.

Keizer, S., R. op den Akker & A. Nijholt, July 2002. Dialogue Act Recognition with Bayesian Networks for Dutch Dialogues, in: K. Jokinen & S. McRoy (eds.) *3rd SIGdial Workshop on Discourse and Dialogue*, Philadelphia, Pennsylvania, 88-94.

Kerminen, A. & K. Jokinen, 2003. Distributed dialogue management in a blackboard architecture. Proc. Workshop on Dialogue Systems: Interaction, Adaptation and Styles of Management. 10th Conf. of the EACL 2003, Budapest.

Lappin, S. & H. Leass, 1994. An algorithm for pronominal anaphora resolution. *Computational Linguistics*, 20(4):535-561.

Nijholt, A., J. Zwiers & B. van Dijk, 2003. Maps, agents and dialogue for exploring a virtual world, chapter in: J. Aguilar, N. Callaos & E.L. Leiss (eds.) *Web Computing*, International Institute of Informatics and Systemics (IIIS), to appear.

Oviatt, S. Ten myths of multimodal interaction. *Comm. ACM*, 42(11):74-81, 1999.