System Support for Scalable Distributed Virtual Worlds

Karl O'Connell and Vinny Cahill
Distributed Systems Group,
Department of Computer Science,
Trinity College Dublin,
Ireland
{koconnll, vjcahill}@dsg.cs.tcd.ie

ABSTRACT
Lack of bandwidth has been shown to be a major impediment to achieving realism in large scale virtual worlds with many interacting entities. Recent projects that have addressed this problem have, for the most part, been tied to a single application domain, typically the development of realistic military simulations. This paper presents a brief overview of the techniques to achieve scale adopted by the object execution environment of the VOID shell, a virtual world development toolkit.

Keywords
Distributed Virtual Worlds, Object-Orientation, Event-Based Communication, Scale.

INTRODUCTION
Although a number of projects have considered support for multi-user, distributed interactive virtual worlds (VWS), most of the resulting systems have been small scale (supporting a maximum of 100 simultaneous users). Those projects which have addressed scale have for the most part been tied to a single application domain usually the development of large scale realistic military simulations.

In current distributed vw applications, it is the available bandwidth that determines the size and richness of the vw. As the number of participants and entities within the vw increase so do the bandwidth requirements. SimNet and DIS [1] use predictive methods and dead reckoning to reduce communication between entities within the world. However both of these systems have problems when scaled due to their homogeneous world databases, where all object state changes must be communicated among all of the users of the environment. The NPSNET IV project [2] has attempted to overcome this problem by introducing the concept of Areas of Interest (AOI), whereby *only* vw entities within a particular area of interest would communicate and receive messages from each other. An IP Multicast address may then be assigned to the AOI. NPSNET IV, however is based on the use of the DIS network protocol which is targeted primarily at the development of large scale military simulations.

This paper describes the approach we have adopted in the Object Execution Environment (OEE) of the VOID shell [3], a toolkit for the development of scalable distributed VWS. The VOID Shell is being developed as part of the MOONLIGHT ESPRIT project and supports the development of VWS which are not tied to a particular application domain, e.g., video games, simulations, collaborative work and teleconferencing systems. The execution environment is based on the ECO model [4] which uses anonymous event-based communication. The OEE in addition to using events provides zones as mechanism for grouping VW entities according to their functionality and geographical location within the VW or physical location on the network. Through the use of zones, the VW application developer can control the number of event notifications propagated by the application. Objects can be defined programatically with a C++ based language, with extensions for events, constraints, objects and zones.

THE ECO MODEL
In the ECO [4] (Events, Constraints and Objects) model, objects which are instances of classes communicate using events. An event represents a change to the state of the system. Each event has a name and zero or more parameters. The parameters of an event are typed. For the specific occurrence of an event the parameters are instantiated with values. These values, together with the event name, describe the state change that has occurred.

An object can inform other objects about a state change, and it can react if it is informed of some state change by other objects. The former is accomplished by announcing an event, and the latter by binding a method of the object to the required event. This binding can be static (at object creation time) or dynamic. The same method can be bound to several events, and the same event can have several methods (of the same or of different objects) bound to it. A binding can be established only if the signatures of the event and of the method match (i.e have the same number of parameters, and the types of the corresponding parameters are the same). Thus events may be considered as a form of one-to-many anonymous communication.

Constraints

In addition to events, the object model supports constraints. Constraints are named conditions which control the propagation and handling of events. The motivation for constraints is threefold. Firstly, *notify constraints* provide a means of restricting the propagation of events to objects which are specifically interested in particular occurrences of those events, thus enabling a more efficient implementation of event-handling. Secondly, *real-time constraints* provide a mechanism for associating various real-time requirements with a class. Finally *synchronisation constraints* provide a mechanism for associating synchronisation policies with a class.

The aim is that event handlers should only be called when the object to which they belong is definitely interested in the specific event in question. Consider the common case of a Collision event whose parameters might include the identities of the objects that collided and to which many objects are likely to be subscribed. Many of them will be interested only in collisions between *themselves* and another object. Without notify constraints each collision event would be handled by every subscribed object, each of which would have to check whether the collision concerned it or not. The constraint system allows propagation of events only to those objects that want to receive them. In particular, to support the distributed case and to avoid the unnecessary transmission of event notifications, notify constraints may only refer to parameters of the event and never to the local state of an object. An object may decide, based on its local state when it is informed about an event of interest that the processing of the event should be postponed or even cancelled. If the local state is such that the processing should go ahead, it may be the case that multiple flows of control are allowed within the object.

SCOPING EVENTS

Notify constraints are however only one way in which event propagations may be restricted. Consider the following scenario of a vw containing student and supervisor entities. Let us suppose that the supervisor entity S_1 is in his/her virtual office with a particular student entity St_1 and three other student entities St_2, St_3 and St_4 are, for example, in a virtual research lab, another room within the VW. In this example when S_1 raises an event, say a supervisor-says event, they may wish that only the student in their virtual office receives notification of its occurrence. This situation may be resolved using notify constraints, however, it would involve each of the student entities keeping track of their location to determine if they wished to receive event notifications. Also for reasons of security it may be desirable that the supervisor entity has the ability to restrict the event propagations to a particular secure location i.e. their office.

We propose a model of scoping event propagations to overcome this problem. Objects in the model are organised into zones, where a zone is simply a set of objects and events are propagated to a particular zone. However, communication is still anonymous as the object raising the event need not know which objects are members of that zone. Scoping ensures that objects will not receive notification of an event's ocurrence even though they may be subscribed to that particular event type and satisfy all of the relevant notify constraints, unless the event has been specifically propagated to a zone of which they are a member. In the model, the scoping of the event propagations may be based at the discretion of the application programmer on the functional, geographical, or physical location of the object receiving the event.

Zones in eco may overlap allowing an object to be a member of two or more zones. This is useful in applications where an object may be a member of both a particular geographical and a functional zone simultaneously. Zones may also be nested, allowing large zones containing many objects to be subdivided. This caters for the situations in which an object in a vw receives for example audio events from objects in its vicinity (i.e. objects it can hear) but may also receive less frequent position update events from objects in an outer enclosing zone (i.e. not in hearing range but still visible). Objects may also change zones dynamically, giving the application developer the ability to create applications whereby entities may move from one section of the vw to another. Finally the scope of an event may be explicitly limited to a particular zone of which the object raising the event is not a member, allowing objects to communicate with other objects in a different zone.

SUMMARY

This paper has presented a brief overview of the techniques adopted by the OEE of the VOID shell toolkit to support the development of large scale VWS. At the current time, the design and specification of the ECO model along with constraints and zones is complete. A class library allowing C++ programs to use events and constraints is available while work is ongoing at present on the implementation of the OEE using ORBIX and ISIS.

ACKNOWLEDGEMENTS

This work is being developed as part of the moonlight project which is partially supported by the European Union under contract 8676. The authors would like to acknowledge the many helpful contributions from our project partners as well as Gradimir Starovic, Brendan Tangney, Stephen Collins and Tom Dinneen.

REFERENCES

1. Locke J. An Introduction to the Internet Networking Environment and SIMNET/DIS. Technical Report, Computer Science Department, Naval Postgraduate School, August 1993.

2. Macedonia M.R., Zyda M, Pratt D.R., and T Barham P.T. Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments. In *IEEE Virtual Reality Annual International Symposium (VRAIS'95)*, 1995.

3. Karl O'Connell, Vinny Cahill, Andrew Condon, Stephen McGerty, Gradimir Starovic and Brendan Tangney The VOID Shell: A Toolkit for the Development of Distributed Video Games and Virtual Worlds. In *First International Workshop on Simulation and Interaction in Virtual Environments (SIVE)*, 1995

4. Gradimir Starovic, Vinny Cahill and Brendan Tangney. An Event Based Object Model for Distributed Programming. In *Proceedings of the International Conference on Object Oriented Information Systems (OOIS'95)*, Dublin City University, Ireland, 1995.