Authoring Embodied Agents' Behaviors through Natural Language and Planning *

Jan M. Allbeck, Rama Bindiganavale, Karin Kipper, Michael B. Moore, William Schuler, Norman I. Badler, Aravind K. Joshi, and Martha Palmer
University of Pennsylvania
200 S. 33rd St.
Philadelphia, PA 19104
allbeck@graphics.cis.upenn.edu

1. INTRODUCTION

In any virtual simulation or AI based game involving agents, there is a need to create or to dynamically alter agent behaviors. These behaviors can be generated by scripting [5], or by using rules in a rule-based system [4], or by specifying goals that are input to a planning or reactive system [1, 3]. In our system, we incorporate all of these methods and additionally allow a user to dynamically create and modify the behaviors of the agents.

Our goal is to explore an architecture for authoring the behaviors of interactive, animated agents using natural language instructions. The work presented in this paper is an extension of previous work found in [2], which uses natural language input to dynamically alter the behaviors of agents during real-time simulations. In that architecture we were able to give both immediate instructions and conditional instructions to the agents. However, we were not able to give negative directives or constraints, such as "Do not stand in front of the car door" or "Do not walk on the grass." In order to successfully carry out these type of instructions, we need a planner which can dynamically alter an agent's behavior based on constraints imposed by new instructions.

We extend our current architecture to include a planner and the ablility to keep a record or history of all actions successfully completed by the agent. This will also allow us to give conditional, iterative instructions to the agent, such as "Check every abandoned vehicle." The history feature ensures that the agent will correctly check every abandoned vehicle only once. The planner also allows us to give abstract instructions such as, "Take cover behind the drum." This instruction does not include information about how to take cover behind the drum. The agent could walk, run, crawl, swim, or do any number of other translatory actions to position itself behind the drum. Also, the instruction has no information about the path the agent should take. The planner will provide the information that is needed for animation but not provided by the natural language input.

2. ACTIONARY

The *Actionary* (Figure 1) is the core component of our system. It contains persistent, hierarchical databases of agents, objects, and actions. The agents are treated as special objects and stored within the same hierarchical structure as the objects. Actions are represented as PARs (Parameterized Action Representation). Each PAR can either be uninstantiated (UPAR), containing only default properties for the action or be instantiated (IPAR), containing specific information about the agent, objects, and other properties. All the UPARs are stored hierarchically within the *Actionary*. PAR schemas are similiar to UPARs, except that their hierarchy is derived from natural language verbs and semantics, whereas the UPAR hierarchy is derived from motion semantics within the animation domain.

During the initialization phase of a simulation, the *Database Manager* loads relevant parts of the *Actionary* into the *World Model*. The model is constantly updated during the simulation, recording any changes in the environment or in the properties of the agents and objects.

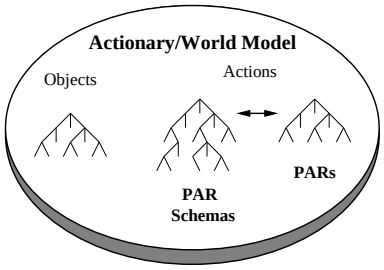

Figure 1: Actionary and World Model

3. ARCHITECTURE

Figure 2 shows the new architecture of our system.

The user inputs natural language instructions for a specific agent through a GUI. The *NL transducer* parses the instructions, translates them into situation calculus expressions encapsulating references to PAR schemas, and sends them to the *Agent Process*.

Within the *Agent Process*, these expressions are stored as a list of *Desired Situations* along with other situation calculus expressions generated from previous instructions. The *Desired Situations* set collectively represents the desired future behavior of the agent. The *Rule Manager* uses the *World Model* to evaluate the expressions in the *Desired Situations* set and sends them to the *Goal Manager* when there are any changes. All actions successfully completed by the agent are

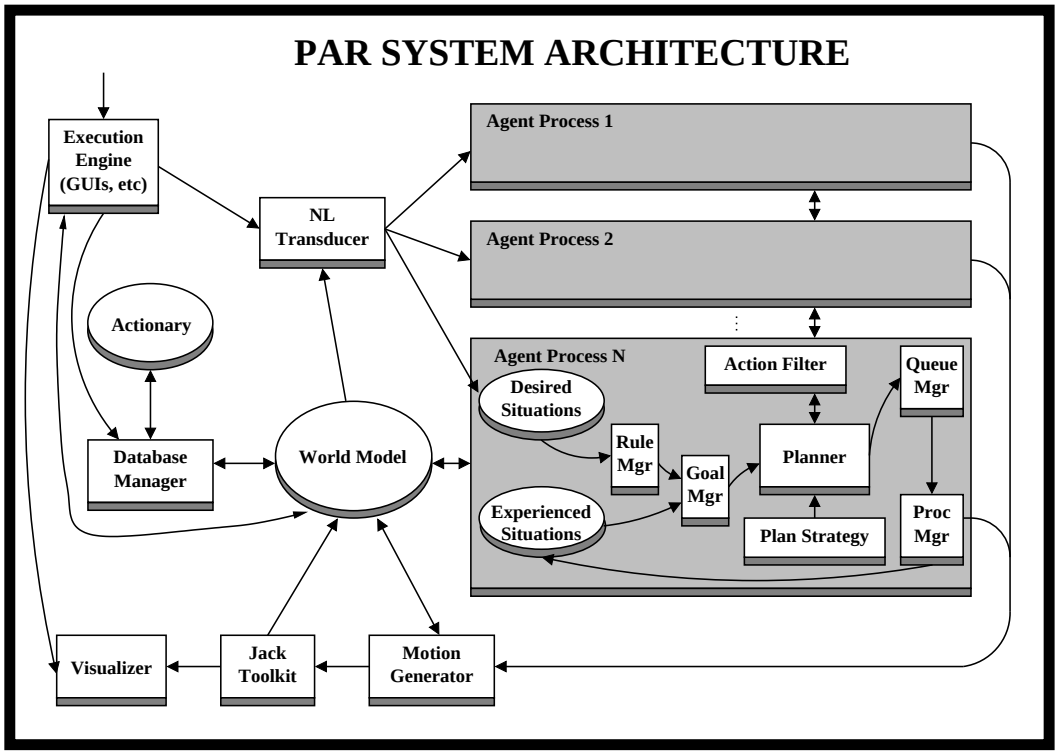

Figure 2: New Architecture Diagram

stored as a list of *Experienced Situations*. The *Goal Manager* uses the *Desired Situations* and the *Experienced Situations* to determine new goals and sends them to the *Planner*. The *Planner* further evaluates the situation calculus expressions and retrieves the PAR schemas from them. For each PAR schema, the *Planner* needs to retrieve the set of all relevant UPARs from the *World Model*. The *Action Filter* first eliminates some of the UPARs from the set based on the agent's capabilities, and sorts the remaining UPARs based on the agent's characteristics. For example, the *Planner* may ask for all of the translatory PAR actions. If one of the actions returned is "walk" and the agent is an infant not capable of walking, "walk" will be removed from the set. Also, if the agent is a happy child, the *Action Filter* may prefer "skip" to "walk," but if the agent is a dominant, business woman "walk" would be preferable. The ordering of actions is based on the agent's role, metamotivational state, perception of the situation, culture, personality, and emotions. The *Planner* solves an abstract planning problem where the initial state comes from the *World Model*, the goal state from the *Desired Situations*, a preference order of available actions from the *Action Filter*, parameters to select plan structure from *Plan Strategy*, and constraints, also from *Desired Situations*, are used to eliminate possible plans from consideration. These IPARs are processed by the *Queue Manager* and *Process Manager* and then sent to the *Motion Generators* for the actual execution of the action. After successful completion of the action, the *Process Manager* adds the relevant information about the action to the list of *Experienced Situations*. The multiple agents in the environment communicate with each other through message passing.

4. ACKNOWLEDGMENTS

This research is partially supported by U.S. Air Force F41624-97-D-5002, Office of Naval Research K-5-55043/3916-1552793, AASERTs N00014-97-1-0603 and N0014-97-1-0605, NSF IIS-9900297 and SBR-8900230, Army Research Office ASSERT DAA 655-981-0147, NASA NRA NAG 5-3990, and Engineering Animation Inc.

5. REFERENCES

[1] J. Bates, A. Loyall, and W. Reilly. Integrating reactivity, goals, and emotions in a broad agent. In *14th Annual Conference of the Cognitvie Science Society*, Indiana, July 1992.

[2] R. Bindiganavale, W. Schuler, J. M. Allbeck, N. I. Badler, A. K. Joshi, and M. Palmer. Dynamically altering agent behaviors using natural language instructions. In *Proceedings of Autonomous Agents*, 2000.

[3] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning, and planning for intelligent characters. In *SIGGRAPH'99*, pages 29–38, 1999.

[4] W. L. Johnson and J. Rickel. Steve: An animated pedagogical agent for procedural training in virtual environments. *SIGART Bulletin*, 8(1-4):16–21, 1997.

[5] K. Perlin and A. Goldberg. IMPROV: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH '96*, pages 205–216, Aug. 1996.