Planning with Smart Objects

Tolga Abaci Jan Ciger Daniel Thalmann
Virtual Reality Laboratory
EPF Lausanne
CH-1015 Lausanne, Switzerland
{tolga.abaci | jan.ciger | daniel.thalmann}@epfl.ch

ABSTRACT

This paper presents a novel method of employing "smart objects" for problem solving in virtual environments. Smart objects were primarily used for behavioral animation in the past. The paper demonstrates how to use them for AI and planning purposes as well. We formally define which operations can be performed on a smart object in terms of their requirements and their effects. A planner uses this information to determine the correct sequence of actions needed to achieve a goal. This approach enables intelligent agents to solve problems requiring a collaboration of several agents and complex interactions with objects.

Keywords

Smart objects, artificial intelligence, virtual reality, animation, planning.

1. INTRODUCTION

A significant number of virtual reality applications require that the virtual characters are able to manipulate the objects in their environment. Such interactions can be arbitrarily complex and their precision requirements vary as well (i.e. ranging from simple, single-shot motions to sequences of numerous motions that require high accuracy). Many existing applications try to tackle this problem in ad-hoc ways; the usual solution is to combine pre-designed or motion-captured key frame animations with simple object animations. Another, more general, approach is to use a concept of smart objects where the responsibility for the animation is shared between the virtual character and the object itself [Kal01].

The smart objects paradigm has been introduced for interactions of virtual humans with virtual objects [Kal01]. It considers objects as agents where for each object interaction features and plans are defined. Even though smart objects are more flexible than other approaches when it comes to animation and behaviors, the fact that interaction plans are fixed imposes a severe limitation from the artificial intelligence point of view, reducing the capability to adapt to new situations and to solve more complex, dynamic problems. This could be addressed by the use of planning techniques.

Conference proceedings ISBN 80-903100-7-9
WSCG'2005, January 31-February 4, 2005
Plzen, Czech Republic.

One of the first published works about AI planning is the STRIPS planner from 1971 [Fik71]. It introduced the concept of operators, with preconditions and effects. The state of the world is expressed using predicate calculus. This method of describing the planning problem is still popular and was used in many planners – e.g. UCPOP [Pen92], Prodigy [Vel95]. One of the most popular planners using the STRIPS representation is Graphplan [Blm97] and its many derivatives, such as Blackbox, Sensory Graphplan [Wel98], Temporal Graphplan [Smi99] and many others.

Our implementation employs a modified version of Sensory Graphplan (SGP). It extends the standard Graphplan and adds sensing actions and conditional effects. It builds contingency plans – plans where the initial truth value of some predicate may be uncertain and the planner plans for both eventualities indicating which actions have to be taken in each case (planning worlds). As such, it is more suitable for virtual reality simulations because the input language is much more expressive compared to the standard STRIPS-like planners.

In this paper, we present a method how to extend the smart object concept for use with SGP. Embedding the high-level information together with animation

data in the smart object allows for more efficient planning, because only relevant operations and data are considered. Another advantage is that the embedding allows integrating the creation of the logic data into the design pipeline. This ensures that the smart object animations and the corresponding high-level information are created at the same time and in a consistent way.

2. EXTENDING SMART OBJECTS

Smart objects provide not only the geometric information necessary for displaying them on the screen, but also semantic information useful for animation purposes. We store this information in the form of sets of attributes attached to the scene graph nodes of the object.

The attributes convey various kinds of information – e.g. important places on or around the object (e.g. where and how to position the hands of the virtual character in order to grasp it), animation sequences (e.g. a door opening) and general, non-geometric information associated with the object (e.g. weight or material properties) The semantic information in the smart object is used by the virtual characters to perform actions on/with the object, e.g. grasping, moving it, operating it (e.g. a machine or an elevator).

However, simple uses of semantic information are not sufficient if more complex behavior is desired. For example, moving of the crate (a smart object) is easily animated using a script and few attributes of the object. Moving the same crate by a virtual character through a closed door requires a higher level planning (i.e. "open the door first, if not open already and then push the crate through") and moving a heavier crate may require two virtual characters and careful planning to do it. All such simulation requirements put high demands on the script driving the virtual characters because it has to know about all possible situations which may occur. This is impractical and inefficient. Planning has the potential to solve this problem; however, with many possible actions it can become very cumbersome because the complexity of the search space explodes and certain cases may be simply intractable.

There is another way to address this problem. Just as we move the object-specific "animation intelligence" from the virtual characters to the smart objects, we can add also the planning and AI-related data there. The virtual character does not have to know how to interact with every kind of object; he can acquire the necessary capabilities on the fly from the object.

Smart objects [Kal01] already contain "interaction plans", which are essentially scripts containing the animation of the action itself. These scripts coordinate the human and object animations to create the intended result, which could be a virtual human pushing a crate, opening a door, etc. Our proposal for the extension of smart objects consists of the following items to be associated with each action:

- Preconditions for the action. These conditions have to be satisfied in order to be able to perform the action.
- Effects of the action. These predicates will be added/removed from the representation of the world state when the action is performed.

The action script is usually written in some high level scripting language (in our case Python), the preconditions and effects are expressed in PDDL format used by the STRIPS planners (i.e. Lisp-like).

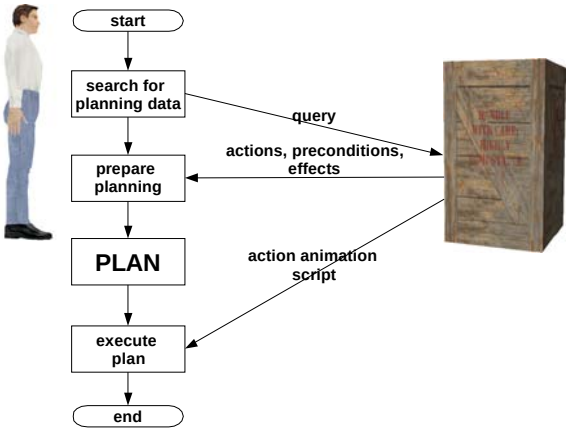

Figure 1 Algorithm outline

The general flow of events when interacting with extended smart objects is shown in Figure 1. The algorithm is as follows:

1. The virtual character collects the relevant information about the current state of the surrounding world. This is taken from the agent's own beliefs about the world and from the smart objects involved in the interaction.
2. Prepare planning step builds the problem representation.
3. Planning is performed. The result is a plan and the set of alternative planning worlds in case that there were some predicates with uncertain value in the initial state. There is always at least one planning world – in this case it corresponds to the initial state directly.
4. The virtual character executes the plan. The actions are taken from the plan, mapped to the corresponding low-level functions and executed.

Sensing actions scheduled by the planner are used to determine the status of the originally uncertain predicates from the initial state. The result determines which planning world the agent is in and therefore decides which branch of the plan has to be executed.

This general algorithm does not guarantee that the agent will be able to solve every solvable problem. It is best-effort heuristics only, because the knowledge of the agent about his surroundings is limited. The agent may not be aware of critical information needed to solve the problem. Another possibility for planning failure comes from the fact that it is very hard to select the relevant objects from which to retrieve the planning information. In fact, this is as hard as the planning problem itself, because the agent will know whether the object is or is not relevant to the task only after the plan is built.

3. RESULTS

The described system was implemented as an extension of our existing virtual reality platform VHD++ [Pon03] and our agent framework described in [Aba04]. The agents driving the virtual characters are implemented as Python scripts; the planner is running as compiled LISP code.

We constructed a simple scenario for evaluation of the proposed approach. In our case, we have a virtual art gallery which received two crates (one large one, one smaller one) with new art. The goal of the two virtual humans is to move the crates inside the lobby of the gallery. Both agents have basic facilities for teamwork (forming a team, disbanding a team), communication and some rudimentary capabilities, like navigation in the virtual environment. However, they do not have any a-priori knowledge about how the crates can be moved.

Our crates are modeled using the extended smart object approach. They contain reference to the file with the geometry of the object (mesh, textures, etc.), position and orientation data for the hands of the agents during the animation, proper position where the agent has to be before the animation script is started and finally the planning data.

The difference between the small and the large crates is that we have defined the large crate as a heavy object and therefore it needs two people to move it.

Planning data have associated animation scripts, in our case a simple animation moving the virtual human and the crate on the screen, using inverse kinematics to keep the hands in position.

a.

```
((((TRANSPORT GINO SMALL_BOX FRONTYARD
    LOBBY))))
```

b.

```
((((RECRUIT-HELP SELF TEAMMATE1)))
 (((MOVE TEAMMATE1 ANYWHERE FRONTYARD))
  (((DECLARE-DOOR-PASSABLE SELF DOOR)))
 (((PREPARE-PUSH SELF BIG_BOX))
  ((PREPARE-PUSH TEAMMATE1 BIG_BOX)))
 (((TEAM-PUSH SELF TEAMMATE1 BIG_BOX
    FRONTYARD LOBBY))))
```

Figure 2 Plans for pushing the small crate (a) and the big crate (b)

We are asking the agent Gino to form a plan to get the big box to the lobby and afterwards to execute it (see Figure 2b). The plan tells the agent to get somebody to help. “SELF” is the agent which submitted the planning request and “TEAMMATE1” is a collaborating agent asked to help. During the execution of the plan the originating agent (team leader) will negotiate the team formation and when successful, it will substitute the real name of a team member for “TEAMMATE1”. The next step consists of moving the teammate from his current position to the front yard, where the crates are. “Anywhere” denotes a special place, from which it is possible to go everywhere. The team leader does not care where the teammate is at the moment, but it needs to establish him into a known state (and move him to the necessary place). The operation “DECLARE-DOOR-PASSABLE” is a helper operation which declares the two places on the sides of the door as connected in case that the door is open. The remaining two steps are self-explanatory. The resulting action after executing the associated animation script from the smart object is depicted in Figure 3.

Figure 3 Two agents pushing the large crate

For the small crate, the resulting plan is simpler since the agent can immediately perform the needed action – Figure 2a. The animation performed by the associated animation script is shown in Figure 4.

Figure 4 Gino pushing the small crate

4. CONCLUSIONS

Our scenario shows the possibility of the virtual characters (agents) to "learn" how to interact with previously not encountered objects by exploiting the information stored in them. Furthermore, such information encapsulation allows us to let the agent work with only the information relevant to his task, simplifying the planning process.

From the design point of view, keeping the animation data and formal representation of the interaction in one place is beneficial for ensuring that all required elements will be created. It is feasible to create an authoring tool for extended smart objects which will help generate the formal representation and animation script template from the specified description.

The process of agent development is simplified as well, because the developer does not have to design an agent capable of performing many specialized actions. It is sufficient to create a simple agent with few basic capabilities (such as navigating the virtual environment) and leave the rest to a generic procedure created on the fly based on the declarations and code defined in the extended smart object. It could be considered as filling in an action template based on the data in the smart object.

We demonstrated how the smart object approach can be extended to handle the formal representation of the interaction. The smart objects can handle not only the animation alone but also the formal description of it. This tight coupling between animation and its formal description enables the virtual characters to perform sophisticated actions which are either very complex to achieve otherwise or just impossible outright. It allows us to apply the known artificial intelligence techniques to improve the realism of the simulation and to provide richer experience to the user.

5. ACKNOWLEDGEMENTS

The project was sponsored by the Federal Office for Science and Education in the Framework of the EU Network of Excellence AIM@SHAPE.

6. REFERENCES

[Kal01] Kallmann, M. Object interaction in real-time virtual environments. Phd Thesis, EPFL, 2001.

[Blm97] Blum, L. A. and Furst, L. M. Fast planning through planning graph analysis. Journal of Artificial Intelligence, Vol.90, pp.281-300, 1997.

[Fik71] Fikes R. and Nilsson, J. N. STRIPS: A new approach to the application of theorem proving to problem solving. Journal of Artificial Intelligence, Vol.2, pp.189-208, 1971.

[Pen92] Penberthy J. S. and Weld D. S. UCPOP: A sound, complete, partial-order planner for ADL. Third International Conference on Knowledge Representation and Reasoning, Cambridge, MA, October 1992.

[Vel95] Veloso M., Carbonell J., Perez A., Borrajo D., Fink, E. and Blythe J. Integrating planning and learning: The PRODIGY Architecture. Journal of Experimental and Theoretical Artificial Intelligence, Vol.7, 1995.

[Wel98] Weld D. S., Anderson C. R. and Smith D. E. Extending Graphplan to handle uncertainty & sensing actions. In proceedings of AAAI 98, 1998.

[Smi99] Smith D. E. and Weld D. S. Temporal planning with mutual exclusion reasoning. In proceedings of IJCAI, pp.326-337, 1999.

[Pon03] Ponder, M., Papagiannakis, G., Molet T., Magnenat-Thalmann N., Thalmann D., VHD++ development framework: Towards extendible, component based VR/AR simulation engine featuring advanced virtual character technologies, Computer Graphics International (CGI), pp. 96-104, 2003.

[Aba04] Abaci, T., Ciger J. and Thalmann D. Speculative Planning With Delegation. In proceedings of CYBERWORLDS 2004, to appear.