

A Collaborative Access Model for Shared Virtual Environments

Steve Pettifer and James Marsh

Advanced Interfaces Group, The University of Manchester, Oxford Road
Manchester, M13 9PL+44(0)161 275 6259,{srp|marshj}@cs.man.ac.uk

Abstract

As shared virtual environments move beyond mere exemplars of computer graphics techniques, and evolve more meaningful content, issues such as ‘ownership’ or ‘access’ become important. Some artefacts in synthetic environments have ‘real world’ counterparts and require ‘real world’ access control and security. Others are entirely synthetic, and may require more esoteric access control. In either case, support for access model definition is becoming an important consideration for the future of virtual environments. In this paper we explore the different types of access required by different styles of VE, and develop a novel access model appropriate for these situations.

1: Introduction

As shared or Cooperative Virtual Environments (CVEs) become larger in scale and more sophisticated of content, issues of protection and security become increasingly important. If shared VEs are to be used widely for more realistic purposes than merely as high-technology novelty chat rooms or demonstrations of graphical rendering, issues such as object ownership and the management of resources by communities and individuals become significant. In this paper we discuss the role of access control in virtual environments and present a new mechanism and supporting architecture for allocating and managing the relationships between users and the contents of the VEs they inhabit.

Access control is a familiar concept in such fields as operating system design and Computer Supported Cooperative Working (CSCW). The term ‘access model’ refers to the set of mechanisms used by a system to determine the operations that may be carried out on a given object by a given user. For example, in a traditional computer file system the access model determines whether a user can read from or write to a file. Similarly in CSCW applications, access rights govern how users interact with and share resources. Such systems often rely on a built-in

set of rules or access patterns which determine whether a user, group of users, or user playing a particular role, can perform a specified action. Considering a general class of VEs (i.e. not just those acting as ‘front ends’ to file systems or CSCW applications) it is evident that problems of access management apply to a wide variety of issues ranging from low-level interaction with a synthetic object to higher level ownership rights. The general area of object access in VEs was identified by Snowdon *et al.* in 1996 [16] as being overlooked in the underlying VE systems; five years later the situation is mostly unchanged. Historically, the lack of access control in VEs may be explained by the fundamental technological challenges that must be met in order to generate even the most primitive three dimensional world; such higher level issues as access control have been of secondary importance. However as network and graphical hardware advance and behavioural sophistication in VEs improves, issues related to usability and reliability such as access control come to the fore. Our recent experience of implementing a number of collaborative virtual environments has highlighted this fact. In the following sections we discuss a range of VEs and examine the limitations of existing access models in the light of the requirements inherent in these environments.

2: A classification of access requirements in virtual environments

The term ‘virtual environment’ covers a wide range of applications that use a synthetic three-dimensional space as a metaphor for interaction (indeed, the term is also used to mean textual or 2D environments, though here we apply its meaning primarily to interactive 3D worlds). The applicability and appropriateness of access control varies between VE application types, which for this purpose can be considered under three broad categories:

- 1) **Facsimile Environments.** This category of VE aims primarily to reproduce factors of the real world in the virtual in order to facilitate training, learning (e.g. [2,17]) or visualisation of structure (e.g. [9]). Much of the behaviour and appearance of objects in such environments is governed by real world laws (e.g. use of radiosity or particle tracing to give realistic appearance, use of physical simulation to give realistic object dynamics).
- 2) **Abstract Environments.** In these environments (e.g. [3,10]) the three dimensional space is exploited in order to present information to the user. The layout, appearance and behaviour of such environments is governed by the semantics of the data rather than specifically by real-world factors (such as a 'groundplane'), though it is common to borrow some features from the real-world in order to aid comprehension and navigation of the layout.
- 3) **Social Environments.** In the previous two categories can be thought of as 'functional environments' which have a specific application-driven purpose and (most likely) well defined access requirements. This third category, whilst not being entirely orthogonal to the others, covers the general use of VEs for social and entertainment purposes (e.g. games such as Doom, Quake, Unreal, online environments such as AlphaWorld or Blaxxun, and more experimental virtual environments (e.g. [5,18]). The defining characteristic of such environments is that they are may borrow from any field with a view to making an environment 'conducive to social interaction' or 'fun to be in' etc. This means that that access requirements are open ended and determined by social convention as well as underlying application semantics.

Though these categories are by no means exhaustive or entirely independent of one another, they serve as a basis for discussing various classes of access requirements. For example, in the Facsimile and Abstract classes of environment—where in both cases a clearly defined underlying application is represented in a three-dimensional environment—the type of access will be determined by the application semantics (e.g. the environment represents some form of CSCW application with pre-defined roles and access rights [1]). In Social Environments, the access requirements will be more open ended. A number of examples based on recent and current shared virtual environments helps illustrate this:

DIVIPRO: an environment for Distributed Interactive Virtual PROtotyping of engineering models. In a shared virtual environment containing CAD models, in which an expert demonstrates a complex engineering assembly procedure to a trainee it may not be desirable for all users to have equal access to all objects at all times.

The expert may require more privileged access to the objects that form the assembly such that an uninterrupted demonstration can be performed before handing control to the trainee. Further, the expert may wish to be able to over-ride manipulations performed by the trainee, or recover the environment from mistakes. Such access issues have implications for both the usability of the application, and importantly have implications for the low level behaviour of the supporting VR system in terms of optimising and marshalling network resources.

QSPACE: a visualisation of abstract information [14]. This is an abstract space in which users create three-dimensional molecule-like structures representing queries of various database systems including libraries and on-line music and book shops [13]. The structures may be made available for browsing to other users of the system, and represent an expression of interest in a particular genre or subject. However the initiator of a query may not wish all such information to be made publicly visible, may wish to navigation of the structures to be limited to a certain set of colleagues, or may want to make the structures modifiable or replicable by users with certain characteristics.

The Senet Game [7]. This is an educational environment in which two young students learn to play an Egyptian board game by direct manipulation of the virtual pieces and communication via text entirely within a synthetic environment, and under the tuition and supervision of a teacher [6]. Access issues here involve turn taking amongst players, control of the board by the teacher, and importantly the ability of the teacher to maintain an overview and to control the actions of the students in order to regulate unruly or disruptive behaviour.

PLACEWORLD [4]. A shared social environment constructed as a multi-media art installation. Multiple users interact with one another as well as a multitude of embedded worlds and artefacts. Users are able to exploit 'generator' objects to create content for the world, and to create their own 'home' environments populated with artefacts. 'Hyperlinks' can be made from environment to environment to enable easy navigation to favourite locations. In this environment where no underlying 'application' is present, and where the resources content of the worlds are entirely 'virtual' with no real-world counterpart (or, for example, financial value) issues such as ownership and control are more open ended, and yet require some support from a system level in order to maintain structure in the environment over time.

3: System support for access control

The most common example of system support for an access model in computing is that provided by the

majority of mature operating systems to manage files. These techniques can usually be reduced to a three dimensional matrix with users, files (more generally 'resources') and 'operations' (such as read, write, execute) along the axes, with entries in the matrix representing 'grant' or 'deny'. Two-dimensional interpretations of this 3D matrix are called variously 'access lists' or 'capability lists'. More sophisticated models have been developed (for CSCW, mostly e.g. [9, 12]) which include the notion of a 'role' played by a user (or number of users) [6,13], and which allow for some migration/extension of access rights through mechanisms such as inheritance. Whilst these models are appropriate for the domain in which they were developed, they are too restricted to be able to deal with the types of access issues indicated above. In particular, they centre on a set of defined resources and operations. Indeed, it would appear that a single pre-defined access model that covers all the eventualities described above would be a complex and unwieldy structure. Nevertheless, some degree of system support for access control is required for the following reasons:

- 1) To define a framework within which an appropriate and coherent type of access control may be constructed by the application programmer
- 2) To provide an interface such that the users of the system can interact with and understand the behaviour of the access control system
- 3) To enable a degree of coherency/brokering between applications such that information/artefacts from one application may be transferred to another.

An additional benefit of having *system* support for access control is that, as indicated in example 1 above, it provides higher level information to the lower level system functions in order to enable optimal use of system resources. For example, providing meaningful real-time manipulation of a single object between two geographically distant users is a significant research challenge given the limitations of today's network technology, and solutions will inevitably require significant usage of both network and compute facilities; knowing 'in advance' that control of single 'shared' object is in fact passed back and forth between individual users allows the system to focus its network resources on those objects that might be genuinely manipulated simultaneously.

Our research proposal therefore concentrates on the development of an *extensible* access framework embedded within and supported by the fundamental architecture of Deva [15], a distributed virtual reality system developed by our research group. Deva's architecture represents a number of novel mechanisms for emphasising enabling the provision of *behaviour* for VE applications, and for that behaviour to be distributed to

remote users in order that they maintain a coherent and shared experience of the environment they inhabit. Being explicitly aware of the behaviour of objects in its environments, the Deva programming model is well suited to the development of an access control framework that interacts with that behaviour.

4: Scope of the model: access versus security

Though they are related concepts, the distinction between an 'access model' and a 'security model' is an important one. The former is first a policy decision, and second an implementation by the system of that policy that enables the moderation of the relationship between a user and the other objects in a system. The latter consists of algorithms that enable the access model to retain its integrity under malicious attack.

Access models focus on the ways in which the availability of resources is managed. In principle, an access model could be a 'gentleman's agreement' between users (though in practice it would be unworkable due to the large number of objects involved if for no other reason). While such models can be used to ensure the secrecy of resources they can also promote openness. The access model is the means by which the collective decisions of the nature of the environment are expressed. The 'security model' is what enforces the access model (e.g. [8]). Computer security, with all its associated encryption algorithms, political implications and complex mathematics is a vast topic that we do not intend to consider in depth here. We concentrate instead on describing a novel access model and take into account the issue of security only in as much as it impinges on a prototype implementation of our scheme.

5: The collaborative access model

Environments of type Facsimile and Abstract will tend to have access issues defined by their application; though this requires some support from the underlying system to implement it, the definition of access requirements at least will be clear. For social environments the definition is more open ended and evolutionary. The model we propose therefore aims to provide a mechanism whereby inhabitants of a social virtual environment may describe how they wish entities that they create/own to be related to by other inhabitants. By catering for this most demanding category, we aim to fulfil the requirements of the other two classes in the process. We wish to avoid prescribing the types of operation that may be permitted, and aim to provide a model that has a more eloquent response to user

interaction than the normal 'grant' or 'deny'. The mechanism attempts to provide a framework within which the inhabitants themselves may develop appropriate access policies. With the vision of very large numbers of users inhabiting extensive virtual environments, the idea of 'super user', 'group' and 'user' seems too impoverished, and imposes an undesirable and forceful hierarchy that is unlikely to have a meaningful correspondence in terms of the social layout of the space as it evolves. Thus we aim to provide a collaborative access model that allows evolution of policy along with content.

5.1: Details of the model

In this section we describe the high-level concepts of the collaborative access model. It is implemented using the Deva entity/component architecture, which allows runtime creation and modification of objects and their state and function [15]. For this model we implement the following main concepts:

- **Entity:** a general resource within the system. An entity may be graphically represented (such as a 'a house' or a 'a teapot') or may be an abstract concept with no visible representation (such as 'a group'). Entities contain an amount of 'state' and provide a number of 'methods' that alter that state. They are in most respects 'objects' in the 'object orientated' sense except that their overall functionality is amenable to introspection and modification during their lifetime.
- **User:** a particular kind of entity that is related to a person in the real world
- **Key:** an entity that describes a user's *dynamic* relationship with a particular method/function of a named entity

There are two types of access in such a system: 'class based' and 'object based'. The former is a programmatic or application developer issue, the latter more important for the users of the environment (though in Deva, classes are essentially objects and thus the two issues merge at a practical level).

5.2: Class based access

This type of access deals with the integrity and structure of the underlying programs/objects that form the VE. Traditional object orientated languages/systems (e.g. C++, Java, SmallTalk, Eiffel) provide for a degree of access control between classes of objects. Thus, in C++ for

example, it is possible to declare at compile time the methods and state of a class that maybe accessed by 'objects of the same class', 'objects of a specific class' and 'any object'. In C++ this is a rather crude mechanism based on 3 levels of privacy ('private to me only', 'private to me and anything derived from me', and 'available to all') with finer grained access made possible by declaring exceptions to these rules. The other languages provide finer grain and perhaps more elegant solutions to this problem. However all deal with access at what is essentially a compile-time level and only have a notion of 'class' access rather than ownership or use of the object at runtime. Thus it is either valid, or a compile-time error for Class A to attempt to access method B of Class C. This type of access is a fundamental requirement of any modern programming environment.

5.3: Object based access

Object based access refers to the relationships between the affordances of objects and their users. For example, the availability of an object to be 'owned' or 'moved' or 'used' in some way. This is the type of access highlighted by our exemplar environments.

5.4: The mechanisms of the model

In our model, processing is achieved by entities communicating/interacting by calling methods upon one another. The access model is driven through this mechanism. That is, at run-time, whenever an entity attempts to perform an operation on another entity, the basic mechanism that makes the interaction possible intervenes and validates that operation before allowing it to continue. This is akin to every function call in C being intercepted and validated by some mechanism, though in Deva is performed using a lightweight message passing mechanism to enable entities on different processors to communicate efficiently. The entities themselves are coarse grain enough, and the implementation of the access model lightweight enough for this not to be a performance issue.

The Deva programming model enables entities to combine functionality from a number of sources. Some functionality, called *innate functions*, are defined particularly for an individual entity. Other functionality is *imbued* upon an entity by the environment in which it is created. Such functions may be 'enforced' by an environment (thus being a 'law of the world') whilst others are optional (being more the nature of conveniences or defaults). For example, an environment in which the

laws of Newtonian mechanics are present may enforce the notion of mass upon all its contents, and may provide default methods for approximating the mass of an object that does not inherently have such a property; more abstractly an environment could require that all its contents have a method allowing them to be rendered in a particular style. Our access model enforces a notion of 'authentication', a 'law' that checks that the appropriate conditions hold before allowing any operation to continue. An overview of the operation of the model is provided in the following sections.

5.5: Overview of operation

All operations that are performed within environments using the access model are moderated by functions provided by that environment. Operations are allowed to continue under one of two circumstances:

- 1) the object itself allows the operation to succeed, or
- 2) the originator of the operation has a valid key for that operation.

A key is an object managed by the environment that has two main properties: first a 'signature' identifying which object (or user) originator of the operation, the object upon which the operation was requested and the details of the request itself; and second a 'payload' or general piece of code that describes under which circumstances they key is valid for use. The payload code takes the same form as the code used to define the objects themselves, and may make any query or interact with other objects in the environment. Thus keys may be defined for such access as:

- "Users can leave vehicles in this area as long as it is between 6pm and 11pm"
- "Only members of this select group can collect mail from this box"
- "Everyone can enter here as long as they have visited the town hall at least once"

Each user is provided with a key manager component in their user interface to the environment. This maintains a list of keys owned by that user, as well as 'requests' for validation of keys by other users. Thus 'interactive' conditions can be placed in the payloads of keys enabling intervention by other users. For example:

- "This operation may continue if the owner of this object agrees"
- "You can enter here if 50% of the group members allow it"

When a user is presented with a request to validate a key, they respond by *returning a new key payload*; that is another fragment of code. Often this will be a small package of code that is interpreted simply as 'yes' or 'no', and such common responses are pre packaged by the system and made available through simple graphical user interface operations. Importantly, however the user may choose to respond to a request *with a more complex payload*. In essence this is deferring their response to others or placing particular conditions in their response. This allows access requirements to be propagated throughout the system, creating complex compound rules that are supported by the underlying system. For example code may be returned indicating such intentions as:

- "I will agree to this after 6pm"
- "I agree, but only this once"
- "I agree to this only if all of Users A, B and C do too" (thus requesting a poll of the other users, who may themselves return conditions)

As well as providing simple 'yes' and 'no' responses, the graphical user interface provides a simple means of configuring the more common types of complex response. If the user's desired response is not provided by any of the pro-forma ones in the interface, an editor enables them to hand-code it. The user's interface can also be configured to respond automatically to certain classes of key request using pattern matching and other rules. For example:

- "Always say yes immediately to requests originating from users in the same class as me"
- "Silently ignore any requests from this user"

6: Current State and Future Work

The Collaborative Access Model we have presented currently exists in an early stage of implementation, including the low-level infrastructure coded in terms of laws within the Deva environment, and a prototype graphical user interface enabling creation and management of keys. The PlaceWorld environment, containing a number of simultaneously active social worlds and content for users to explore is under continued development, with the intention of making a public release of both the Deva system and this specific environment in Q1 2002. This environment will be used as a testbed to enable large-scale validation of the mechanisms of the access model. In the meantime work continues on the development of the underlying system, and a formal grammar for specifying the behaviour of entities and keys.

Further information about this work can be found at <http://aig.cs.man.ac.uk>

7: References

1. Benford S., Bowers J., Fahlen L.E., Greenhalgh C., Mariani J. & Rodden T. (1995). 'Networked virtual reality and cooperative work'. Presence, Teleoperators, and Virtual Environments, 4, 364-386
2. Boud A C, Baber C, Steiner S J, 'Virtual Reality: A tool for assembly?' Presence Teleoperators and Virtual Environments, Volume 9:5 MIT Press Oct 2000, pp486-496
3. Chalmers, M. 'BEAD: Explorations in information visualization'. Proceedings of SIGIR 92(Copenhagen, Denmark, June 1992). ACM Press, pp 330-337.
4. Cook J, Pettifer S, 'PlaceWorld: An Integration of Shared Virtual Worlds', accepted for publication, SIGGRAPH 2001 Sketches and Applications Program, Los Angeles, August 2001
5. Crabtree A., Pettifer S., Rouncefield M., 'Social Interaction in Virtual Environments', Proc. Laval Virtual 2000, Laval, France, 18-21st May 2000
6. David Ferraiolo and Richard Kuhn. 'Role-based access controls'. In 15th NIST-NCSC National Computer Security Conference, pages 554--563, Baltimore, MD, October 13-16 1992
7. Economou D., Mitchell B., Pettifer S., West A., 'CVE Technology Development Based on Real World Application and User Needs', in proc. IEEE Knowledge Media Networking, National Institute of Standards and Technology, Washington June 14th-16th 2000
8. G. Coulouris and J. Dollimore. 'Requirements for security in cooperative work: two case studies'. Technical Report 671, Dept. of Computer Science, Queen Mary and Westfield College, University of London, 1994
9. G. Smith and T. Rodden. 'SOL: A Shared Object Toolkit for Cooperative Interfaces'. International Journal of Human-Computer Studies, 42(2):207--234, February 1995.
10. Gibson S., Howard T., 'Interactive Reconstruction of Virtual Environments from Photographs, with Application to Scene-of-Crime Analysis', Proc. ACM Symposium in Virtual Reality Software and Technology 2000 (VRST'00), Seoul, Korea, October 2000
11. Hendley, RJ, Drew, NS, Wood AM and Beale, R (1995). 'Narcissus: Visualising information. Proceedings of Information Visualization 95 Symposium (Atlanta, USA, October 1995). IEEE, pp 90-96
12. Honghai Shen and Prasun Dewan, 'Access Control for Collaborative Environments,' Proceedings of the ACM Conference on Computer Supported Cooperative Work, November 1992, pp. 51-58
13. Luigi Guiri. 'A new model for role-based access control'. In Proc. of 11th Annual Computer Security Application Conference, pages 249-- 255, New Orleans, LA, December 11-15 1995
14. Pettifer S, Cook J, Mariani J, 'Q-space: a Virtual Environment for Interactive Abstract Data Visualization', in proc. International Conference on Virtual Reality 2001, Laval, France, May 16-18, 2001
15. Pettifer S., Cook J., Marsh J., West A., 'DEVA3: Architecture for a Large Scale Virtual Reality System', in proc. ACM Symposium on Virtual Reality Software and Technology, VRST2000, Seoul, Korea, October 2000
16. Snowdon, D. N., Greenhalgh, C. M., Benford, S. D., Bullock, A. N. and Brown, C. C., 'A Review of Distributed Architectures for Networked Virtual Reality', Virtual Reality: Research, Development and Application, 2, (1), pp 155-175, 1996, Virtual Press
17. Tendick F, Downes M, Tolga G, Cavusoglu M, Feygin D, Wu X, Eyal R, Hegarty M, Way L W, 'A Virtual Environment Testbed for Training Laparoscopic Surgical Skills', Presence, Vol 9:3, July 2000, MIT Press, pp 236-255
18. Waters R, Anderson D B, Barrus J W, Brogan D C, Casey M A, McKeown S G, Nitta T, Sterns I B, Yerazunis W S, 'Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability' tech. report 96-02, MERL, Cambridge, Mass., Jan. 1996.