Using constraints to describe high-level behaviours in virtual worlds

N. Richard (INRIA/ENST) - Nadine.Richard@inria.fr
P. Codognet (INRIA/SONY CSL) - codognet@csl.sony.fr
A. Grumbach (ENST) - Alain.Grumbach@enst.fr

Abstract

Our aim is to make the authoring of virtual worlds easier for non-specialists, in particular when programming pseudo-intelligent agents within virtual environments. We are designing a high-level language as well as a library of reusable building blocks. We are now working on the definition and the implementation of an homogeneous model for describing 3D objects, their attributes and their behaviours. Constraints are a simple declarative way to describe strong logical relationships between objects; those constraints must be satisfied as much as possible, in order to keep a system in a coherent state. When considering virtual worlds creation, we need to distinguish several kinds of relationships an author may want to express, especially constraints between agents living in an everchanging environment.

Keywords: *virtual worlds, VRML, agent behaviours, constraint programming.*

1. Describing high-level behaviours

Current languages for describing 3D worlds such as VRML (*Virtual Reality Modeling Language*) are more oriented towards scene description than interactive animation and only provide very low-level primitives for animating objects. We need a higher-level language in order to describe behaviours of virtual autonomous agents. In order to keep our model as simple and homogeneous as possible, we assume that an object is an agent, that is any object may have a more or less complex behaviour.

A behaviour can be described as a set of sequences, each sequence being composed of concurrent actions defined within some time periods. We here consider a discrete time: at each time-point, concurrent agents are running simultaneous actions until quiescence and then go to the next time-point.

Because some behaviours like non-collision or gravity may be naturally seen as **constraints**, one of the basic ideas of our approach is to use constraint programming concepts[1] for designing a declarative high-level language. A constraint is simply a logical relationship between several entities. When considering a language with high-level structures such as predicate relations, constraints apply to some variables and thus restrict the degrees of freedom (possible values) the variables can take. A **constraint solver** is in charge of checking the satisfiability of all the accumulated constraints and of providing an adequate solution when constraints are violated. The programmer only has to declaratively state the constraints, while the constraint solver is in charge of defining the operational behaviour.

Constraints are therefore an obvious way to enforce hidden relations between virtual agents (*e.g.* minimal distances or non-collision). They have already been successfully used in 2D authoring systems[2] to maintain geometric properties integrity while drawing, and more recently in 3D animation[3] for motion and kinematics control.

2. Classifying constraints for virtual agents

From a cognitive point of view, the creator of a virtual world would like to define two different types of constraints:

- **Internal constraints** are part of the agent behaviour. For example, one could define constraints like "do not collide with walls" or "follow this particular object".
- **External constraints** are defined between several agents and do not depend on the agents behaviour. For instance, a lead is a strong relationship between a dog and its owner.

From a constraint programming point of view, we would like to offer an homogeneous way for defining both types

of constraints. But we need to distinguish two categories of constraint solving before:

- **Static constraints**, of which solving is launched only once. It could be used to describe non-animated compound objects for scenery set-up or to make a trajectory planification. Then the constraints are "frozen" in some output specification for the 3D world.
- **Dynamic constraints**, which must be satisfied all the time and are continuously re-evaluated. Such a perpetual constraint solving should be useful in an everchanging and unpredictible environment.

By mixing both approaches, we can make some concrete classes explicit:

- Placing five cylinders around a circle is an external and static constraint: the result of the constraint solving phase is converted for generating a 3D scene, once for all before the creation of the virtual world.
- Finding a way in an already known labyrinth is an internal and static constraint: the result given to the agent is a precomputed trajectory – observe that we will only consider that the agent has a perfect knowledge of the scene graph, which is of course not realistic.
- A dog lead is an external and dynamic constraint: the dog and its owner can move in an unpredictible manner, but are bound to stay within a fixed maximal distance.
- A non-collision constraint is an internal and dynamic constraint when the agent moves in a changing or unknown environment, for example if the labyrinth can be modified by the user at any time or if the agent does not *a priori* know about the maze geometry.

3. A language for behaviour description

The basis of our language is the **Timed Concurrent Constraint framework** proposed in[5], which provides simple but powerful building blocks for reactive programming. This extends the framework of Concurrent Constraint languages[4] with a notion of discrete time, adequate for most animation schemes (*e.g.* VRML).

Basic concurrent actions performed by the agents are either posting a constraint to a shared store (`Tell`), suspending until some constraint is entailed by the store (`Ask`), performing some method (`Call`), or posting an action to be performed at the next time-point (`Next`).

4. Examples

The first example shows two very simple but interesting behaviours. The cube can be dragged and dropped on the ground. The cone moves according to a specified trajectory and has a non-collision constraint applied to the cube: when the cube is placed on the cone trajectory, the cone avoids the cube by the left or the right depending on the shortest distance. In this example, the constraint is dynamic (the cone must react immediatly when the cube is moved) and is part of the cone behaviour (the cube does not care about the cone). The important point is that the collision solving mechanism makes the scene to be both interactive and very realistic.

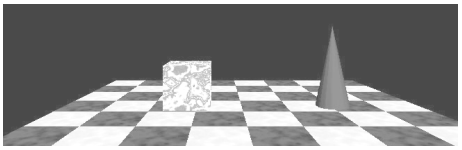

Figure 1: *The moving cone and a dragable cubic obstacle.*

In the second example, the user can move blocks in a room in order to create a labyrinth. The agent will have to find its way out by asking to its local constraint solver the right trajectory it will have to follow. We intend to implement this example in VRML to illustrate static internal constraints.

5. Conclusion

We have presented a possible integration of constraint programming for describing high-level agent behaviours living in virtual worlds. We are currently defining a generic model and implementing this framework in VRML97 in order to assess the expressive power of constraints for behaviour description. The constraints will be implemented using `Script` nodes in Java, making it easier to port to the Java3D environment if needed.

References

1. V. Saraswat, P. Van Hentenryck, P. Codognet et al., *Constraint Programming*, ACM Computing Surveys, vol. 28 no. 4, Dec 1996.
2. M. Gleicher, *Practical issues in Graphical Constraints*, In: proc. PPCP'94, 2nd Workshop on Principle and Practice of Constraint Programming, 1994.
3. E. Gobbetti and J-F. Balaguer, *An Integrated Environment to Visually Construct 3D animations*, In: proceedings of SIGGRAPH 95, ACM Press, 1995.
4. V. Saraswat, *Concurrent Constraint Programming*, MIT Press, 1993.
5. V. Saraswat, R. Jagadeesan and V. Gupta, *Timed Default Concurrent Constraint Programming*, Journal of Symbolic Computation, 22 (1996), pp. 475-520.