Natural Language Control and Paradigms of Interactivity

Marc Cavazza and Ian Palmer

Electronic Imaging and Media Communications Department
University of Bradford
Bradford, BD7 1DP, United Kingdom
{m.cavazza, i.j.palmer}@bradford.ac.uk

Abstract

There is considerable interest in the prospect of controlling computer games in natural language, as this would open the way to new gaming experiences. In this paper, we report results from a small, yet fully-implemented, prototype for natural language control based on a DOOM™ game emulator. The system, developed on top of a parser integrating syntactic and semantic processing, is able to analyse sublanguage commands into appropriate game actions. One key aspect of natural language control is the proper interpretation of action semantics from the natural language input to the behaviour of the player character in the game. In the long term, the use of natural language input would require compatible AI techniques supporting real-time behaviours. It also has implications, yet to be explored, in terms of artificial actors' autonomy.

Introduction

In this paper, we report results from our first experiments carried out with a prototype for Natural Language Control (NLC) a of computer game. Our initial objectives were, beyond the implementation of NLP techniques, to investigate the relations between NLC, game AI and interactivity paradigms. We describe the techniques used to generate behaviours from NL input and draw conclusions for the integration of NL input with AI techniques underlying agents' behaviours. We also raise questions about possible evolutions of AI techniques in relation to real-time behaviours and reactivity.

NL input could be seen just as a user-friendly interface that substitutes for more traditional menu-based interfaces. It is actually a new interaction paradigm that is strongly related to the AI techniques used in computer games. Possible uses of speech/NL input in games reflect this evolution towards extended interactivity and new gaming experiences. These include:

- Rapid selection of items from a list, such as weapon selection and/or weapon programming (in air combat, naval warfare or space combat games). The amount of NLP might be minimal in some cases.
- Selection of complex configurations as an alternative to multi-level menus [Gentner and Nielson, 1996]. It has been reported that performing a command sequence can involve more than 15 elementary menu operations. This is even more relevant to console-based games, which are not always properly equipped for menu-based interaction.
- Increasing the realism and believability of artificial actors. One can distinguish here between NL control of a player character and conversation with artificial actors. In the former case, emphasis is on control of the character's behaviour, while in the latter it is more often on information access.
- Controlling a game (e.g. for strategy games) through appropriate high-level concepts without having to specify the primitive actions into which the high-level tasks should be decomposed.

In the next section, we describe the output of the NLP module and the early stages of its processing by the animation system.

The Integrated NLP Module

Our prototype is based on an NLP module developed on top of a proprietary animation system (REALISM), which is used to emulate the main features of the DOOM™ game, *adapted to a third-person perspective*. The NLP methodology followed the habitability principle [Zoltan-Ford, 1991] [Wauchoppe et al., 1997]. After conducting a corpus study of on-line DOOM™ spoilers, we designed a parser which is able to process the most relevant syntactic constructs encountered when formulating a command. These commands generally involve a small number of gameplay actions in the context of complex spatial descriptions: "pick up the stimpack at the end of the room", "turn right after the barrel", "open the blue door with the skull key", etc. It should be noted that we have not created a complete copy of the DOOM™ game, but that we have taken the desired features from this to create a DOOM™-like environment to allow us to experiment.

Parsing is based on the Tree-Furcating Grammar formalism [Cavazza, 1998], which supports an integrated syntactic and semantic analysis. This facilitates both

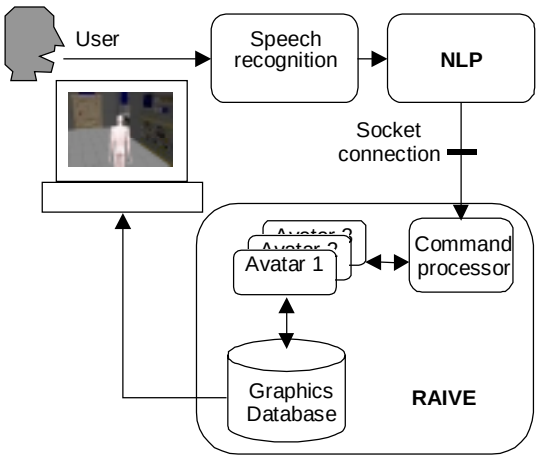

Figure 1: System architecture

linguistic data description and the resolution of traditional problems, such as syntactic attachment ambiguities. These are highly relevant due to the high frequency of PP-groups introduced by spatial prepositions (at, to, close-to, near, on, etc.). The final output of the "linguistic" step is a semantic feature structure that represents the action described by the command, together with its parameters (Figure 2). This feature structure is built synchronously during the parsing process, by aggregating semantic features from different words along the functional links provided by the syntactic relations. From this semantic representation, a structured command message is generated and passed to the animation system. The structured messages are unambiguous, so that processing of a sentence such as "shoot the baron near the stairs with the plasma gun at the left of the room" would generate a [S E BARON L STAIRS I PLASMA_GUN L LEFTROOM] command. Use of such command strings can also serve as a basis for distributed messages.

From the command string, the animation system i) identifies the discourse objects and ii) generates appropriate behaviours in the system to simulate the course of action. In the following sections, we will see how the structure of the command message provides a uniform basis for these operations.

The architecture of our prototype is as shown in Figure 1. The Natural Language Processing (NLP) component connects through a socket to the RAIVE (REALISM Artificial Intelligence Virtual Environment) system that emulates the DOOM™ functions. This allows the system to run on separate machines, and the system has been tested running on either single or dual SGI O2 workstations. The control messages are passed from the NLP module via the socket to the RAIVE module. The NLP module acts as a client that produces one command string for every natural language sentence entered, passing these to the RAIVE module to be processed.

To illustrate the process, consider the sentence "shoot the demon near the blue door on the left of the room with the chain gun near the barrel". This would generate the command string [S E BARON L DOOR_BLUE L LEFTROOM I CHAIN_GUN L BARREL]. The processing of this string can be broken down into:

(i) Identification of discourse objects based on their relative spatial locations.

(ii) Identification of game actions and their parameters.

The string is parsed to achieve these goals as follows. As the command strings are of moderate complexity and there is a need to perform simultaneous operations on the graphic database, we have adopted a procedural approach to command string parsing. The first symbol always indicates the main action. In the example, the 'S' is processed as signifying a shoot action, followed by the 'E' which introduces the action object (i.e. target). The target is optional, as are many parameters, and its omission will result in the guided actor firing in the direction that it is currently pointing.

As there is a target in this example, the system now knows that it must search the graphics database for a baron, but in this case there is a location condition associated with the target. This is indicated by the 'L' followed by a string identifying a blue door. The door is also followed by a location condition (the next 'L') specifying the left of the room. The aggregated location parameters will thus be used as a filter to be applied on the graphic database, in order to identify discourse objects. The initial baron target parameter is pushed on a stack and the system then searches the graphics database for blue doors. The first blue door found is checked for proximity to the left of the room. Each room has four default landmarks, the front, the rear (or end), the left and the right relative to the main entrance to the room. Each landmark has a proximity distance associated with it, and in the case of the default landmarks (and in practice many others) these overlap, i.e. it is possible to have an object at the 'left-front' of the room. If the door is not on the left of the room, the next blue door in the database is processed and checked for proximity to the left of the room. This is repeated until either a door satisfying the condition is found or the search fails and the command is aborted. If a door is found, it is stored and then the database is searched for a baron. Once found, the baron is checked for being 'near' the door in the same manner as discussed for the door location. Processing of candidates continues until the first is found that satisfies the location condition. If none are found that do, the command is discarded. If a baron is found that does satisfy the condition, the identifier of the baron is pushed on the guided actor's target stack and processing of the command continues.

The next section of the string, the 'I' specifies an instrument for the action, i.e. a weapon. If the actor already has the specified weapon (in this case a chain gun) in its inventory, then this is selected. However, this particular string specifies a location for the weapon, so the implication is that the actor will not already possess a chain gun. The location of the weapon is specified as being

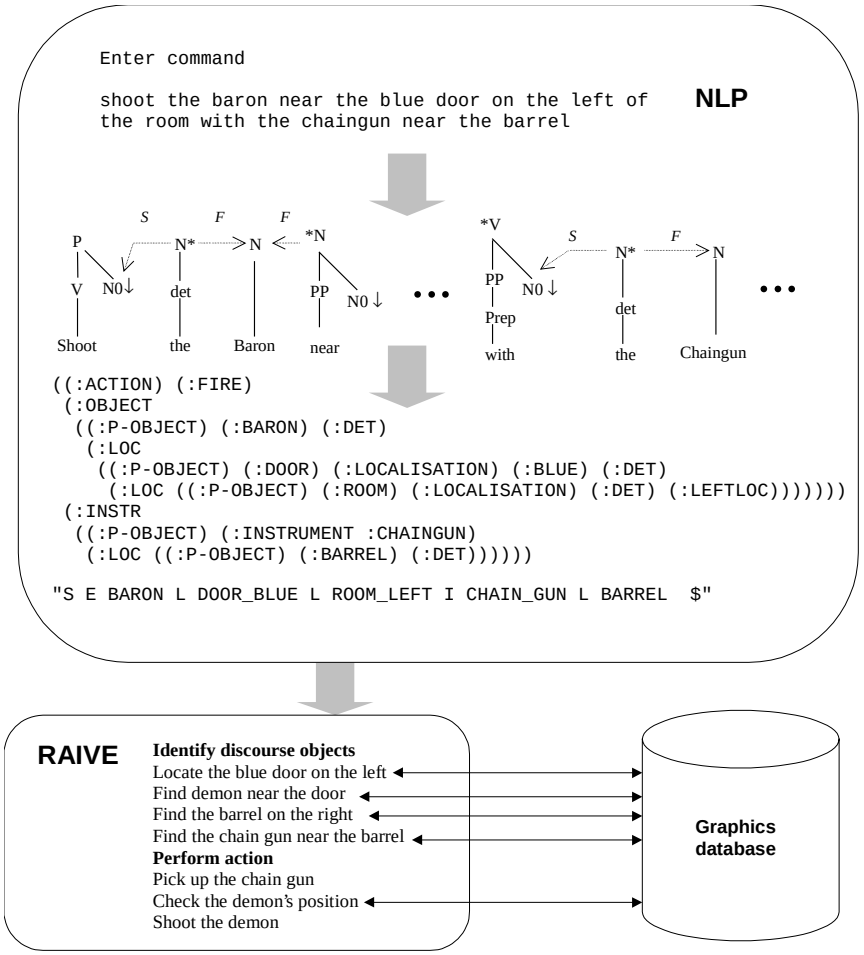

Figure 2: Example of command processing

near a barrel, and the barrel has a location condition, this time on the right of the room. Again, the graphics database is then searched for a barrel and its location is checked against that specified in the command string. If one is found, then the database is searched for a chain gun that is near the barrel, which is then pushed on the target stack, later enabling the guided actor to pick up the gun. At this stage, the system analyses all the pre-conditions required to trigger appropriate behaviours (e.g. it identifies the chain gun to be collected). The generation of the actual behaviours is discussed in later. Figure 2 summarises the command interpretation process.

This example demonstrates a number of features about reference resolution at the animation level. The first is that the system has the ability to cope with multiple targets which it stores on a stack together with their associated actions (e.g. demon/shoot, chain gun/pick up). In this way, a single command may result in a script being generated consisting of a number of sequential sub-actions at the RAIVE level.

The second feature is that if the actor already possessed a chain gun in its weapon inventory, then the actor would not have moved to pick up the chain gun specified, i.e. it would have ignored the second part of the command. This is for two reasons. Firstly, only one chain gun may be carried at any time (there are limits on how many of each type of item that can be carried). This means that the specified one could not be collected anyway. Secondly, one chain gun is as good as any other (i.e. the actor will do the minimum work to satisfy any command).

The third feature is demonstrated by the fact that the baron may have moved between the command being issued and the guided actor obtaining the chain gun. This does not prevent the system from identifying the correct baron. The actor will obtain the current location of the demon from the graphics database just before firing and turn to aim at that position. In this way the system can handle the discourse objects in a dynamic environment after a single identification step.

Also, note that if there were only one baron and one chain gun in the room, the command "shoot the baron with the chain gun" (resulting in the action command string [S E BARON I CHAIN_GUN]) would result in exactly the same action as the more complex command given above.

Overall Results

Though we used a small vocabulary of less than 200 words, we encountered few problems in the expressivity of our command language, considering the very limited set of actions that are part of the simulated DOOM™ game. Our parser can process commands with many embedded spatial prepositions or definite noun phrases: sentences up to 25 words are easily parsed, while the average command is rather 10-15 words in length. Response times are fast, in the order of 100 ms on a 150MHz R1000. This is required for use with a speech recognition system, as our preliminary experiments using the same vocabulary and grammar have shown recognition times to be in the 500ms range [Cavazza and Palmer, 1999]. However, we have been mainly using keyboard input for the initial system assessment (which did not include user acceptance, response times, full gameplay, etc.).

The overall performance of the NLC prototype depends on parsing, semantic analysis and behavioural interpretation. For most commands in our initial test set, the system was able to generate a proper behaviour corresponding to the command. Failure to do so was often the result of incorrect descriptions for lexical data that could subsequently be corrected. Strongly ungrammatical sentences, i.e. beyond what is normally expected to be part of the oral command style as identified in the design phase, would often result in the parser failing to produce any usable output. On the other hand, correct but meaningless sentences, for instance commands corresponding to actions which pre-conditions are not satisfied (e.g. reaching non-existing landmarks), resulted in no action being taken while the system remains ready to process incoming commands. Also, the system is able to process *command sequences*, as the pre-conditions of some actions ("turn left after the barrel") assume that other actions have been carried (moving towards a barrel). The mechanism implementing this feature will be outlined in the next section. We will describe in more detail the final interpretation steps, which correspond to behaviour generation in the animation system. This has been facilitated by using a behaviour-based animation system such as REALISM to emulate the DOOM™ features.

Action Semantics: NLC and Behaviours

NLC is confronted with some fundamental problems in NL semantics when it comes to interpreting actions, or designing a course of action in response to a command. Namely, that there is no direct correspondence between a single high-level action and the detailed behaviour that would implement such an action. This derives from the fact that the high-level semantics of an action most often involves a plan-like decomposition. This is for instance the case with "fetch" [Takeda et al., 1996] or "pick up" actions which can be subdivided as i) moving towards the object and ii) collecting the object once in contact. Action parameters can introduce similar plan-like decompositions when they include dynamic information (one example being changing directions once a landmark is reached, e.g. in "Turn right after the barrel"). The proper interpretation of an action thus consists in mapping the lexical content (as provided by linguistic resources) to the action primitives available in the gaming environment. Because of the difficulty of generating and solving plans, especially when integrating several actions, previous work in NLC has made use of compiled plans or procedural approaches [Geib, 1994] [Kurlander and Ling, 1995]. Another recent solution to this problem has been suggested by Badler et al. [1999], which consists in defining ontologies for actions (or *actionaries*) to be embedded in the animation system. This would lead to a convergence between the animation primitives and semantic elements uncovered by the NLP step. The main difficulty remains to determine how to take into account gameplay concepts, whether as part of a plan-based decomposition or of an action ontology.

Our own approach to command interpretation is largely procedural in its nature. Some of these procedures can be directly generated from action pre-conditions and the explicit contents of the command string. This is the case for instance when fetching the chain gun in Figure 2. Other procedures required to cope with more complex processes are part of the action descriptions themselves. Iterative actions, actions whose pre-conditions may change over time and complex spatialised actions, such as path planning, fall under this category.

Primitive motion actions such as Walk and Run actually use a generic locomotion behaviour in the character, the two commands merely using different velocities. The command interpreter at the character level identifies the action command and then passes an appropriate velocity to the locomotion behaviour, which will store the new velocity and then use this during the next simulation step to move the character. In this way it is possible to have more a sophisticated set of motions, such as 'side-step' or 'jump', by merely passing new parameters to the locomotion engine in the character. Similar modifications would be possible for the other actions, such as "shoot repeatedly".

Complex motion requires an appropriate script to be generated by the RAIVE system. For example, the *turn* (right & left) actions have an optional 'after' parameter, e.g. the result of an instruction "turn right after the barrel" would be [R A BARREL]. The script generated by this command has a procedural interpretation in terms of RAIVE actor behaviours to cope with the dynamic nature of the action pre-conditions. In the 'turn' example, one procedure constantly checks the target position relative to the actor. When the target is behind the character, the behaviour uses a procedure to set the velocity of the actor to a new value to implement the command. These procedures are part of the actor representation and are not generated dynamically.

In the current system, there is a delay between the actual passing of the object and the turn to ensure that the

character does not collide with the target object. This has been set experimentally at present but in future implementations will use target size to ensure a correct delay between passing and turning. The use of temporal relative condition parameters implies two things: the ability of the character to store information for use in later sub-actions of a script and the fact that some action commands may result in no change in character state. This latter point is because if the actor is not in motion when a turn or halt command is issued with an after parameter, then no change in its velocity will occur.

We can also illustrate the problem of action semantics with an example from path planning, which is a significant component of Game AI for at least two game genres for which NLC is relevant (adventure games and strategy games). Typical high-level instructions would refer to complex trajectories not only mentioning a goal location or landmark, but also giving indication as which area to avoid or to walk through. Also, some additional constraints could be included, such as not turning one's back to a door. If we consider traditional geometric resolution methods such as A*, the path is computed off-line and then followed by the character. If we leave aside the problem of dynamic environments, we are still left with the problem of converting high-level semantics into the search-based path planning process. In this context, if a command specifies regions to be avoided or to be walked through, this can be simply interpreted by changing cost functions for cells in these regions. But taking into account more complex semantics (such as avoiding "danger" areas, which is a game-related concept, depending on the specific conditions) demands other mechanisms. We have thus proposed a variant of ε-admissible variant of A* in which a secondary heuristic can be used to accommodate some application-dependent data [Bandi and Cavazza, 1999], as a partial solution to this problem. This again illustrates that NLP techniques cannot be considered in isolation.

NL and Interactivity

We have seen that NLC has to tackle problems of action semantics in order to generate appropriate behaviours. However, the use of NLC also impacts on the control paradigms for player characters. This can be illustrated through two important concepts, which are i) control timescales and ii) character autonomy.

NL Control of a computer game is only possible if NLP timescales are compatible with gameplay timescales. If we exclude the case where turn-taking determines NL input, there is a need for processing NL input in real-time. This can require some AI computations to be interrupted to take into account new information or commands inputted, or the behaviours taking place as a result of previous computations to be superseded. *An important conclusion which confirms our initial analysis is that in the general case NL-based interactivity would imply reactive or real-time AI techniques supporting the system's behaviours.* Let us assume that we want the game to be under total control from NL input. This means that unless a natural interaction cycle can be defined as part of the gameplay, there is a need to interrupt any AI computation to take into account additional input that may either complement or alter the current plans. This would in theory imply the use of either reactive techniques, real-time AI or some anytime implementation of AI techniques. The alternative, also depending on the availability and efficient implementation of the above techniques, consists in defining a set of ad hoc elementary behaviours, reactive enough to ensure a fine granularity for action. If we took search algorithms as an example technique (without specifying their actual use in terms of game behaviour), the various approaches could be represented by traditional search (large timescales), real-time search (timescales can be fixed depending on the gameplay) and agent-based reactive techniques (short timescales, sub-optimal solutions). It would also appear that a smaller granularity for action facilitates the integration of NL into action planning, as it simplify the problem of hierarchic decompositions of meanings.

In the prototype we have implemented, player characters have very limited autonomy, corresponding to the required ability to carry low-level actions, such as climbing stairs or avoiding obstacles. *However, another approach to NLC is to have largely autonomous but "obedient" characters.* An obedient character can be defined as an autonomous character for which user commands have absolute priority over its own problem-solving capabilities. This solves the conflict between gameplay and autonomy. An obedient character will always apply the player's strategy, even if this leads to a non-optimal or poor behaviour. This ensures that the issue of the game is under the control of the player. On the other hand, the autonomy of the character enables it to cope with low-level tasks or to complement under-specified instructions from the player.

Finally, a possible conclusion would then be that our DOOM™ application is *not* adequate in terms of interactivity and timescales for player actions. Another option would be that NLC is only used for "tactical" instruction (especially for games in the so-called "sneak-them-up" category (Metal Gear Solid™, Syphon Filter™, Tenchu™), for which most of gameplay consists in avoiding high-intensity combat phases).

Conclusions

A limitation of our approach is certainly that we have developed a NL interface to a game whose structure was essentially developed for direct "physical" control. While this might be inappropriate from a narrative/gameplay perspective, we claim however that it was a useful starting point to gain a proper understanding of high-level control issues and the various levels of control that are strongly related to game design.

Our conclusion is that intelligent/autonomous, but obedient characters would support NLC much better than

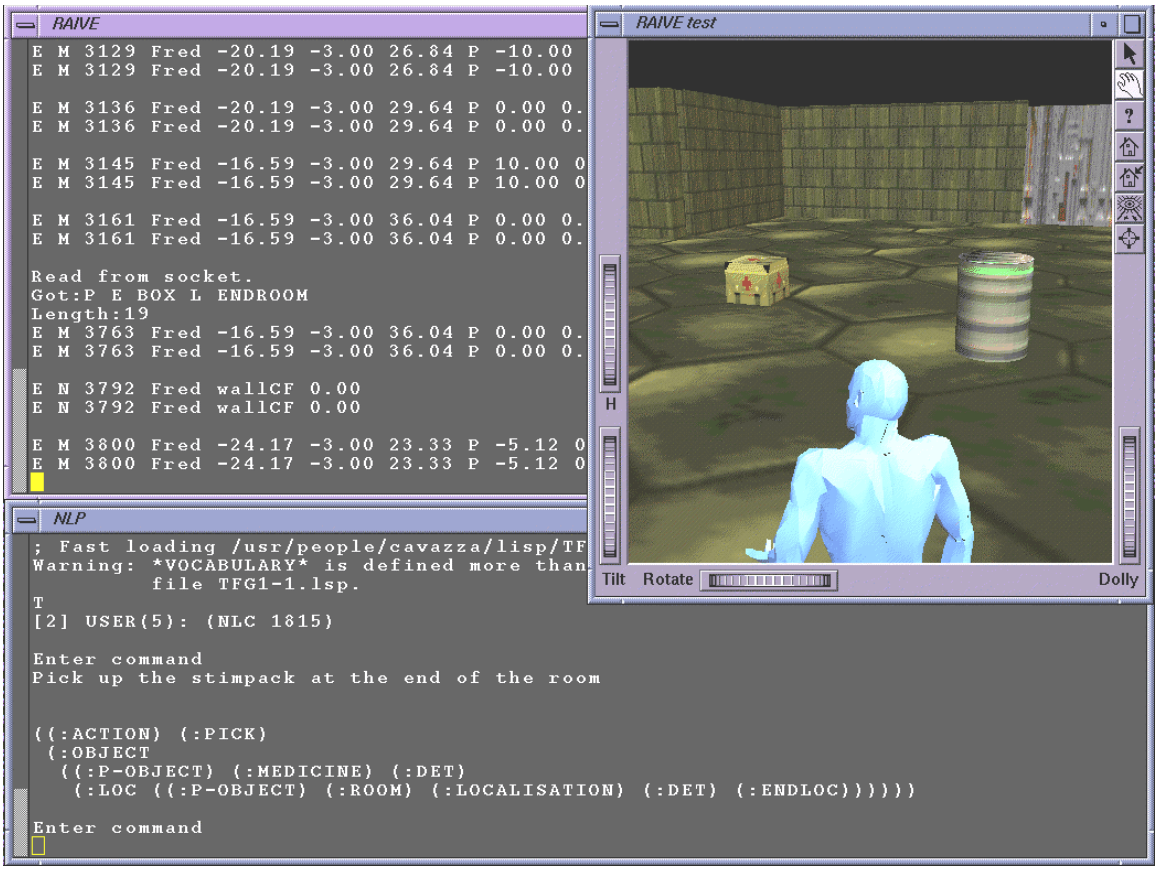

Figure 3: The Prototype in Operation

characters with limited autonomy. This might make the perspective of NLC more remote but links the success of NLC to the development of autonomy and real-time AI techniques.

In this context, NLC of character-based games would be appropriate under the following circumstances:

- strong “personality” of the player character as part of the game design (Solid Snake, Lara Croft, etc.) which creates an “emotional link” between the player and the character that can motivate spoken interaction
- course of action can be planned off-line, i.e. the game has many low-intensity phases during which key actions to be taken can be decided.
- gameplay/storytelling based on key tactical choices, with options complex enough to benefit from a high-level description

However, the first potential areas of application for NLC could be constituted by games with large timescales, complex commands and a multiplicity of actors to control, such as strategy or simulation games. Most of the integration problems we have addressed in the course of our experiments are still relevant in this context, as well as the techniques we have developed to cope with them.

References

Badler, N., Bindiganavale, R., Bourne, J., Allbeck, J., Shi, J. and Palmer, M., 1999. Real time virtual humans. *Proceedings of Digital Media Futures*, Bradford, UK.

Bandi, S. and Cavazza, M., 1999. Integrating World Semantics into Path Planning Heuristics for Virtual Agents. *Proceedings of the Second Workshop on Intelligent Virtual Agents*, Salford, UK., pp. 23-32.

Cavazza, M., 1998. An Integrated TFG Parser with Explicit Tree Typing. *Proceedings of the fourth TAG+ workshop, Technical Report IRCS-92-18*, University of Pennsylvania, pp. 34-37.

Cavazza, M. and Palmer, I., 1999. A Prototype for Natural Language Control of Video Games. *Proceedings of VSMM’99 Conference*, Dundee, Scotland, pp. 36-45.

Gentner, D, and Nielson, J., 1996. The Anti-Mac Interface. *Communications of the ACM*, 39, 8, pp. 71-82.

Geib, C., 1994. The intentional planning system (ItPlanS). In *Proceedings of AIPS'94.*

Kurlander, D. and Ling, D., 1995. Planning-Based Control of Interface Animation, *Proceedings of CHI'95*, Denver, pp. 472-479.

Takeda, H., Iwata, K., Takaai, M., Sawada, A. and Nishida, T., 1996. An Ontology-based Cooperative Environment for Real-world Agents. *Proceedings of Second International Conference on Multiagent Systems*, pp. 353-360, Kyoto, Japan.

Wauchoppe, K., Everett, S., Perzanovski, D., and Marsh, E., 1997. Natural Language in Four Spatial Interfaces. *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pp. 8-11.

Zoltan-Ford, E. 1991. How to get people to say and type what computers can understand. *The International Journal of Man-Machine Studies*, 34:527-547.