Dialogs with BDP Agents in Virtual Environments

Arjan Egges and **Anton Nijholt** and **Rieks op den Akker**
Department of Computer Science
University of Twente
PO Box 217, 7500 AE Enschede
the Netherlands
Phone: (31) 53 4893686, Fax: (31) 53 4893503
Email: {egges,anijholt,infrieks}@cs.utwente.nl

Abstract

In this paper we discuss a multi-modal agent platform, based on the BDI paradigm. We now have environments where more than one agent is available and where there is a need for uniformity in agent design and agent interaction. This requires a more general and uniform approach to developing agent-oriented virtual environments that allow multi-modal interaction. We first focus on a formal model for conversational planning agents, for which we discuss the specification of beliefs, desires and conditional plans. Then we discuss the relation between natural language and the communication as it is defined for these agents. We also show how referential problems are treated in the process of interpreting the informational content in a dialog situation. The general agent framework will be used in the specification of agents in a virtual environment, that can interact in a multi-modal way with others.

1 Introduction

In this paper we discuss our research on an agent platform, based on the BDI paradigm, which we plan to use in multi-user, multi-agent virtual environments where interaction between users and agents can take place in multi-modal ways.

We have built several virtual environments where different agents and different interactions between visitor and one or more agents can occur. Our main environment is the virtual music theatre, a VRML built world where visitors can explore a theatre environment and can interact with an embodied agent that knows about performances, performers and available tickets. This agent can be accessed using natural language dialog—using the keyboard—and the agent itself uses text-to-speech synthesis with corresponding visemes in the face [Nijholt and Hulstijn, 2000]. In the course of years several variants and extensions have been made available. One version included a speech accessible navigation agent, another version allowed multiple users to visit this environment as embodied avatars and to chat with each other. In [Nijholt and Hondorp, 2000] we discussed a situation where we have agents in a virtual environment, where these agents may represent human visitors or domain-defined agents with particular knowledge, where all of these agents may have different capabilities and 'physical' (in a virtual world) appearances and where these agents are able to communicate with each other. Depending on the agents and on the application, the communication should allow verbal and nonverbal interaction and also show the effects of the communication on the visualized environment. That is, an agent is commanded to perform a certain task in the environment or takes the decision to perform a particular task leading to changes in the environment. Obviously, in the communication between agents (including the human visitors) references will be made to the environment that is visible for the agents. In order to have a smooth and natural communication between agents that represent humans and the artificial agents it is necessary to take into account communication issues that deal with believability, trust and affect [Nijholt, 2001].

All these issues will be attacked in our AVEIRO (Agents in Virtual Environments) project. This project aims at developing a framework for agent behavior and agent communication in visualized (virtual, 3D) worlds, where there is multi-modal communication between agents and where agents may have different capabilities, including the capability to represent humans, (part of) their properties and (part of) their behavior, including the possibility to capture this behavior using head-trackers, eye-trackers, data gloves and haptic devices and motion capturing technology. However, in order to do so we need to develop frameworks for agents and agent communication. We already introduced a framework that allows different agents to send messages to each other. It has been used in the implementation of a navigation agent interacting with a visitor and some software agents [van Luin *et al.*, 2001] and in the design and implementation of a personal assistant [Zwiers *et al.*, 2000] that can make suggestions to a visitor of our virtual theatre environment.

This paper describes the framework for introducing and defining agents in our environment that allow communication and acting at the level of the meaning of the utterances these agents exchange. Sections 2–4 present the general framework that we have developed. Section 5 shows how inter-agent communication is established in this framework. Then, in Section 6, we will present one kind of interaction that is possible with the agent architecture: interaction through dialog. Section 7 extends this notion by showing how our agents can operate in a multimodal environment with e.g. mouse point-

ers, speech recognition and a virtual body.

2 The BDP architecture

A BDP (Belief, Desire, Plan) agent is completely specified by defining it's:

1. name
2. initial belief state
3. conditional plans

The name of the agent is necessary to give it an identity in a multi-agent framework, see Section 5. The initial belief state of the agent consists of Beliefs and Desires. The state of an agent (when running) consists of two parts: a list of beliefs and desires (also called the *belief state* or BS) and an *event buffer*. In our agent definition, the events in the buffer refer to *messages* from other agents. When the agent is started, its BS will be equal to its initial belief state.

The agent knows a set of *conditional plans* (or CPs) that can change its state. These CPs contain a list of parameters, a list of conditions and a sequence of atomic actions that represents the plan. Next to these notions, every CP has a priority. The agent knows three kinds of atomic actions:

1. add a belief/desire to its state
2. remove a belief/desire from its state
3. send a message[1]

These actions do not ensure the consistency of the agent's state. For example, an action could be: add the belief 'book1 is in box1' to the state, while the state contains the belief 'book1 is not in box1'. The developer should ensure the consistency by properly defining the agent and its CPs.

We will give a small example of a BDP agent. Consider an agent that can fetch mail from a mailbox. Suppose that the agent knows that there is mail in the mailbox. This agent would have a desire to fetch the mail and it would have some CPs, like 'walk to the mailbox' and 'take mail from mailbox'. The CP 'take mail from mailbox' has certain conditions: the agent has to be near the mailbox, the mailbox has to be open and there has to be mail in the mailbox. The CP 'walk to the mailbox' has as a condition that the agent is not near the mailbox. A planning system could then try to find a list of CPs that puts the agent in a desirable state, for example: walk to the mailbox, open the mailbox, take mail from the mailbox, close the mailbox and walk back.

We can specify this agent in TASL[2], an agent specification language for defining BDP agents. A part of this definition can be given in TASL as follows[3]:

```
agent mailAgent {

   init {
      I believe I am inside the house;
      I believe there is mail in the mailbox;
      I desire to fetch the mail;
   }

   cp takeMailFromMailbox {

      priority {
         0;
      }

      parameters {
      }

      conditions {
         I believe I am near the mailbox;
         I believe the mailbox is open;
         I believe there is mail in
            the mailbox;
      }

      plan {
         take the mail from the mailbox;
         remove the belief that there is
            mail in the mailbox;
         add the belief that the agent
            is holding the mail;
      }

   }

   ...
   ...
   ...
}
```

To represent beliefs, desires, predicates and other kinds of logical forms, we use the resolved quasi logical form (QLF) as it is presented in [Alshawi, 1992]. QLFs are written in Prolog list style. We have extended these QLFs with the modal **B** (belief) and **D** (desire) operators. To express the belief that there is a man that loves Mary, we write:

```
[B,me,quant(exists,M,[man,M],[loves,M,mary])]
```

In a similar way, we can express that John desires not to be a fireman :

```
[D,john,[not,[fireman,john]]]
```

Furthermore, we have extended the QLF formalism with two *pointers* that refer to the contents and sender of a message in the event buffer: ***sender** and ***contents**. In our current implementation, the event buffer has a size 1, so these pointers are `null` if the buffer is empty. If the buffer contains a message, they refer to the sender and contents of this message.

These QLFs, combined with TASL provide a means to specify agents. To conclude, we give one example of a complete specification of an agent: the Hanoi agent.

Suppose we have three sticks. On the first stick are three discs: a large disc, a medium disc and a small disc. The goal is that all three sticks are on the last stick. However, only one disc at a time can be moved and it is forbidden to place a disc on another disc that is larger than itself.

Assume that we want to define an agent that solves this problem, we will call it the Hanoi agent. The agent knows three sticks: stick 1, stick 2 and stick 3. Also, it knows three

[1] Section 5 deals with this issue more thoroughly.

[2] Twente Agent Specification Language

[3] In the following examples, the event buffer is not used, because the agents do not need any interaction to operate. Also, the priority of every CP is not yet of importance.

discs: disc 1, disc 2, and disc 3. The agent knows that all three discs are on stick 1. Furthermore, the agent knows that disc 3 is larger than disc 2 and that disc 2 is larger than disc 1. The agent has the desire that all the discs are on stick 3. In this way we have defined the *initial belief state* of this agent. The agent knows only one CP: move a disc `D` from a stick `S1` to another stick `S2`. The conditions of the CP will be defined in such a way that discs can only be moved according to the rules of the game. The plan of the CP contains two actions: remove the belief that `D` is on `S1` and add the belief that `D` is on `S2`. The complete agent specification is given as follows:

```
agent hanoiAgent {

   init {
      [disc,disc1];
      [disc,disc2];
      [disc,disc3];
      [stick,stick1];
      [stick,stick2];
      [stick,stick3];
      [on,stick1,disc1];
      [on,stick1,disc2];
      [on,stick1,disc3];
      [larger,disc3,disc2];
      [larger,disc2,disc1];
      [larger,disc3,disc1];
      [D,me,quant(forall,D,[disc,D],
          [on,stick3,D])];
   }

   cp move_D_from_S1_to_S2 {

      priority {
         0;
      }

      parameters {
         D : [disc,D];
         S1 : [stick,S1];
         S2 : [stick,S2];
      }

      conditions {
         [on,S1,D];
         quant(forall,Dx,
               [and,[disc,Dx],[on,S1,Dx]],
               [not,[larger,D,Dx]]);
         quant(forall,Dx,
               [and,[disc,Dx],[on,S2,Dx]],
               [larger,Dx,D]);
      }

      plan {
         [remove,[on,S1,D]];
         [add,[on,S2,D]];
      }

   }

}
```

3 The BDP interpreter

In this section, we present the BDP interpreter. The first subsection contains a general description of this interpreter. The second subsection focuses on how the BDP interpreter chooses CPs to apply.

3.1 General structure of the BDP interpreter

The BDP interpreter always knows what the state of the agent is and what its CPs are. The interpreter is roughly based on the abstract BDI interpreter as it is discussed in [Weiss, 1999] The following program generally describes this interpreter:

```
initialiseState(agent);
while (!allDesiresReached(agent)) {
   eventBuffer
      = getExternalEventbuffer(agent);
   options = generatePossibleCPs(
                  eventBuffer,agent);
   selectedCP = selectProperCP(
                  options,agent);
   applyCP(selectedCP,agent);
   removeReachedDesires(agent);
}
```

First, a BDP agent has to observe its state. Are there still desires that can be satisfied? If this is the case, the agent has to generate a list of CPs that it can apply in its current state. This list is generated based on the information that the agent has. The agent does not only have to deal with internal information (its state), but with external information as well. The agent has to regard its beliefs and the event buffer to construct a list of optional CPs. Given a set of substitutions for its parameters, a CP is optional if its conditions hold after applying the formerly mentioned substitutions. Our BDP interpreter assumes a *closed world*. Statements that are not in its BS are regarded to be *false*. For checking the conditions, our current implementation does not (yet) use any inference mechanisms, because for most simple cases, they are not necessary and furthermore: using an inference machine is time-consuming. So finally, this part of the BDP interpreter will deliver a list of CPs with a list of possible substitutions for every CP.

Then the agent has to choose a proper CP, that is, a CP that serves its desires (how this is determined, we will discuss later). If a proper CP is found, it can be executed. This delivers a new state. All desires that are reached can be removed from the new state of the agent, and the agent can again see if there are any CPs that it can apply.

3.2 Selection of a proper CP

Giving the current state of the agent and a list of CPs, the agent can calculate how it can reach a desirable state. A desirable state is defined as a state where one ore more desires are achieved. For searching within the state graph of the agent, we use the uniform cost search algorithm (as discussed in [Norvig and Russell, 1995]). The interpreter generates possible states during the search process, so that not the whole state graph of the agent has to be constructed in order to find a desirable state. In our implementation, the BDP interpreter selects the first desirable state that it finds as objective and it selects CPs to execute in order to reach this desirable state.

The agent can thus generate a path to a desirable state. But when the agent is in the desired state, this does not yet mean that all of its desires are achieved (because a desired state

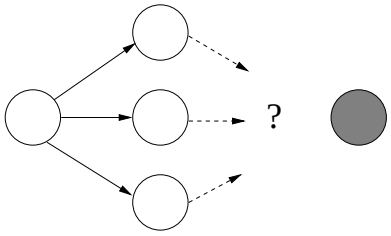

Figure 1: A state graph with an initial state, three intermediate states and a desired state (grey).

is a state where *one or more* desires are achieved). If there are still desires to be achieved in the desired state, the agent again has to calculate a path to the next desired state, until all desires are reached. This is also a way to determine if some desires are unreachable. The agent will never find a state in which these desires are achieved, so it will first reach all other desires. Then the shortest path algorithm will not find a path to a desired state anymore, and the remaining—unreachable—desires can be abandoned.

However, often, a desire can only be achieved by performing interaction. How can an agent determine a path to a desired state, if this state can only be reached by processing input events? Surely an agent can impossibly check the conditions of a CP for all possible input values. The agent just doesn't know what happens, which brings non-determinism in the system. Now, the priorities of CPs are handy. Suppose we have a state graph of an agent with an initial state, a desired state and several CPs (some of them dependent on input) that lead to other states. Suppose the agent can reach three different states from the initial state. In every one of these states, only CPs that depend on input can be applied. Figure 1 shows such a graph.

In this figure, the agent starts, but can't find a path to a desired state. But perhaps, going to one of the three intermediate states results in finding a path to the goal state. When the agent is in the initial state, it does not know to which state it should go. Perhaps the first state would lead directly to the goal state after a certain input, but then again, if this input does not occur, the agent can wait forever. Just doing nothing and staying in the initial state is not a good idea as well. This could lead to a situation in which an agent chooses to do nothing forever, because every input might lead to an non-deterministic state again. We want this agent to be *social*, that is: the agent should be willing to communicate with others to achieve its desires.

So in some way the agent has to choose between the three intermediate states. Of course, the agent can calculate the length of the path to every one of the three states. We know that the agent prefers a short path to a long path. Choosing the right intermediate state then boils down to choosing the path that is the shortest. Now it all comes down to a proper and complete agent specification, with CPs that are applicable in a lot of states so that the agent can constantly choose between different CPs.

As soon as the agent has reached a state in which only CPs that concern input are applicable, it will have to wait until one of these CPs can be executed (thus, that the agent receives input of some kind). Then it can execute this CP. This brings the agent into a new state, from which it can again search for a path to a desired state, and so on.

3.3 Operational semantics of TASL and QLF

[Sadek *et al.*, 1997] describes ARTIMIS, a generic communicative agent. They present a logical framework, formalised in a first-order modal language. ARTIMIS is based on the BDI model: it has a logic of belief and also a logic of intention and action. Also, it has axioms that say something about the behavior of e.g. the belief operator. Since our system doesn't (yet) incorporate any kind of logical inference, we have not defined any axioms on which logical inference depends. However, the uniform cost search algorithm that we apply, can—in a sense—be seen as a constructive proof machine, since it constructs a plan to infer a statement (a desire) from a given set of beliefs.

4 Plans, desires, intentions and commitment

According to [Weiss, 1999], a BDI architecture is a type of agent architecture that contains explicit representations of beliefs, desires and intentions. Beliefs are the information an agent has about its environment, which may be false; desires are those things that the agent would like to see achieved, and intentions are those things the agent is either committed to doing (intending *to*) or committed to bringing about (intending *that*).

The concepts 'belief' and 'desire' are very clear, while the concept 'intention' gives rise to many discussions. [Cohen and Levesque, 1990] gives a description of what intentions are, according to their philosophy. In this case, intentions are modelled as *persistent goals*. "Intention will be modelled as a composite concept specifying what the agent has chosen and how the agent is committed to that choice" [Cohen and Levesque, 1990]. If intentions are defined by a choice and a commitment to this choice, development of a formal agent platform will be difficult. How should we define these 'commitments' and how are choices related to these commitments? The definition of [Weiss, 1999] mentions two categories of intentions, present-directed and future-directed intentions. [Wooldridge, 2000] defines intentions as desires that an agent has committed to achieving, and a desire corresponds to a plan.

In this research, we want to concentrate on dialog systems that use agent technology. The agent model that is to be used, should be clearly defined and non-ambiguous. Therefore, we leave the concept of intention as it is and we use a part of the BDI model: the beliefs and the desires. In stead of intentions, we use plans that the agent can execute. However, abandoning the concept of intentions does *not* mean that we find intentions are useless. On the contrary, Cohen and Levesque discuss intentions and show disadvantages of systems that replace these intentions by plans: "... although there surely is a strong relationship between plans and intentions, agents may form plans that they never 'adopt' and thus the notion of a plan lacks the characteristic commitment to action inherent in our commonsense understanding of intention." [Cohen and

Levesque, 1990]. We know that intentions can be a very useful notion, but for now we will start with a more simple agent definition so that the basis of the BDP agent comes forward in a clearer way.

5 Communication between agents

Agents communicate by means of sending messages containing the name of the sender, the name of the agent addressed, a time-stamp, and a content, to a central Communicator [Zwiers *et al.*, 2000]. Agents entering the platform subscribe themselves to this Communicator. The only function of the Communicator is to forward messages it receives to the agents given by the adress (if the adressed agent is known at the Communicator). The Communicator allows agents to run on whatever what location or machine.

The content of a message contains a conversational act type and a quasi logical form. Conversational act types can denote for example requests, wh-questions, yes/no questions or declarations. (this is to compare with KQML or ACL). The request to put a book in a box—for example—is encoded as follows:

```
[imp,[inBox,box2,bookA]]
```

The wh-question "which books are in box2?" is encoded as:

```
[whq,quant(wh,X,[book,X],[inBox,box2,X])]
```

How does an agent know to what other agent he should address his questions? To represent that agent A knows that agent B has information about theatre performances the BS of A contains:

```
[knows_about, B, performance]
```

If agent A receives a question like

```
[whq,quant(wh,X,[name,X],
        quant(exists,Y,[performance,Y],
              [title,Y,X]))]
```

("show me the titles of the performances") from agent C and agent A does not have this information in his own BS, agent A will forward the query to the agent B and after evaluating the answer received from B he can return an answer to agent C.

6 Natural language interaction

A special form of interaction is natural language interaction. Given the structure of agents that communicate through sending and receiving messages, we will now discuss how such a structure can be used to construct a dialog agent (for a general structure of this agent, see Figure 2).

6.1 The NL parser

Natural language input is first analysed by a left-corner bottom-up chart parser for unification grammars that outputs a typed feature structure containing a (partial) specification of the type of utterance (imperative, declarative, wh- or y/n-question, np or pp) and and a semantic specification containing the functional parts of the utterances (predicate, subject, object, and modifiers). This semantical representation is tranformed into a quasi logical form, that is sent as the contents of a message to the agent. The grammar and lexicon formalism used is similar to PATRII. The types of feature structures

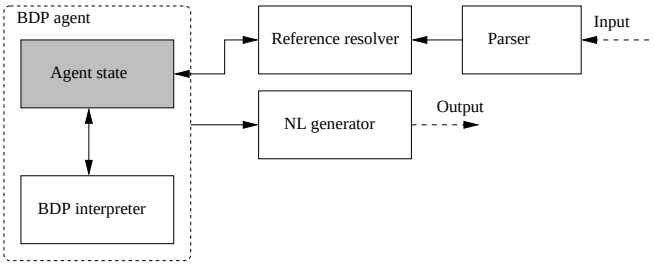

Figure 2: Structure of a dialog agent. The BDP agent part consists of a state and an interpreter module. A dialog agent contains also a parser, reference resolver module and a natural language generator.

are given by a special type feature. A separate type specification contains the type hierarchy (which should be bounded complete in order to have uniqueness of type unification) and the features with their types that belong to the types (this is comparable with the type specification of ALE). The NL-specification (types, grammar and lexicon) is partly domain specific, partly domain independent.

6.2 Determination of the intended conversational act

The utterance types output by the syntactic analyser are considered as indication of a conversational act type. The function of the dialogue and the role of the participants. are important for inference of the conversational act type. What is the speakers intention when he utters a declarative sentence like: "John walks in the park"? Does he inform the hearer that this is the case and should the hearer update his belief state with the belief "John walks in the park"? Or should the hearer update his belief with the information that the speaker beliefs that John walks in the park? Should the hearer verify what the speaker says? How an agent handles these kinds of questions, depends on the definition of its CPs.

6.3 The reference problem

The agent in dialog keeps a record of the actual topics of the dialog in a focus list, consisting of a set of beliefs that the agent has about the user and the objects that are in focus. This focus list is used to resolve referential terms. The QLF may contain such referential terms for indefinites and indexical words or phrases in the natural language utterance. The phrases "a book" and "that box" in the request "put a book in that box" are represented as referential terms (using an informal syntax) `<A,X,[book,X]>` and `<DEMO,X,[box,X]>` respectively and occur in the unresolved conversational act `[imp,[inBox,<DEMO,X,[box,X]>,<A,X,[book,X]>]]`.

For solving reference problems, the agent should know what is 'in focus' during the dialog. For this, we need special predicates and constants that indicate who introduced which *discourse referents*. Discourse referents are special constants, that refer to objects that are introduced during the dialog. Look at the following dialog part:

User: there is a blue book
Agent: okay

After this dialog part, the BS contains:

```
[referent,Rn]
[introducedBy,User,Rn]
[book,Rn]
[blue,Rn]
```

When these referents are removed, depends on the implementation of the agent's CPs.

The task of the Resolver is to resolve such unresolved terms using the current BS of the agent and the beliefs of the agent about the current objects that are in focus. Hence, if in a recent dialog act a particular box was spoken about, the focuslist contains `[infocus, box2]` and the referential term `<DEMO,X,[box,X]>` will be resolved by the object constant `box2`.

In case the Resolver can't find a unique referent, this information is added to the BS of the agent: `[cannot_resolve, box]` or: `[ambiguous,box]`. If the agent specification contains a conditional plan that explains how to handle in this case, the BDP-interpreter will generate a plan—based on its updated BS—that the agent can follow to continue the dialog, for example by asking: `[qref,<A,X,[box,X]>]` followed by the answer `[inform_ref,[box2]]`. Notice that the conditional plans of the agent specification are responsible for the coupling of issues raised in a question asked by the agent ("what box do you mean?") and the referent responded by the dialog participant to which the question was adressed ("box2"). So the agent knows that he should interpret "box2" as an answer to his question.

7 Multi-modal interaction

This section discusses how our BDP agents can operate in a multi-modal environment. The first subsection will explain this in general, the second subsection concerns the relation between an acting BDP agent and a physical environment.

7.1 Multi-modal interaction using the BDP agent framework

Suppose that a user points with a mouse-pointer at a book on the screen—showing a virtual library—and asks "who is the author of this book?". The mouseEvent handler sends a position of the mouse Event to the Inventory of the VirtualWorld that returns information about the object in the Inventory that has this position. This information can be sent to one or more agents in the following form:

```
[POINTED_AT,sender,time_stamp,BOOK#23019]
```

For example, agent A includes this knowledge in his BS. Now he knows that agent with name 'sender' pointed at some object that has unique identifier, known to the agent as object of type book (i.e. the belief `[book,BOOK#23019]` is in his BS). This object is now in focus of the dialogue. At about the same time the agent receives a message from the NL-analyser with the following query:

```
[whq,quant(wh,X,[author,X],
                [author,&lt;DEMO,Y,[book,Y]&gt;,X])]
```

The term `<DEMO,Y,[book,Y]>` is resolved in the belief context of the agent by unification of the variable `Y` with the object in focus that matches type book. The unification result is the resolved term `BOOK#23019`. After substitution of this resolved term in the query the agent can interpret the resulting resolved query:

```
[whq,quant(wh,X,[author,X],
                [author,BOOK#23019,X]]]
```

He may then ask for example a librarian agent the name of the author of the book in question and send the answer to the agent from which he received the query.

7.2 Relation of a BDP agent to the physical world

Since the whole system is based on the principle of agents sending and receiving messages containing communicative actions, the performance of a physical action (like picking a book from a box) is realized by the agent sending a request to the Virtual Environment Agent (VEA) to perform the action. The VEA agent returns a message to inform the requesting agent about the succes or failure of the action. The Inventory and the Graphical Representation of the VE are updated, showing the new situation in the VE. Also the agent's BS is updated.

Making observations occurs in a similar way. The agent sends a message to the VEA "I'm looking in that direction. What am I seeing?" and the VEA responds with a message "You are looking at `book3`". The agent then updates its BS by adding the belief that it is looking at `book3`.

8 Future work and conclusions

Until now the system (implemented in Java) has shown to be valuable to specify simple dialogues between agents and between agent and an NL-user.

There are several directions to continue work in. In the current implementation the typing system for the typed feature structures is not used by the Resolver nor in the formalism for representing the agent's beliefs. So the Resolver can only resolve the term "opera" in case this was literally mentioned earlier in the dialog. If the agent's focus list contains the term "performance", he cannot resolve "opera" with a performance. In a type system in which an opera is declared as a subtype of performance the Resolver could unify these types and solve the referential term. In general a type system allows the Resolver to complete partial specification output by the parser using dialog state information.

Generating plans is a time-consuming task. We plan to use the agent system into already existing larger VE's like the VMC (information desk agent, navigation agent) and the Jacob system (Instructor agent). Experiments with such more involved multi-modal VE's have to show how 'intelligent' agents can be without running into real-time performance problems.

Further, since the belief state part of the agent specification uses the same formalism as the one used for the representation of the logical contents of the conversational acts, the natural language system can also be used to partly specify the agent by means of natural language (like the definition of the mail agent in Section 2). A tool for processing an agent specification in NL is a future project.

References

[Alshawi, 1992] Hiyan Alshawi, editor. *The Core language engine*. MIT Press, 1992.

[Cohen and Levesque, 1990] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[Nijholt and Hondorp, 2000] Anton Nijholt and Hendri Hondorp. Towards communicating agents and avatars in virtual worlds. In A. de Sousa and J.C. Torres, editors, *Proceedings EUROGRAPHICS 2000*, pages 91–95, August 2000.

[Nijholt and Hulstijn, 2000] Anton Nijholt and Joris Hulstijn. Multimodal interactions with agents in virtual worlds. In N. Kasabov, editor, *Future Directions for Intelligent Systems and Information Science*, Studies in Fuzziness and Soft Computing, pages 148–173. Physica-Verlag, 2000.

[Nijholt, 2001] Anton Nijholt. Towards communicating 3d embodied agents. In *Synsophy International Workshop on Social Intelligence Design*, Matsue, Shimane, Japan, 2001. to appear.

[Norvig and Russell, 1995] P. Norvig and S. J. Russell. *Artificial intelligence: a modern approach*. Prentice Hall, 1995.

[Sadek *et al.*, 1997] M.D. Sadek, P. Bretier, and F. Panaget. ARTIMIS: Natural dialogue meets rational agency. In *Proceedings of IJCAI97*, 1997.

[van Luin *et al.*, 2001] Jeroen van Luin, Rieks op den Akker, and Anton Nijholt. A dialogue agent for navigation support in virtual reality. In J. Jacko and A. Sears, editors, *Proceedings ACM SIGCHI Conference CHI 2001: Anyone. Anywhere*, pages 117–118. Association for Computing Machinery, March/April 2001. Extended Abstract.

[Weiss, 1999] Gerhard Weiss, editor. *Multiagent systems : a modern approach to distributed artificial intelligence*. MIT Press, 1999.

[Wooldridge, 2000] Michael J. Wooldridge. *Reasoning about Rational Agents*. MIT Press, 2000.

[Zwiers *et al.*, 2000] J. Zwiers, B. van Dijk, A. Nijholt, and R. op den Akker. Design issues for navigation and assistance agents in virtual environments. In A. Nijholt, D. Heylen, and K. Jokinen, editors, *Twente Workshop on Language Technology 17*, pages 119–132, 2000.