Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring

Chris Greenhalgh and Jim Purbrick,
School of Computer Science and IT,
University of Nottingham, UK.
Email: {cmg, jcp}@cs.nott.ac.uk

Dave Snowdon,
Xerox Research Centre Europe,
Meylan, FRANCE
Email: Dave.Snowdon@xrce.xerox.com

Abstract

MASSIVE-3 is our third generation of Collaborative Virtual Environment (CVE) system. This paper describes the goals, design and implementation of key aspects of the MASSIVE-3 system, and in particular its support for data consistency, and world structuring and interest management. MASSIVE-3 adopts a distributed database model, in which all changes to items in the database are represented by explicit events that are themselves visible to the system. Networking is logically multicast, but physically client-server (the reasons for this are explained). MASSIVE-3 makes application behaviours explicitly visible within the database in the form of "Behaviour" data items. MASSIVE-3 implements and extends work on consistency by the University of Reading. In particular, it adds an explicit "Update Request" data item, which allows the system to support a number of different consistency mechanisms within a single virtual world. World structuring in MASSIVE-3 extends the notion of "Locales" from the SPLINE system to include distinctions based on functional class, organisational scope and fidelity. It also allows flexible and general replication and rendering policies to be specified and used for interest management.

1 INTRODUCTION

MASSIVE-3 is the third generation of Collaborative Virtual Environment (CVE) system created by the Communications Research Group at the University of Nottingham. Its origin is in the EPSRC-funded HIVE (Huge Interaction Virtual Environments) project, which brought together work on consistency in CVEs from the University of Reading [1], with work on interaction and awareness management from the University of Nottingham (e.g. [2]). This paper describes the goals, design and implementation of key aspects of the MASSIVE-3 system, and in particular its support for data consistency and world structuring.

Each generation of MASSIVE has had significantly different goals and a correspondingly different design. MASSIVE-1 was a direct and complete implementation of the spatial model of interaction [3] as it existed at the time (1994). Its application domain was tele-conferencing and small-group meeting support. It was implementation using unicast peer-to-peer data distribution between user applications, with a single server process performing initial introductions between applications (the "spatial trader" or "aura manager"). This distribution approach was a direct and literal implementation of the spatial model of interaction. By analysing the network bandwidth requirements of MASSIVE-1 we found that the system would not be able to support more than twenty or so mutually aware users on an Ethernet LAN (we had only 10 Mbit/s Ethernet at the time) because of the $O(N^2)$ bandwidth requirements of a unicast peer-to-peer approach (especially with network audio) [3].

Consequently, MASSIVE-2 was designed from the outset to use network-support multicast communication wherever possible (specifically IP multicast) [4]. It also implemented an extended version of the spatial model of interaction, which includes the notion of "third-party objects". MASSIVE-2 also had a more sophisticated interaction model than MASSIVE-1, which allowed users to manipulate objects within the virtual world. In normal use we found that MASSIVE-2 would support up to about 20 mutually-aware users (with audio) on the workstations we had (SG O2 w. R5000 processor). This limitation was due to available CPU throughput, rather than bandwidth (at least on a LAN or high-speed WAN). MASSIVE-2 was also hard to extend or modify. For example, there was also no obvious way to extend it to support access over lower-bandwidth networks (e.g. dial-up modem connections).

Based on these experiences, and others, we set ourselves the following goals for MASSIVE-3:

- It should support hierarchically structured virtual objects. MASSIVE-1 had no support for sub-objects, and MASSIVE-2 had only limited support; this was because of unresolved issues in reasoning about compound objects in the spatial model of interaction.
- It should implement proposals for data consistency and ameliorating the effects of network delay from the University of Reading [1].

- It should provide a route to support heterogeneous computers and networks, including domestic users with modem-based connections. This would require flow and congestion control at the network level, and ideally additional support for self-monitoring and control (i.e. for Quality of Service considerations).
- It should have a capable and flexible mechanism for interest management (i.e. for structuring virtual worlds and the interaction that occurs within them).
- It should support persistent virtual worlds, i.e. virtual worlds that outlive a single creating application.

We believe that the system has met these goals, as described in this paper.

This paper is structured as follows. Section 2 overviews the data model and distribution architecture adopted for MASSIVE-3. Section 3 then describes more of the realisation and main capabilities of MASSIVE-3's distributed database. Section 4 focuses on support for data consistency, in particular the different forms of consistency that can be combined in a single MASSIVE-3 session. Section 5 focuses on world structuring and interest management in MASSIVE-3. Finally, section 6 presents conclusions and identifies areas for future work.

2 ARCHITECTURE OVERVIEW

At the heart of MASSIVE-3 is a distributed database, which allows applications to share descriptions of objects and activities within a 3D audio-graphical virtual world. Each distributed database (called an "Environment") describes one well-defined portion of a virtual world (the exact relationship of environments to virtual worlds is described in section 4). Each database can contain any number of hierarchically structured data items (the environment itself is the single root of this hierarchy). This is essentially a shared scene graph that includes additional semantic information and hooks to application-specific behaviours (see section 3).

Each environment has both a name and a unique system identifier; these never change. In addition, it has a low-level address that is used to access the environment. This comprises an IP address and a port number, and may change over time. A specialized MASSIVE-3 name-server (the "Trader") translates environment names to IDs and IDs to network addresses.

Each environment database is fully replicated, i.e. every application which is interested in a particular environment obtains and maintains a complete copy of it. Currently, every joining application contacts a single master process for a given environment. However it would be relatively simple to extend the system to allow a joining application to contact one of a number of servers (or any other application which has already joined the environment and is prepared to serve in this way).

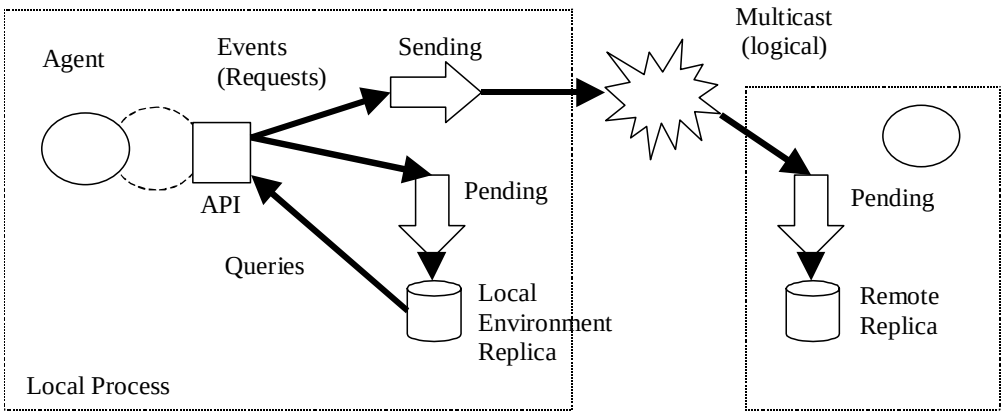


Figure 1: Agents, Environment replicas and messages in HIVEK

Any application which has joined an environment can add new items to it, and change or delete current items (subject to ownership constraints, described in section 4). Every change to an environment is represented within the system by an event object. This event object fully describes the proposed change (e.g. the identity of the item, its new value, any other changes that it depends on). Normal applications never change their local database information directly; instead, every change is made by creating a suitable event within the system. This event can then flow through the system, both updating the local database, and being sent over the network to update remote copies of the database. This is illustrated in figure 1. Note that each application is represented within the system by an Agent object. Also note that events are queued and managed in sending and pending queues (for events which are out-bound to other applications, and which are due to update the local environment copy, respectively).

As indicated in figure 1, communication between copies of a single environment is logically multicast, i.e. each event will be received by every other application that is currently maintaining a copy of that environment. However, this is actually realised as a client-server system, using unicast network protocols (TCP/IP). The master process for each environment also acts as a

network server, forwarding event objects to all processes currently replicating the environment.

The reasons for adopting this approach rather than a native multicast approach are as follows:

- TCP has established flow and congestion control capabilities; this is still an active research area for multi-sender multicast.
- It provides a direct route to support variations between network connections: if a connection is too slow to support the rate at which events are offered to it then the corresponding sending event queue will grow. This can be used as a trigger for application-level adaptation, e.g. to begin to summarise outgoing events, or otherwise reduce the amount of communication required.
- It will work over any part of the Internet, whereas multicast availability remains partial and variable in quality and bandwidth.
- Using multicast should give lower bandwidth utilisation over shared networks. However this is irrelevant over single-user dial-up connections, and core network bandwidth is not normally the first bottleneck for our current applications (especially with the widespread availability of 100 Mbit/s LANs).
- If a single server becomes a bottleneck in itself then a tree of servers could be used to reduce the required fan-out. Remember also that a single server handles only one part of an entire virtual world.

3 ENVIRONMENTS

The previous section gave a broad overview of the approach to distribution adopted in MASSIVE-3. This section provides some additional details regarding the capabilities and implementation of an environment.

3.1 Data Types and Behaviours

Each data item can be one of several standard types, including:

- A 3D homogeneous matrix transformation (which can describe any combination of rotations, shears, scales and translations).
- A 3D geometry, currently identified by filename or URL.
- A textual name-value pair or "attribute".
- A behaviour, i.e. a piece of application-specific code to be executed in this context.

Attributes are used to add semantic information to the shared database. For example, attributes identify the data items that comprise a user's embodiment, and are also used to communicate audio and video session information for in-world audio and video support.

Behaviours in the database currently comprise a name that identifies an embedded class to be instantiated, plus a list of argument values. In the future we plan to use URNs and/or URLs to identify behaviours, and support dynamic loading of code classes. In the previous MASSIVE systems code existed only within the particular applications accessing a virtual world, and was therefore effectively invisible within the system (other than through its direct effects). Giving behaviours an explicit representation within the shared database has two key benefits:

- It can trigger the automatic instantiation of behaviours in remote processes. For example, the current system uses a remote interpolation behaviour to make user movement appear smooth, even if the rate at which updates is sent is limited. Remote behaviours can also be used to reduce network communication requirements, since a single high-level event can trigger a complex behaviour in each application.
- An environment can be stored (e.g. by being written to disk) and have its behaviours restored when it is recalled, since they are explicitly identified within the recorded database. This is required to support the persistence of environments that include behaviours.

The DIVE system [5] allows behaviours to be specified using TCL scripts associated with virtual objects; these have the same potential benefits (MASSIVE-3 currently uses only C++ for behaviours).

3.2 Local state

Normally, all items and events within an environment are fully replicated, and visible to every application that has joined the environment. However, MASSIVE-3 also allows items and events to be designated as "local only". These items and events will not be communicated over the network, and so will only be visible within the application which originates them. Local items can be introduced anywhere in an environment, e.g. a local sub-tree of items could be used to give a personal "head-up-display" annotation attached to global items. The only restrictions are that global items cannot be children of local items (since the remote replicas would have no known parent) and events always have the same scope as the items to which they apply (otherwise the global consistency mechanisms would be confused).

3.3 Spooling

When an application wishes to join an existing environment it has to contact the master process for that environment and obtain an initial snap-shot of the state of the environment. MASSIVE-3 allows this contact and state transfer to be handled by a background thread, while the main application continues to run using the environments which it has already joined. This is referred to background spooling of the environment state.

The actual spooling of the state is achieved as follows. When the master application receives the request to join, it traverses the environment database and creates a

linked list data structure pointing to all of the items in the database at the moment of joining (including their values at that moment). This linked list is logically prepended to the queue of events to be sent to the joining application. Consequently, the first events which the joining application receives are “add item” events, which cause it to create copies of the items which were already present within the environment when it joined. Subsequent updates to the environment are queued up behind these initial events, and will be delivered and enacted once the initial transfer is complete. Reference counting is used on the data items to avoid copying the complete database for sending to each new joiner.

3.4 Record and replay

We have already observed that it is possible to write the complete description of an environment to storage (e.g. to disk), and to retrieve it at a later time to recreate that environment. It is also possible to record each and every event as it passes through the system. These recorded events can then be used to reconstruct the activity within the environment at a later time. MASSIVE-3 allows recordings of past activity in an environment to be integrated into a current world using a "temporal link". A temporal link defines the relationship between the current virtual world and the recorded environment, e.g. at what speed the recording is replayed, from which point, and how it appears in relation to the current world (e.g. size, position, colour). This capability is described more fully in [6].

4 CONSISTENCY

The previous sections have described the general structure and capability of MASSIVE-3. This section describes its support for a number of different forms of consistency for data items within the database.

4.1 Basic Consistency

We begin by describing the basic consistency model adopted in MASSIVE-3, before describing our extensions. The basic consistency model is a direct implementation of proposals from the University of Reading designed to support responsive interaction with objects in a virtual world in the presence of network delays. These techniques are most fully described in [1]. The consistency model has the following defining features:

- At any given moment each data item has a single designated owner. Only the owner is allowed to update or delete the item.
- Each data item has a two part sequencer (counter). This defines an unambiguous order for all updates to that item. The first part of the sequencer is incremented (and the second part reset) whenever an update is made which must be reliably delivered to all other processes. Only the second part of the sequencer is incremented when an update is made which need not be reliably delivered (e.g. one of a sequence of position update messages).
- Every message includes a list of item sequencer values that must be reached before the event can be enacted (i.e. actually update the database). It also indicates how enactment of the event updates the corresponding sequencers. Note that these sequencer values can be defined by the application to suit its own requirements.
- An ownership request event can be used by any application to request ownership of a given data item. The current owner may refuse the request, or may acquiesce, transferring ownership to the requesting application. The ownership transfer increments the item's reliable sequencer, ensuring that item ownership is also unambiguously ordered.

This is a conservative consistency scheme, sometimes identified as "singe transferable ownership", with the additional distinction between reliable and unreliable updates. A typical update cycle is illustrated in figure 3 (a) and (b).

As well as sequencer values, each event can have a “soonest enactment time”. This allows an application to introduce an event into the system before it is due to occur (provided that the application can determine in advance that the event definitely will occur). The event can then be distributed, and subsequently enacted in every process at the same time.

Reading's work also proposes some strategies for determining when a process should ask for ownership of an item. It is possible to use these strategies in MASSIVE-3, although they are not used automatically.

4.2 Update Requests

In order to support a range of consistency schemes we have added a new database item type called an "Update Request". By adding an update request item to an environment, an application indicates to all observing processes that it wished to make a particular update but was not able to do so (or chose not to). The update request includes the value to which the item would have been changed. This simple addition admits a much more flexible approach to consistency, as will be explained in the following sections.

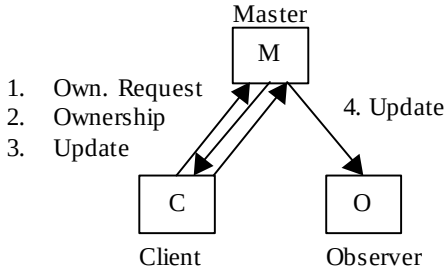
4.3 Centralised Updates

By default an application which owns an item but which does not mind giving away ownership will also monitor for update requests on that item. When it observes a new (or updated) request it will actually perform the corresponding update. Thus, the owner of the item automatically serialises the update requests (irrespective of which processes they come from). This approach to consistency is a form of total ordering (via a single

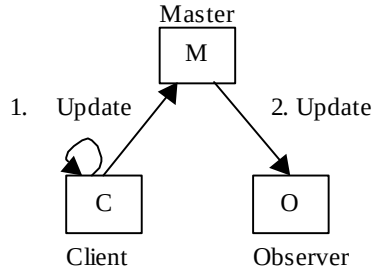
sequencing process per item). This is illustrated in figure 3 (c).

This approach has two potential advantages over ownership transfer. Firstly, the first update to an item will be observed more quickly than with ownership transfer since the request travels to the owner where it is immediately enacted and distributed to all processes (two network trips). Compare this with ownership transfer, where the first update takes three or four network trips before it is observed (as in figure 3 (a)). Secondly, any process can generate updates at any time and they can have an effect. With ownership transfer, only the current owner can generate updates, and no process can generate updates while ownership is being transferred.

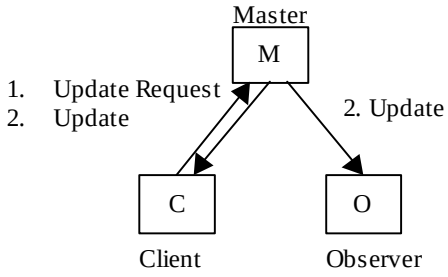
The critical disadvantage of centralised updates in many contexts is that each and every update goes through the owning process, and so there will always be a delay between a process generating an update request and observing the change (assuming that it is not the owner). With ownership transfer, once a process has ownership it can generate updates and be certain that no conflicts can occur. Consequently they can be immediately enacted at the generating process. This allows more responsive user interaction once ownership has been obtained.



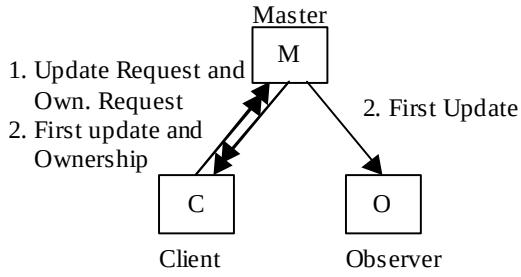
(a) Ownership transfer (first update).



(b) Ownership transfer (subsequent update).



(c) Centralised update.



(d) CIAO.

Figure 3. Consistency mechanisms in MASSIVE-3.

4.4 CIAO

As described above, a shortcoming of the single-transferable ownership approach is that the first update to an item (e.g. a user picking up a virtual object) must be delayed while the user's application requests and obtains ownership of the item to be moved. In Reading's approach, various heuristics are used to anticipate the need for object ownership, and to request it in advance of the actual need.

The CIAO system [7] uses a different approach: the initial update and the request for ownership are combined into single message. When the current owner receives this message it can simultaneously perform the first update and transfer ownership to the requesting process. This allows the first update to be performed and observed sooner than would be possible with ownership transfer alone. This is illustrated in figure 3 (d).

This approach is effectively a combination of centralised update (for the first update) and ownership transfer (for subsequent updates). It can easily be realised in MASSIVE-3 by an application creating an update request corresponding to its first update, and then immediately following this with a request for ownership of the item. When the update request arrives at the current owner it will be enacted (assuming that the owner is prepared to do this), making the first change immediately observable. This is immediately followed by the ownership request, causing ownership to be transferred to the requesting application.

Note that MASSIVE-3 allows different consistency strategies to be used on different items in the same database; all that is required is that processes are consistent in their use of ownership transfers, updates and update requests on a given item. This might be achieved by hint attributes in the database, or by other cooperation between the different applications (e.g. through use of the same distributed behaviour code). For example, the HIVE

project demonstration client allows the user to toggle manually between ownership transfer, centralised update and CIAO-style update for user manipulation of virtual objects.

4.5 Constraints and Cooperative Manipulation

Update requests can also be used to realise constrained or cooperative updates to data items. The system allows an owning application to mark a data item with a "control" lock. This indicates that no other process may take ownership, but the owner may respond to update requests. Essentially, the owner wishes to retain ultimate control over the item, but may be prepared to make some updates.

For example, an application which is implementing a 3D slider widget may wish to ensure that the slider always stays in a valid position. The application cannot therefore give ownership to another (untrusted) application, since the other application could do absolutely anything with the slider. If, however, the controlling application makes the slider control-locked, then other applications will only be able to request that the slider be moved. The controlling application can examine the requests, and determine for itself how the widget should move. The pattern of communication is as in figure 3 (c) (centralised updates).

In a similar way, consider the example of several users attempting to manipulate the slider at the same time. With a control lock, each user application will be generating its own update request. The single controlling application can examine all of these requests together and determine a composite response (if it wishes to). The only disadvantage of this approach is the reduced responsiveness (for the one application that might otherwise have been the owner), since this has now become a modified centralised update scheme.

5 WORLD STRUCTURING AND INTEREST MANAGEMENT

In all of the MASSIVE systems, interest management (or "awareness") has been a primary concern. The essence of most interest management techniques is to divide a virtual world and its contents into appropriately sized "chunks" of information and communication, and to allow each application to deal with only those chunks which are directly relevant to it.

In MASSIVE-1 the unit of interest management (the "chunk" size, or granularity) was one medium of one object (e.g. the graphical appearance of a single object). For most applications this is a little too fine, resulting in significant management overheads, especially in worlds containing many objects (e.g. visualisation applications). In MASSIVE-2 the unit of interest management was one medium of one spatially-defined region (associated with a third party object, which is a part of the spatial model of interaction). For MASSIVE-3 we wanted to find a more flexible approach which would not be limited to spatial organisation, and which would not have excessive overheads. We also wanted a more explicit notion of what items were associated with what units of replication, to support system introspection and self-management.

To these ends MASSIVE-3 builds upon and integrates several elements of the state of the art in interest management, including locales, functional and organisational distinctions, and abstractions. These are considered in turn in the following subsections.

5.1 Locales

In MASSIVE-3 we have adopted (and extended) the concept of "Locales", introduced in the SPLINE system [8], to provide a powerful and expressive means of structuring and composing a virtual world. The key concepts of locales are reviewed below.

- In a locale-based system the locale is the fundamental unit of world composition. Each locale typically corresponds to a distinct region of the virtual world, such as a room, corridor, open space or vehicle, and can contain virtual objects as well as users' embodiments (their representations with the virtual world).
- A key feature of the locale approach is that there is no single global coordinate system for the whole virtual world. Instead, each locale defines its own independent coordinate system.
- A complete virtual world is composed by linking together a number of locales. Each link from one locale to another includes a 3D transformation which defines the relationship between the locales' coordinate systems. Note that these links and transformations can be asymmetric.
- These transformations can be used to construct non-Euclidean worlds as demonstrated in Diamond Park [8] (e.g. a building that is larger on the inside than on the outside).

All MASSIVE-3 virtual worlds are composed from one or more locales, with inter-locale links (or "boundaries") defining the relationships between locales, e.g. their relative sizes and positions.

5.2 Aspects

Locales are an effective means of subdividing a virtual world according to spatial characteristics (e.g. one locale might correspond to a particular room). In MASSIVE-3 we further sub-divide a locale into one or more "Aspects". These are the fundamental unit of interest management in MASSIVE-3, and each aspect is realised as one environment database (as described in the previous sections). All distributed data in MASSIVE-3 must be contained in a particular aspect of a particular locale. An aspect is defined by the following properties (expanded below):

Functional class(es)	Type or class of content, e.g. EMBODIMENT, TERRAIN, BASE
Organisational scope(s)	E.g. PLATOON-A, BATTALION-3, TEAM-A, …
Fidelity	"Quality" of this aspect, e.g. resolution, accuracy, …
Cost	Cost of the aspect, e.g. some function of state size, bandwidth, rendering,…

The use of functional classes follows [9]. Typically, different functional classes of data (e.g. different media) will only be of interest to a subset of observers, and this can be exploited if those functional classes are separated within the interest management framework. A single aspect can contain one or more functional classes. We define a specialised functional class called "BASE" which must exist for every locale. The base aspect contains all of the links to neighbouring locales and all of the links to the other aspects of that locale. This allows the system to learn about locales and world topology by replicating only base aspects, which will normally be relatively compact and static.

Following [10] we add the notion of organisational scopes, orthogonal to functional class. These will normally be defined by the application to reflect organisational associations between objects. For example, a game may define an organisational scope for each team, while a military simulator may define an organisational structure to mirror the military hierarchy. The system could also use organisational scopes to split overcrowded aspects into sub-parts.

Fidelity and cost are considered in the following sections.

5.3 Fidelity and Abstractions

MASSIVE-3 allows a locale to contain aspects with the same functional and organisational scopes, but different fidelities. An observer can then choose the aspect with the fidelity that suits their requirements and available resources. We call reduced-fidelity aspects "abstractions", as in [4] (c.f. "aggregates" in [10]). Abstractions may display data at a lower resolution, for example using geometry with lower polygon counts or objects with lower update rates. Alternatively, abstractions may contain composite representations of multiple objects.

For the purposes of rendering, aspects with different functional or organisational scopes are complimentary (providing additional non-overlapping information) while aspects that differ only in fidelity are alternatives (providing variants of the same underlying information). So two aspects differing only in fidelity will not be rendered at the same time. However, aspects that differ only in fidelity *may* be replicated at the same time, to allow rendering to switch quickly between the alternatives without waiting for the new aspect to be loaded across the network.

In addition to abstractions of single locales we also allow abstractions of multiple locales, or abstractions of abstractions (as in [10] and [4]). A normal link between two locales indicates only that those locales are linked (e.g. adjacent). However a parent locale (defined by explicit parent/child links) is regarded as an abstraction of its child locales.

5.4 Policies and Interest Management

The main "work" of interest management is to determine what should be replicated, received, rendered, etc. In the approach adopted here interest management is performed at the level of locales and aspects, i.e. the interest management system must choose which locales (and which aspects of those locales) should be replicated, and which of these should be rendered.

SPLINE [8] had a built-in interest management policy that replicated and rendered the user's current locale and its immediate neighbours. In MASSIVE-3 we have avoided a single specific policy, adopting instead a framework into which application-defined policies can be dynamically inserted, as required. In this framework all policies have the same basic structure, outlined below.

1. Begin new selection round: reset budgets and internal state.
2. Sort known locales and aspects according to a policy-specific ordering criterion (i.e. assigning priorities).
3. Iterate over each aspect in the sorted list: determine whether to select it, based on aspect/locale characteristics, choices made so far, and remaining budget(s). If required by the policy re-sort remaining locales and aspects after each selection.

For example, SPLINE's adjacent neighbour strategy can be seen as a special case of a more general strategy that replicates all locales within N steps of the current locale. This policy has no specific budget, sorts the known locales according to topological distance from the current locale, and accepts all locales (aspects) whose distance is less than or equal to N.

Currently, MASSIVE-3 includes standard policies to select locales or aspects based on topological distance, Euclidean distance, awareness (each inter-locale link includes an awareness value describing the level of awareness across the link), and benefit/cost (which are statically configured at present, but could be derived from actual activity). Different policies can be specified for different functional and/or organisational types, and for different fidelities. Multiple policies can then be combined using logical conjunctions (e.g. policy A OR policy B). Figure 4 shows a simple example of replication policies based on topological distance.

MASSIVE-3 uses independent policies for replication, graphical rendering and audio (although the rendering and audio policies can only choose among currently replicated aspects, whereas the replication policy chooses among all known aspects).

There remain open issues regarding the balance of control over policy between a particular client application and the author(s) of the virtual world. For example, some virtual worlds only make visual sense if the rendering policy has certain properties. Our interim solution to this problem is to allow locales to contain hints regarding the appropriate selection policies to use with them (represented by specific attributes).

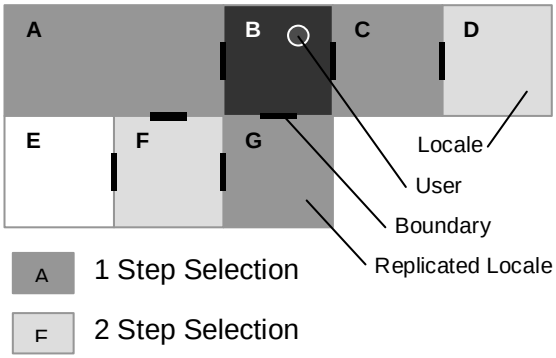


Figure 4. N-step selection examples.

6 CONCLUSIONS AND FUTURE WORK

We believe that the explicit event-based architecture has good potential to support self-management and provide an effective and adaptive service for a range of capabilities of machine and network. We are currently investigating specific enhancements to support low-bandwidth users.

By adding explicit update request items to the MASSIVE-3 database we have been able to support a range of different data consistency schemes, even on different items within the same virtual world.

We believe that locales, aspects and policies offer an expressive and flexible framework for structuring virtual worlds and managing replication and rendering. For example, this capability has also been fundamental to our approach to record and replay mentioned in section 3.4 (since a replayed aspect can be treated in many respects exactly like a normal aspect, and so is easily integrated with the rest of the system).

We also anticipate that locales and aspects, because of the semantic distinctions which they embody, will provide useful hints for managing persistence and caching, and we will be exploring this in the near future.

Other specific areas for further exploration include:

- The best choices of replication and rendering policies, especially if these can integrate dynamic information on likely resource requirements from the running system.
- Appropriate and effective forms of abstraction that could be used in a range of applications and virtual worlds (i.e. could be provided as general system facilities).
- A more complete integration of the locale approach to interest management with our prior work on the spatial model of awareness, including full support for awareness negotiation in MASSIVE-3.

7 ACKNOWLEDGEMENTS

This work has been supported by UK EPSRC research grants GR/L 01527, GR/L 02074 and GR/M09223. We would like to thank Dave Roberts at the University of Reading for valuable contributions to this work and to the HIVE project.

8 REFERENCES

[1] Roberts, D.J., and Sharkey, P.M. (1997), “Maximising Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, Whilst Minimising the Effect of Network Delays”, Proceedings of the 6th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE), June 18-20, 1997, Cambridge, Massachusetts, USA, IEEE Computer Society, ISBN 0-8186-7967-0, pp 161-166.

[2] Chris Greenhalgh, "Awareness Management in the MASSIVE Systems", Distributed Systems Engineering, Vol 5, No 3, September 1998, pp. 129-137, IOP Publishing.

[3] Chris Greenhalgh and Steve Benford, MASSIVE: A Virtual Reality System for Tele-conferencing, ACM Transactions on Computer Human Interfaces (TOCHI), Volume 2, Number 3, pp. 239-261, ISSN 1073-0516, ACM Press, September 1995.

[4] Greenhalgh, C. M. and Benford, S. D., Supporting Rich And Dynamic Communication In Large Scale Collaborative Virtual Environments, Presence: Teleoperators and Virtual Environments, Vol. 8, No. 1, February 1999, pp. 14-35, MIT Press.

[5] Emmanuel Frécon, Chris Greenhalgh, Mårten Stenius, “The DIVEBONE - An Application-Level Network Architecture for Internet-Based CVEs”, in Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST’99), December 20-22nd 1999, University College, London, ISBN 1-58113-141-0, ACM, pp. 58-65

[6] Chris Greenhalgh, Jim Purbrick, Steve Benford, Mike Craven, Adam Drozd and Ian Taylor, “Temporal Links: Recording and Replaying Virtual Environments”, submitted to SIGGRAPH 2000.

[7] Un-Jae Sung, Jae-Heon Yang, Kwang-Yun Wohn, “Concurrency Control in CIAO”, in Proceedings IEEE Virtual Reality ’99.

[8] Barrus, J.W., Waters, R.C., and Anderson, D.B. (1996), “Locales: Supporting Lange Multiuser Virtual Environments”, in IEEE Computer Graphics and Applications, 16 (6), Nov., 50-57.

[9] M.R. Macedonia, M.J. Zyda, D.R. Pratt, D.P. Brutzman, and P.T. Barham, "Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments", in Proc. 1995 IEEE Virtual Reality Annual International Symposium (VRAIS95), 11-15 March, 1995, RTP, North Carolina.

[10] S.K. Singhal and D.R. Cheriton, "Using Projection Aggregations to Support Scalability in Distributed Simulation", in Proc. 1996 International Conference on Distributed Computing Systems (ICDCS'96), 1996.