Highly Interactive Scalable Online Worlds

Graham Morgan

Graham.Morgan@ncl.ac.uk

School of Computing Science,

Newcastle University,

Newcastle-upon-Tyne

NE1 7RU

UK

Abstract

The arrival, in the past decade, of commercially successful virtual worlds used for online gaming and social interaction has emphasised the need for a concerted research effort in this media. A pressing problem is that of incorporating ever more elaborate gaming scenarios into virtual worlds while ensuring player numbers can be measured in the millions. This problem reaches across a number of research areas in computing science and has already received attention from the research community in its own right. In this chapter the major problems associated to the provisioning of expected player interaction in large scale virtual worlds is described together with how research efforts may tackle such problems. Conclusions are drawn from observations of related work and a number of future challenges highlighted.

1. Introduction

2. Gaming Scenarios

 2.1 A Classic Model

 2.2 Cause and Effect

 2.3 Ordering

 2.4 Dynamic Environments

 2.5 Reaching Agreement

 2.6 Groups

 2.7 Timely Progression

 2.8 Best Effort

3. Related Work

3.1 Early Days

3.2 Persistent Worlds

3.3 Synchronisation

3.4 Load Balancing

4. Core Problems

4.1 Where am I?

4.2 Who to Tell?

4.3 Ho to Inform?

5 Conclusions and Further Work

5.1 Advanced Interest Management

5.2 Standardisation and Inter-organisational Issues

5.3 Content Management

1. Introduction

There are a number of commercial solutions to online gaming within which players may participate in virtual worlds that are persistent in nature. Such games are commonly termed *Massively Multiplayer Online Role-Playing Games* (MMORPGs), which is usually shortened to MMOs. Vendors generate revenue from such gaming environments by regular financial subscriptions made by players and/or from the value of virtual world artefacts (e.g., virtual land sales, percentage take from the inter-player trading of virtual world artefacts, sale of additional vendor created virtual world storylines and artefacts). Fundamental to measuring the financial success of such games is the number of players actively participating: the more players there are the higher the financial rewards for a vendor. For example, World of Warcraft has boasted over 10 million subscriptions at its peak (subscriptions are typically $14 per month) [114]. An inability to attract sufficient player numbers leaves such gaming environments unprofitable and ultimately a wasted business venture. Such a waste is significant as the budget for bringing such games to market may be in excess of $10 million [100], with some placing the figure closer to $50 million [101]. In addition, once an online game is up and running the maintenance costs may require total investment, including start-up, of close to $500 million to contemplate competing as a market leader [101]. These are the figures commonly discussed as of 2008; in years to come one may assume that vendors of such games discuss investment of in excess of $1 billion. These games are expected to become an integral part of many individuals' leisure time. Having only been around for a decade yet attaining a significant business status, the notion of carrying out research into online gaming should be taken seriously by industrialists and academics alike.

As the number of participating players is an indication of financial success, a pressing research problem is the need to provide scalable solutions for MMOs. One may assume that scalability has been achieved as no new players are ever turned away from a commercial MMO. However, scalability should be measured not only by how many players can log into a virtual world, but how many players can interact with each other at any one point in time and what level of interaction is afforded.

Presenting the most attractive gaming scenarios via rich interaction provides a competitive edge in MMOs and is one element of online worlds that players will immediately identify as desirable. This is because vendors attempt to immerse players in their online worlds. Such immersion is only achievable by the ability to afford heightened realism via a highly responsive environment together with minimal hindrance to in-world player interaction.

There is no doubt that existing commercial solutions have achieved success and brought to market a series of excellent products. The purpose of this chapter is not to indicate that their efforts are not admirable, but to indicate that these are the first steps taken in this area and one may assume that significant improvements will be expected in the future. A subset of such improvements will be related to player interaction within a virtual world whilst maintaining scalability. As this is a fundamental challenge in creating MMOs, research efforts are still required in this area.

There are already a number of research efforts addressing scalability and interactivity in MMOs, with a number of academics contributing to ever more appropriate solutions for over twenty years. Early works do address the scalability/interactivity problem and do provide many of the techniques that modern commercial products base their solutions on. More recently works have continued to address scalability and interactivity in the context of MMOs, yet such works appear in a number of different areas of computing science (e.g., graphics, distributed systems, parallel simulation). As such, the MMO researcher is faced with a wide variety of different approaches and possible solutions. Furthermore, there exists a large body of work conducted that is not achieved in the context of MMOs, but may provide MMO researchers with a valuable resource. In the future, researchers in other fields may recognize the significance their work may have for MMOs and tailor their solutions appropriately.

The aim of this chapter is to provide an introductory text which explores the problems of MMO scalability and to describe research efforts that may be of benefit. This is achieved by first describing the type of gaming scenarios that may occur in MMOs and relating such scenarios to classic problems so far tackled in distributed systems research. Related work is then presented that is directly or indirectly related to MMOs. A series of challenges associated to MMO scalability and interactivity is then presented that are still to be tackled successfully, posing a number of questions that reinforce the difficulty of such challenges. Finally, conclusions are presented with a brief view of what future challenges may hold for the MMO researcher.

2. Gaming Scenarios

In this section we wish to ignore, for the moment, implementation details and concentrate on the basic model for describing gaming scenarios. We assume gaming scenarios are prolonged instances of interaction between players in a virtual world. This is not an attempt to actually determine what a game is in essence, but simply a description relating to the mechanics of interaction required to provision a gaming scenario. What defines a game in relation to human interaction is a field of study best left to psychology [1]. For the purposes of this chapter, a virtual world gaming scenario is considered to be similar to gaming scenarios found in the real world.

To promote a tutorial type style, descriptions are presented in an informal way. Formalisms that present the most accurate descriptions are not presented. Such formalisms do exist in other texts and can be gained by the reader via the references presented.

2.1. A Classic Model

A gaming scenario, in its simplest descriptive form, is a series of events witnessed and generated by artefacts of a virtual world. Artefacts may be player controlled (e.g., avatars representing the embodiment of a player) or non-player controlled. Non-player controlled artefacts commonly refer to either an algorithm implementing some sub-routine to present automated interaction or periodically generated events within a virtual world (such as the onset of sunset). For clarity, all artefacts with the ability to cause events are considered in the same manner here. Therefore, a simple model of a gaming scenario could be described as follows: An artefact, say A_1, generates a series of events, say E_1 and E_2, which may be witnessed by a different artefact, say A_2. A_2 itself may generate a series of events that may also be witnessed by A_1, say E_3 and E_4, with an additional artefact, say A_3, witnessing the events E_2 and E_3 only. In this simple example, two artefacts have generated four events between them and such artefacts have witnessed all these events with a third artefact having only witnessed a subset of events. We show this example in the space-time diagram in figure 1 (arrows indicate the "witnessed" property and black dots represent events).

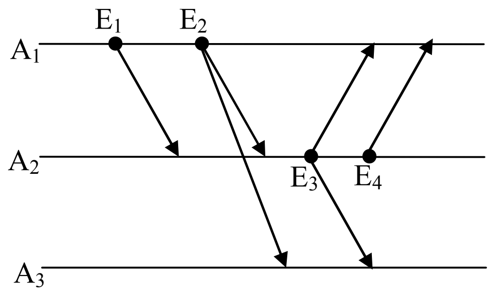


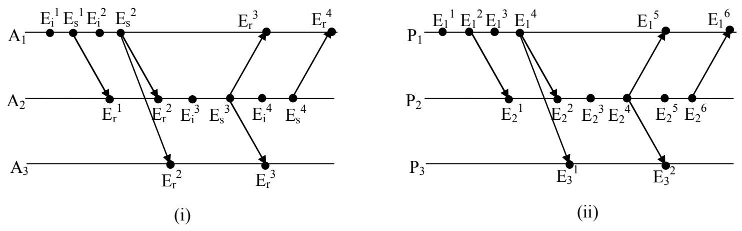
Fig 1. Space-time diagram describing propagation of virtual world events

The act of "witnessing" an event by an artefact may be represented, in its simplest form, via message passing between artefacts: an artefact, say A_1, generates an event, say E_1, that results in a message, say M_1, been sent from A_1 to A_2 to enable A_2 to witness event E_1. This notion of message passing brings our model for gaming scenarios inline with the more general model for distributed computing.

The distributed computing model is now, briefly, described. This description may be found in much more detail penned by other authors (e.g., [2], [3], [4]). However, the description is provided here for completeness and to allow the novice reader sufficient understanding of the model to ease comprehension of this section as a whole. Although reasoning about gaming scenarios with reference to the distributed computing model may appear obvious, this has not been achieved previously with the same detail as presented here.

The distributed computing model is represented by a number of processes connected by a communications network that allows inter-process information flow (message passing) with the overall state of a system described in terms of events and their effect on local processes and channels [4]. Processes may act independently of each other (autonomously) and events may be described in terms of *local* (*internal* - occur at a single process), *send* (sending of a message) and *receive* (receiving of a message). In relation to our discussion so far, we can see that artefact and process are, for all practical purposes, describing the same notion at this level of abstraction. Therefore, to align with other literature artefacts will be described as processes from now on.

Figure 2 updates the diagram in figure 1 to include the send and receive events. In 2.i e_i^x should be read as i identifying the type of event (internal, send, receive) and x identifying the original event as described in figure 1 (to allow comparison). In 2.ii the more appropriate notation is used where i is the artefact (now identified as P for process) associated to the event and x is the number of an event at an artefact (allowing all events to be identified in a unique manner).



E_i^1 E_s^1 E_i^2 E_s^2
E_r^3
E_r^4
A_1
A_2
A_3
E_r^1 E_r^2 E_i^3 E_s^3 E_i^4 E_s^4
E_r^2
E_r^3
(i)
E_1^1 E_1^2 E_1^3 E_1^4
E_1^5
E_1^6
P_1
P_2
P_3
E_2^1 E_2^2 E_2^3 E_2^4 E_2^5 E_2^6
E_3^1
E_3^2
(ii)

Fig 2. Space-time diagram including internal, send, and receive events

The state of a distributed computing model may be considered either globally, or on a per-process basis. Events dictate state change at the process where they occur and the intermediary information link on which they may pass as a message. Considering the space-time diagrams, it is clear that events are ordered in a linear manner at each process. Such a linear ordering is said to represent the *execution* of a process. The global state of a system is said to be represented by the cumulative state of all processes and information flows at a single instance in time. However, as taking such a snapshot is unlikely for many real-world systems, a *consistent global* state suffices. In such a state the premise that all received messages must have been sent must hold, with researchers commonly using this view to describe their systems.

Different assumptions may be made regarding the distributed computing model. These assumptions, ultimately, must reflect the deployment environment of a system. Two basic assumptions that tend to divide the distributed computing community are those pertaining to the reliability of communication links and processes. Processes may fail via a crash manner (faulty processors stop) [9] or byzantine manner (faulty processors continue to produce output) [6] (one must realise there are a number of varying failure models found between these two extremes). Communication links are commonly modelled as either asynchronous (message and processing delays are bounded but unknown) or synchronous (message and processing delays are bounded and known) [5]. For example, systems deployed over the Internet within which compromised (hacked) computers may be present typically favour asynchronous/byzantine type models whereas real-time, failure safe, hardware controlled co-located private network type systems may be more likely represented via synchronous/crash models. Achieving synchronous/crash model environments for deployment requires an overreaching control over all aspects of implementation and is therefore difficult to achieve in many circumstances.

An assumption may be drawn that the modelling of gaming scenarios has its foundations in the theoretical research of distributed computing and, therefore, the same theoretical approach may be used: event generation and dissemination amongst a collection of processes over time can be used to reason about a gaming scenario. This provides researchers into online multi-user virtual worlds with a wealth of existing research from distributed computing on which to draw upon. Indeed, such fundamental work needs to be understood to allow for any reasoning about, and engineering of, the mechanics of gaming scenarios.

2.2 Cause and Effect

Hinted at in the previous section but not explicitly described is the notion of a *causal relationship* between events. This relationship is a key element for aiding in the reasoning about a distributed computing model and, therefore, making progress towards attaining valid gaming scenarios.

The events generated in a gaming scenario may manifest themselves in a variety of ways in a virtual world and may be described via a variety of application dependent types. As a gaming scenario progresses one may assume that the type of one or more events generated by a process may be based on the knowledge of previous events witnessed by such a process. This observation is obvious when considering the alternative: if all processes generated events without consideration of previous events then one would find it inconceivable that a gaming scenario could be described at all (player choice based on current game state is not possible). In essence, when viewed globally we may deduce that an event, say E_1 may have caused an event, say E_2. This is the classic “happens before” relationship as described by Lamport [7] and indicates that E_1 “happened before” E_2 ($E_1 \rightarrow E_2$). The consideration of causal relationships throughout a distributed computation provides a partial ordering of events; partial as simultaneous events (those that do not share a causal relation) may be arbitrarily ordered with respect to each other.

To exemplify the importance of causality consider a gaming scenario consisting of four players (P_1, P_2, P_3 and P_4). The goal of the game is for a player to shoot all other players. The virtual world is constructed from a number of different rooms and players may not shoot beyond the room they are within. For clarity we describe the gaming scenario in plain English first: P_2 enters a room (containing P_1, P_3 and P_4) and is shot by P_1 while P_4 leaves the room and P_1 and P_3 reload their guns at some point during the gaming scenario. To allow this gaming scenario to proceed there is a need to propagate event notification, that is, different players must be informed when certain events happen so they may react. As such, the order in which messages are received are important to ensuring causal relations between events are viewed appropriately by each player. Common practise is to uniquely identify messages in space-time diagrams to afford discussion not only for events but also to associated messages. Furthermore, the notion of a *broadcast* message (same message sent to all possible recipients) is introduced to describe notification of an event for more than one player. A message is described using m_i^j, where i denotes the sending process and *j* denotes the number of the message sent by the sending process. *j* is commonly termed a *logical clock*, in that the message is time stamped not with wall clock time but with a logic based progression (usually incremental integers).

Using the diagram in figure 3 we now describe the scenario stating when events occur. In this model we assume messages are not lost, processes do not fail, and message transit is FIFO. P_2 enters the room where P_1, P_3 and P_4 reside (E_2^1) at approximately the same time as P_4 leaves the room (E_4^1), which is witnessed by all players via M_2^1 and M_4^1 respectively. P_1 loads their gun (E_1^1) and shoots their gun at P_2 (E_1^2). The firing of the gun is seen by all (M_1^1). P_2 realises they are shot (M_1^1) and dies (E_2^2), informing all other players of their mortal wound (M_2^2). During the shooting of P_2, P_3 reloads their gun (E_3^1). A number of events can be ordered arbitrarily with respect to each other (e.g., E_1^1 and E_3^1), with many events exhibiting causal relations (e.g., $E_2^1 \rightarrow E_1^1 \rightarrow E_1^2 \rightarrow E_2^2 \rightarrow E_2^3$).

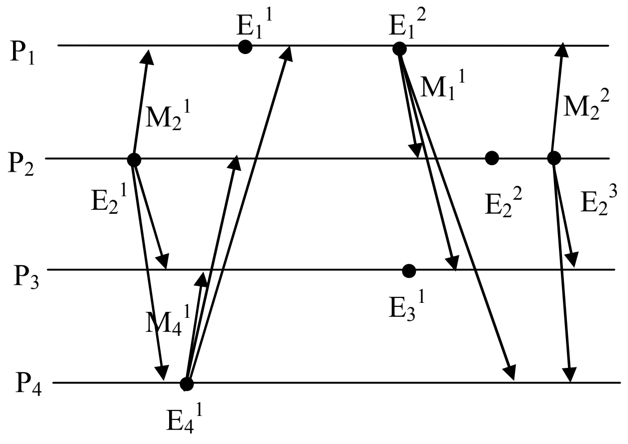


Fig 3. – Causality in gaming scenarios

By considering figure 3, we can identify important information about the gaming scenario and make some judgement on a game's validity. This can only be achieved by retaining the causal ordering of events. In our example this was the case. However, by considering the impact of message latency on our model the ability to maintain causal ordering becomes a challenging issue.

In figure 4 message latency plays an important factor. Consider the message associated to P_3 being notified of P_2's entrance to the room (M_2^1) delayed. As a result, P_3 is notified that P_2 is shot before P_3 realises that P_2 is in the room. Due to the lack of preserving causality P_1 has gained an unfair advantage over P_3 as the opportunity to shoot P_2 was only made available to P_1.

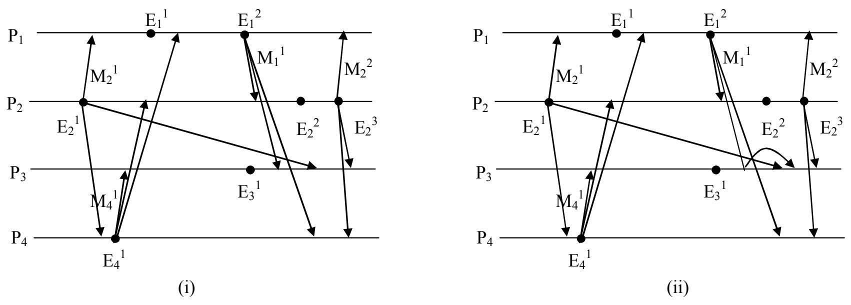


Fig 4 – Causal violation

To preserve causality in figure 4.i there is a need to ensure that P_3 witnesses E_2^1 (M_2^1) before E_1^2 (M_1^1). The term witness is not adequate for describing this process and what actually is required is a distinction to be made between the receiving of a message by a process and the ability to act on such a message. This introduces the classic send, receive and *deliver* approach to describing message handling in distributed computations: although P_3 received M_1^1 before M_2^1, P_3 does not actually deliver M_1^1 until it has delivered message M_2^1 (preserving causal ordering). This delayed delivery is shown in figure 4.ii.

2.3 Ordering

Although causality is an important element that should not be ignored when modelling gaming scenarios, it is by no means the only ordering constraint that should be considered. Sometimes causal ordering is not a sufficiently strong ordering guarantee for the purposes of modelling gaming scenarios. Returning to the example in figure 4.i, the inability to afford an equal opportunity to both P_1 and P_3 in attempting to shoot P_2 is considered a problem. This problem will manifest itself in the virtual world by presenting two different views of the gaming arena to P_1 and P_3: one with P_2 present (P_1) and one without P_2 present (P_3). Even with causal relations maintained, a similar problem may occur with respect to realising who is in the room at the beginning of the gaming scenario.

Consider figure 5 where message transit times are greater than zero and may vary for different links in a network. In this instance M_2^1 is delayed and arrives at P_4 after M_4^1 has been sent (no causal relationship exists between E_2^1 and E_4^1 nor their associated messages M_2^1 and M_4^1). Played out in a virtual world, P_1 will witness P_2 enter the room (P_1, P_2, P_3 and P_4 present) then P_4 leave the room (P_1, P_2, and P_3 present). P_3, on the other hand, witnesses P_4 leaving the room (P_1 and P_3 present) before P_2 enters the room (P_1, P_2 and P_3 present). There is no causal relationship present between E_2^1 and E_4^1 as M_2^1 and M_4^1 arrive at their destinations after E_2^1 and E_4^1 have occurred (indicating that one event could not have caused the other). Unfortunately, the manifestation of this in the virtual world still provides inconsistencies. This indicates that although some events may not be causally related as they happen simultaneously, in a logical sense, there may still be a need to impose some form of ordering on them to preserve a gaming scenario's validity.

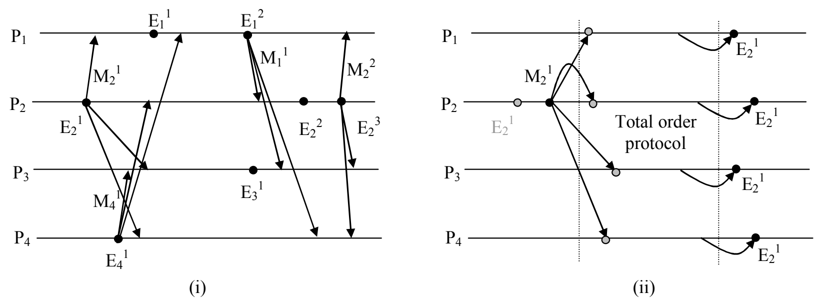


Fig. 5 – Different views

To ensure that P_1 and P_3 install the same consecutive views relating to when P_2 and P_4 are present in the room, an ordering guarantee stronger than causal ordering is required. *Total ordering* [8] is capable of ensuring that all participants view global events in the same order. This is not simply a case of ensuring that M_2^1 and M_4^1 are received in the same order at P_1 and P_3, but the order in which all participants (including P_2 and P_4) receive M_2^1 and M_4^1 must be the same. In fact, to ensure the total ordering of global events at all participants the events themselves must gain their ordering from the underlying protocol governing message delivery. If this was not the case then P_2's view would be that of leaving a room before P_4 entered whereas P_4's view would be that of entering a room with P_2 still present.

Total ordering is achieved with the use of a broadcast to all participants, allowing all participants to ensure they are observing the same ordering of message delivery. Figure 5.ii identifies these steps with respect to P_2 leaving the room. The event equivalent to leaving the room (E_2^1) is attempted (but not carried out – i.e., a request to leave the room by a player) at the originating participant (P_2). This event is shown in a shaded manner to distinguish this from the processing of an event. Once the initial broadcast has been achieved a number of further message passing will be required to ensure total ordering (not shown) until eventually E_2^1 is delivered to all participants, including the originator P_2.

Total ordering is primarily designed to ensure consistency of state for deterministic state machines [10], particularly useful in replication schemes used in fault-tolerance (e.g., [11], [12]). The guarantee that if all replicas receive the same messages in the same order then their states will not deviate (this cannot be guaranteed for non-deterministic state machines). Therefore, state change events should always be propagated across all replicas to ensure states remain mutually consistent. If this route was followed in the example then local events would need to be propagated to ensure all processes maintained a mutually consistent view of the state of a gaming scenario (e.g., E_1^1).

2.4 Dynamic Environments

When discussing total ordering in the previous section a broadcast (message sent to all) was used as the basic message dissemination technique. For practical purposes this is not appropriate as one may expect only a subset of participants to be involved in any one gaming scenario at a time. Therefore, the *multicast* is a more appropriate message dissemination technique, allowing players to join and leave gaming scenarios as they wish. Multicast introduces the concept of a "*group*". A group identifies the recipient of a multicast message with the membership of a group having the ability to change over time. In the example in figure 5 P_2 and P_4 change the membership of the group of players who are "in the room". The problem of "who is in the room", discussed in the previous two sections, highlights another problem that requires more than ordering protocols to aid in deriving an appropriate solution. This problem relates to determining exactly when messages are deliverable in the presence of dynamic group membership. We continue to use the "who is in the room" example to describe the issues that arise.

In figures 4 and 5 P_4 is still receiving messages after they have left the playing area (the room). Therefore, a more appropriate approach would be to restrict multicast messages to include only those inside the room. We would like participants to install the views of room occupancy as follows: P_1 and P_3 ($\{P_1, P_3, P_4\}$ followed by $\{P_1, P_2, P_3\}$); P_2 ($\{P_2\}$ followed by $\{P_1, P_2, P_3\}$); P_4 ($\{P_1, P_3, P_4\}$ followed by $\{P_4\}$). Notice how P_2 and P_4 have views that only include themselves at some point to hinder the inappropriate multicasting of messages (we assume there is nobody else outside in neighbouring rooms).

The ordering of views in a dynamic environment alone is ineffective if we don't order the event dependent messages with respect to view changes. For example, we may be able to ensure that all participants provide the same view changes in the same, total, order. However, if the set of messages in such views varies from participant to participant we will not solve the problem highlighted in figure 5. Therefore, there needs to be some guarantee to ensure the same set of messages is delivered to all participants in the same view, disallowing message delivery when view changes are being determined. For example, in figure 6 the view change event occurs at the initial steps of the gaming scenario, therefore, this view change should complete to ensure all participants' progress with the same messages delivered in the appropriate views. As with ordering, multiple messages will be required to allow all participants to realise the appropriate group membership changes.

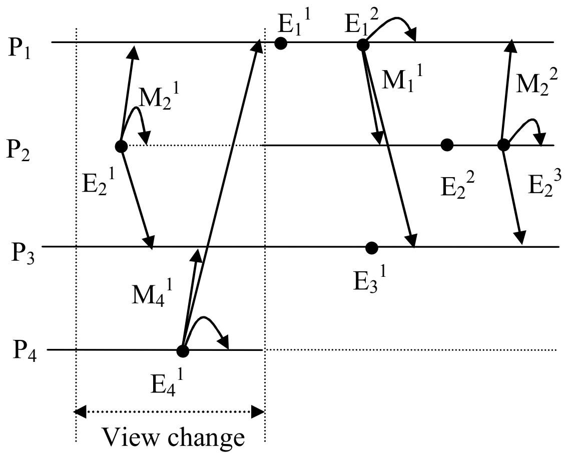


Figure 6 – View changes

Virtual synchrony [13] is the term used to describe the total ordering of view changes with respect to other messages (e.g., the ones responsible for propagating events). Notice that the definition of virtual synchrony does not impose total order on other messages, just the view changes with respect to all other messages. Therefore, it is quite conceivable to have a causal ordering with virtual synchronous systems.

2.5 Reaching Agreement

An underlying problem that arises frequently in distributed computing models is that of *agreement*. In essence, the previous examples are strongly related to agreement as one may assume that agreement on message ordering and group membership is a requirement that processes must satisfy. The agreement problem assumes that a process has an initial value to share with all other processes in its group (all processes must agree on this value) [6]. Alternatively, all processes may have their own, initial, value and all processes must agree on a single value [6]. The latter scenario is known as the *consensus problem* but, for the basic interpretation made here, can be viewed in the same manner as the agreement problem.

Consensus is the cornerstone of many fault-tolerant systems, as reaching consensus on who has failed is a problem that must be handled. For example, if three replica services provide fail-over for clients, all non-faulty replicas must agree on who is faulty to allow fail-over to proceed appropriately. In addition to fault-tolerance and consensus, other flavours of consensus exist: approximate agreement (where agreement is to determine values similar to each other), probabilistic agreement (where agreement is sought with a high probability) [4].

While considering agreement it is worth realising that it is impossible to implement an agreement protocol in asynchronous environments when in the presence of faulty processes [14]. One simple way to visualise this impossibility result is to consider how one may tell the difference between a correct process and a failed one. Basically, when message and processing delays are unknown, it is impossible to tell if a process is slow or failed; how long will you wait for a response? For a broader discussion on the impossibility to resolve a number of problems in distributed computations in general the reader is referred to [25].

Circumventing the impossibility problem of reaching agreement in asynchronous environments has been tackled extensively in the literature on fault-tolerant computing. Two variations are available. One utilises the notion of unreliable failure detectors [15] [16]; described in very brief, but clear terms: allow incorrect suspicion of failure to prevail, as long as some agreement on failed processors may be reached in a number of correct processors at some point in the future (reducing the outcome to a probabilistic chance of success). The compromise made is that correct processes may be incorrectly identified as failed during this process. Another variation, and most widely used, is via transactions: two-phase commit may be used to indicate to a group of processes the steps of preparing a value for committing, then demanding that such a value be "committed" to all participating processes' states [17] [18] [19]. The sacrifice here is that processes guarantee to commit the required state change they promised to and may not participate further until such guarantee is satisfied. Both these approaches carry substantial messaging overheads. In particular, transactions rely on persistent storage to ensure that when a process returns to correct operation any outstanding transactions may be committed. For a discussion relating these two approaches, identifying their differences and similarities, the reader should note the paper [87].

2.6 Groups

A collection of protocols that provide the message dissemination abstractions discussed so far (possibly more) are commonly termed *group communication protocols* [13]. Such systems are primarily the domain of the fault-tolerant research community and concentrate on asynchronous environments, with many design and implementation variations possible. This area of research has provided a substantial number of papers and software products. This is primarily due to the many assumptions that can be made regarding the deployment environment and the behaviour of group members themselves. As the impossibility result is something that cannot be circumvented, the ability to "inch" towards ever more appropriate solutions is a quest taken up by many [4].

Software products that provide group communication services have a number of components: ordering protocols (possibly more than one); failure detectors (based on unreliable failure detectors); group membership protocols (providing dynamic groups); reliable multicast (commonly termed *atomic multicast* – termed *atomic broadcast* in the literature as consideration of sub-groups not necessarily considered in the basic problem description) [15]. In addition to these basic services, such products may also provide: overlapping groups (members may simultaneously belong to more than one group) [17]; open groups (allowing processes to send messages into groups that they are not a member of – the standard alternative is the closed group approach) [12]; partitionable operation (due to incorrect suspicion of failure, or network link failure, groups may partition into multiple, distinct, sub-groups) [21].

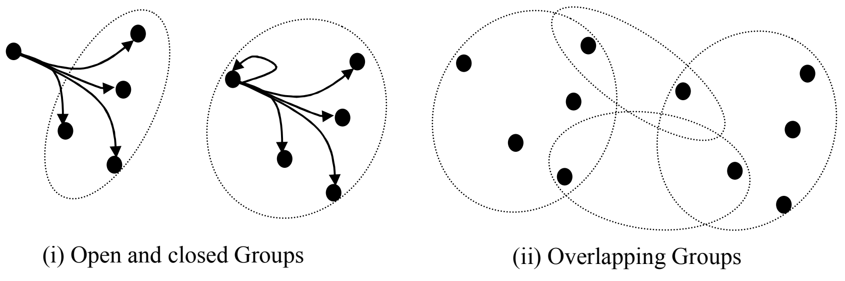


Fig. 7 – Some group configurations

Although there are many minor variations available for the developer to choose from when designing group communication protocols, the primary design choice when considering ordering of messages is between *symmetric* and *asymmetric* approaches [22]. In the symmetric approach all group members cumulatively assume responsibility for message delivery guarantees, requiring group members to participate in a number of message passing rounds with all other group members. In the asymmetric approach a single group member (sequencer) assumes responsibility. Non-sequencer group members unicast their messages to the sequencer, which orders such messages and subsequently multicasts (in order) to group members. An underlying network that provides FIFO message ordering is required for the trivial implementation of asymmetric ordering. In practical situations asymmetric ordering can provide significant performance benefits over symmetric approaches as fewer messages are required (unicast as opposed to multicast) and messages arrive in the appropriate order. However, when a sequencer fails no forward progression can be made until a new sequencer is elected (usually from the remaining group members). Sequencer election (sometime called leader election) faces the impossibility result (agreement required) and is not a trivial issue and may be extremely time consuming to accomplish (possibly resulting in multiple sequencers which must be handled) [23] [24].

2.7 Timely Progression

Not mentioned so far, yet of great importance to modelling gaming scenarios appropriately, is the need for timely progression. In the previous sections there was a logical view taken of gaming scenarios with the length of time required to execute events and send messages not considered. However, virtual worlds are expected to provide players with the illusion of real-time (or at least near real-time) interaction. Events that appear to occur "too slowly" may destroy such an illusion and render the gaming experience inappropriate: virtual synchrony, total ordering, and failure free environments (if such an environment could be created) will not prevent excessive delays in event propagation from ruining a gaming scenario. For example, in figure 6 it may be possible to implement total ordering and virtual synchrony appropriately, but there is nothing in this logical view of the world preventing P_1 from viewing the leaving of P_2 and the arrival of P_4 before P_3 in (real) global-time. By not considering time we are not providing a "fair" gaming scenario for players. All observations so far have been made in "logical time".

Synchronous environments provide an opportunity to include timing when describing gaming scenarios. For example, if one realises that message delays and process delays have a known bound, then synchronisation of local clocks may be achieved with minimal effort. Once this step has been achieved, then placing timeouts on the expectation of player interaction can be worked into an implementation. Furthermore, given the known timeouts associated to a system some design choices may be made to determine what gaming scenarios are actually possible and prevent needless explorations of gaming scenarios that are impractical.

Gaining a synchronous environment is difficult. In practice developers attempt to focus on certain elements of a system that may be made synchronous, possibly using enhanced networking protocols and hardware devices to gain as close to a synchronous environment as practically possible (e.g., [26] [27]). However, even with such approaches a major problem with gaining a universally synchronous environment is the presence of third party devices that are simply beyond a developers/systems control but a necessary part of an overall system's operation. In commercial online gaming these are many (e.g., ISP, home console, gaming interface, variable player interaction times).

If one does not consider real time (wall clock time) then there could be anomalous behaviour exhibited by a system. This is because logical time may create an ordering of events that does not reflect the same ordering when viewed in wall clock time. For example, consider two events E_1 created by P_1 at 10.15am and an event E_2 created by

P_2 at 10.30am. If no causality exists between such events then it is quite conceivable that these events may be viewed in the order E_2 followed by E_1 in a virtual world. The virtual world will be consistent, but the behaviour of the virtual world may appear distinctly odd to players. Therefore, wall clock time is a concern to any that wish to model gaming scenarios appropriately and logical time alone, although important for attaining consistency, is only a partial solution.

2.8 Best Effort

Considering the difficulty, and in some cases the impossibility, of providing gaming scenarios that reflect real-world interactions in commercial virtual world solutions there is a need to make compromises. Compromises must be handled in the game play itself. In other words, the illusion of interaction is maintained while the underlying protocols governing such interaction do not always provide the required message delivery guarantees.

After acknowledging that erroneous situations will occur with respect to message delivery, a developer must decide how much effort (time/processing) the underlying system expends to progress towards appropriate modelling of a gaming scenario at the expense of real-time requirements. In the research community primarily concerned with online virtual worlds this has been termed the *consistency/throughput trade-off* (this term originally concerned itself with the throughput of a network as opposed to additional message passing requirements) [28]. Basically, the consistency referred to is the desire to allow all players to have a mutually consistent view of a gaming scenario. However, in commercial virtual worlds this manifests itself not so much in non-consistency of views but in restrictions on what is and is not possible in gaming scenarios.

In practical solutions the consistency/throughput trade-off manifests itself most visibly when a virtual world is required to be scalable. Scalability in virtual worlds is commonly measured as the number of participants that can be supported simultaneously. As protocols enforcing a degree of consistency tend to produce message volumes that grow rapidly when participant numbers rise and message delivery delays tend to be related to the slowest participant, scalability is difficult to achieve. To achieve scalability there is a need to send fewer messages and not to wait too long before messages become deliverable. Three approaches exist to allow consistency to be “traded” in favour of scalability requirements. These three approaches approximate to the three elements of the distributed computing model described so far:

- *Messages* – relax delivery guarantees
- *Events* – allow players to witness “approximated” events
- *Players* – only inform players of events they may be interested in

Relaxing message delivery guarantees equate to allowing some messages to be “lost” (either at process buffer overflow or network level), and tolerating inappropriate order delivery (possibly with varying view inconsistencies with respect to group membership). Approximated events reduce the need for message passing for event propagation. An event, say E_1, occurs at one process, say P_1, but is not disseminated to other processes, say P_2 and P_3. P_2 and P_3 create the (approximated) event locally (without message passing). When creating such an event some prediction method may be used (a technique commonly used is *dead reckoning*) [29]. The approximated event will be different, but (hopefully) within some error bound as to allow such an event to present an appropriate progression in a gaming scenario. Limiting the number of processes that are sent messages via the identification of player interaction, again, reduces the need for message passing. The basic idea is simple: only send messages to those processes that are actually interested in them and prevent the sending of messages to processes that are not interested in them [30].

Considering the optimisation approaches suggested, guarantees for message delivery for online gaming are more relaxed that those found in the fault-tolerant community's approach to group communications. However, the goals both communities are attempting to achieve are not dissimilar and share a common model. For example, online gaming must approximate a group membership protocol (only sending game events to those interested in them) and at least some messages must be delivered to receiving nodes at some ordering level to afford correct, and expected, player interaction.

3. Related Work

In this section we describe a number of related works that have contributed to the current state-of-the-art for large scale virtual worlds. The earliest works are considered first, followed by descriptions of commercial solutions. The more specific issues affecting scalability (synchronisation and load balancing) are then described. At this point the discussion of related work broadens to include those works that were not carried out in the context of virtual worlds, but tackle similar problems.

3.1 Early Days

The early pioneers in the creation of virtual worlds came from a variety of research backgrounds: high performance graphics, human computer interaction, commercial gaming, virtual reality, military simulation. Many of the basic notions of what it takes to build scalable virtual worlds were discovered and experimented with in these early days. One of the truly admirable aspects of this early work is that real systems were built and demonstrated in both academic and commercial settings. All the techniques that attempt to gain increased scalability, see 2.8, were all demonstrated in these early systems for the first time. The work is substantial (it was quite a busy area in the 80s and 90s) and whole books have been written about these systems (e.g., [28] [31]). Only the most relevant developments that directly relate to the attempts of scalability are discussed here.

Throughout the 80s (1983 onwards) SIMNET (simulator network) [29] was developed to provide the American military with a virtual battlefield on which to train individuals. A number of simulators (e.g., tank) could be networked together. The successor to SIMNET, DIS (Distributed Interactive Simulation), aimed to standardise and generalise a protocol for use in more heterogeneous environments as SIMNET was not an "open" platform [32]. In these early systems message ordering and reliability guarantees are *deliver when receive* (no further message passing to enforce any ordering or reliability). Dead reckoning was used to lower the message passing burden with participant numbers expected to be less than 1000 (designed for around 500). No central server was used, with a peer-to-peer architecture assumed. Participants could arrive and leave at arbitrary points throughout the execution of a simulation. Messages were lost, or arrived out of order. Inevitably, inconsistencies in the simulations would occur (conveniently termed “the fog of war” [28]). Inconsistencies aside, these two early systems provided functioning virtual worlds that served their training purposes well for the American military [31] with increasing standardisation resulting in the *High level Architecture* (HLA) [68].

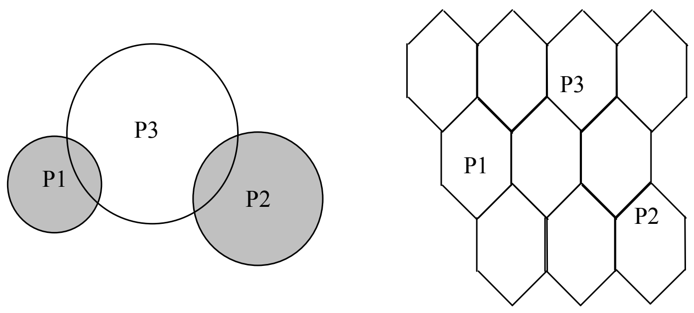
The DIS to HLA transition may be viewed in a similar light as the RPC to CORBA transition that occurred in the mid-90s in middleware technologies; bringing a greater degree of standardisation to how a distributed application may be structured. The HLA went much further than DIS in its prescriptions, indicating artefact representation in a virtual world. Immediately after the introduction of the HLA the amount of work in online worlds appears to have decreased in the literature, possibly due to the USA’s Department of Defence’s instruction that all future work in this area must be HLA based, one can’t say for sure. However, since the late 90s the most successful online worlds have been commercial and non-HLA compliant.

In addition to the high cost military projects, a number of PC games appeared in the 90s that could support networking. As with SIMNET and DIS, no respect was paid to message delivery guarantees (e.g., deliver when receive, send multiple times if important [33]). Such games limited player numbers (4 or less for Doom) with players quite often expected to be co-located on the same LAN to ensure network latency would not hinder game play. Even before these games existed players had enjoyed online virtual worlds in the form of *Multi User Dungeons* (MUD) [88] and novel commercial games that afforded limited networking [89]. These early attempts were more a forerunner of *Internet Relay Chat* (IRC) as communications manifested themselves in the form of text messages between players with little graphical representation. In addition, these works are not well documented and only messages on a variety of newsgroups afford insight into the technical aspects of such systems. For these reasons, these works do not afford a significant insight into constructing large scale virtual worlds.

Pioneering academic work in virtual worlds resulted in NPSNET [30] (and its descendents 2, 3, and 4). The military and early commercial work was not documented in the academic literature at the time; therefore, NPSNET presented the first major advances in understanding how to build online virtual worlds in the public domain. For example, NPSNET-IV could interact with DIS and utilise IP-Multicast for more judicious use of bandwidth [34]. NPSNET used dead reckoning to ease the messaging burden. However, message delivery guarantees were best effort and inconsistencies would still be an issue. Further academic works extended ideas and concepts originated in NPSNET. PARADISE allowed a more intricate modelling for dead reckoning [35] and reduced message sending with the ability to retrieve state information for artefacts that send messages infrequently [36]. This protocol was termed the “Log-Based Receiver-Reliable Multicast”, and allowed receivers that noticed a missing message (by way of logical timestamps) to retrieve such messages from a persistent logging server. In actuality, the protocol is not reliable in the same context as atomic multicast is considered reliable and did not solve ordering issues.

From the perspective of group communications, DIVE (Distributed Interactive Virtual Environment) presents an excellent case study [37] [38]. DIVE is considered a *collaborative virtual environment* (CVE); where emphasis is primarily based on collaboration of participants as opposed to realistic simulation (e.g., shared drawing) and was built on the first fully functioning group communications toolkit (ISIS) [39] in the early 90s. ISIS provides many of the elements described in section 2 (e.g., total and causal ordering, virtual synchrony, failure detection) and so provided DIVE with the strongest consistency possible of all virtual worlds (before and since). Unfortunately, choosing a fully functioning group communications service appears to have been a problem, as later versions of DIVE sacrificed their consistency in favour of scalability (dropping the use of ISIS). With ISIS, DIVE could not support more than 20-30 participants without significant deterioration of interactivity between participants in the virtual world [40]. This was the first and last time that the fault-tolerant approach to group communication services would be used to support a virtual world as ISIS clearly demonstrated the lack of scalability. Such scalability is of little issue when dealing with 3 or 7 replicas, but it is an issue when requiring real-time virtual world access for hundreds, thousands possibly millions of participants.

In the mid to late 90s a CVE was developed named MASSIVE (Model, Architecture and System for Spatial Interaction in Virtual Environments) [42]. MASSIVE provided a novel model for attempting to capture the degree of interaction between participants. The *aura-nimbus* model allowed an artefact in a virtual world to "express" their interest in, and their influence over, other artefacts. This model was actually developed prior to MASSIVE (spatial model) [41] and experimented in a limited manner within the DIVE system, yet is always associated with MASSIVE. Figure 8.i shows an example of the aura-nimbus model where the aura of P_3 is overlapping with the nimbus of P_1 and the nimbus of P_2, indicating that P_3 is sending messages to P_1 and P_2. This model is restricting message passing by only sending messages to those participants that are interested in them. Therefore, one may assume this provides an opportunity for trading consistency in favour for scalability. However, the original intention of this model was to enhance interaction (on a per-artefact basis) rather than gain scalability. MASSIVE went through a number of developments, with the long running project producing a further two versions (MASSIVE-2 and MASSIVE-3) [43]. In practise, the aura and nimbus are represented as boxes in MASSIVE (possibly due to their ease of overlap identification in 3 dimensions and the fact this distracts little from the core requirement of determining interaction) [44].



(i) aura-nimbus (ii) NPSNET

Fig. 8 – Regionalisation of the virtual world

Restricting message passing in favour of achieving scalability was actually attempted in the first instance by NPSNET. In NPSNET regions of the virtual world were divided into hexagonal areas, with artefacts in the same (or bordering) regions capable of exchanging messages (figure 8.ii). Hexagonal areas were chosen as there are at most three bordering areas (as opposed to four when using squares). This provided less area of the virtual world (and therefore choice) when disseminating messages as artefacts reach area borders. For example, when a boundary change occurred it may be best practise to disseminate messages within multiple areas to lessen ambiguity over which artefacts should receive which messages.

When considering message dissemination techniques via the use of regions the size of the regions becomes important for dictating the type of interaction possible within a virtual world. A region must be of sufficient size as to ensure players have the ability to engage in gaming scenarios in one region before entering another region [102]. When a player traverses a region boundary a region's membership must be updated (identify a region a player belongs to). Determining a region size that is suitable for all types of player interactions in a virtual world may not be possible. For example, if region size is decided when considering the top speed of a fighter aircraft then the presence of soldiers travelling on foot may give rise to unnecessary message exchange between foot soldiers. If region size is more suited to foot soldiers then a fighter aircraft may traverse region boundaries with such frequency that region membership may not be resolved in a timely fashion (traverse a region in less time than it takes to realize regional membership changes resulting in an inability for fighter aircraft to engage in gaming scenarios).

Auras and regions have their advantages and disadvantages. Regions do not afford the accurate degree of interaction as auras appear to provide on a per-artefact basis, but the implementation overhead for regions is much lower than auras. This is because there is no discovery stage required when deciding upon the appropriate message recipients in the region approach. For example, an IP-multicast address may be associated to each region and as long as an artefact can realise which region they are in, they can subscribe and multicast to the appropriate multicast address. On the other hand, aura overlap must be detected before message recipients may be realised in the aura approach. This will require an initial protocol step with the sole purpose of identifying appropriate message recipients. This proved an expensive step in practice and can severely limit the scalability of aura based approaches.

Other early works continued the exploration into spatial sub-division exhibited first by NPSNET and then in a different manner by MASSIVE. For example, SPLINE introduced the notion of locales which assumed a much more independent view for each spatial sub-division [45] [46]. Each sub-division may be described within its own co-ordinate system, with the appropriate transformations matrices to allow transition from one locale to another. BrickNet uses a more descriptive mechanism (not necessarily based on virtual world geography) to allow related artefacts to be grouped together and become visible to each other (associated to different virtual environments) [47].

3.2 Persistent Worlds

The virtual worlds described in the previous section do not provide persistent environments. That is, they do not exist as some simulated persistent geographic location at some known, accessible, address. Persistent virtual worlds allow participants to enter a virtual world that provides a degree on continuity; artefacts may be created and persist over periods of time and the results of events on artefacts may persist. For example, a participant may purchase a virtual car, drive their car to the end of a virtual road, return some days, months or even years later and retrieve their car. Of course, someone else may have procured the car and driven it elsewhere in the meantime, but the continuity provided by persistence of artefacts is a factor that aids in classifying these virtual worlds.

Public access persistent virtual worlds available over the Internet present vendors with a commercial opportunity. The computer games industry has been able to use these worlds to generate revenue in a number of ways: pay-per-play (often the client program is free, or sold for a small one off payment, with subscriptions required to allow players to participate) (e.g., [48], [49] [50] [51]); artefact sales (participants trade artefacts with commission gained on sales) (e.g., [52]); client extensions (client side extensions are sold that allow access to additional virtual world areas/storylines) (e.g., [53] [54]); land sales (areas of the virtual world are sold to participants) (e.g., [55]). As these gaming arenas grow one may envisage economic structures developing not too dissimilar in variety to those that exist in the real world [56]. This area of online gaming has grown from an insignificant financial element of the games industry in the late 1990s to become a multi-billion dollar industry in its own right as of 2008 [57].

Persistent virtual world implementations are server based, allowing vendors to regulate the provision of ever evolving alternate realities to maintain player interest and, most importantly, restrict participation to subscribed players. Player consoles connect to a server that provides players with access to a virtual world. Typically, a player's console holds a sub-set of game state with players informing each other of their actions via the exchange of messages between consoles. Such communication is achieved via a server, allowing the regulation of player interaction and game state to be recorded and stored onto a persistent medium if required. As revenue is generated on a per-player basis, the more players that can be supported by a virtual world the more revenue may be generated. Therefore, scalability of a server, in terms of player numbers, is of great importance to ensure commercial success.

To satisfy the demand for processing resources, clusters of servers are employed to cumulatively maintain game state and manage player interactions. The additional processing resources required to support an increase in player numbers is satisfied via the addition of servers to a cluster. This approach to server cluster configuration will be familiar to any developer working with scalable service solutions found in almost all Internet applications; utilise a collection of geographically co-located nodes organised into a cluster that cumulatively support online services (e.g., search engines, e-commerce, enterprise information portals). Such nodes are standard computers in their own right, and may operate as service providers independently of each other. Such computers are general purpose and not necessarily tailored for high performance multi-processor solutions, making them a cost efficient approach to server side scalability.

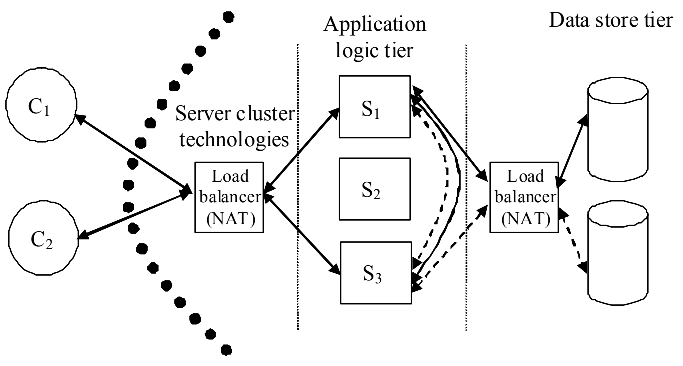


Fig. 9 – Classic n-tier server side solution.

Figure 9 provides an overview of a typical server cluster solution for providing scalable online worlds. Although a simplified view, this will suffice for descriptive purposes. The load balancer ensures players are directed to an appropriate server that may satisfy their service requests (e.g., updating avatar appearance or location). The application logic is where user participation is enacted and overall governance of the virtual world occurs (e.g., avatars fighting, trading artefacts between users). Artefacts, including player's avatars, which populate the virtual world, together with their current state are stored in the data store tier and retrieved as and when required by the application tier. Updates made to persistent artefacts in the application logic tier must be registered in the data store tier to ensure continuity of the virtual world.

Vendors of commercial persistent virtual worlds do not tent to describe in detail the techniques used to achieve scalability at the server side (which is to be expected for a commercial enterprise in a competitive market). However, there is an article describing EverQuest's approach in general terms [58]: a mixture of regions and *duplicate worlds* with each duplicate world supporting approximately two to three thousand players with each world divided into regions based on the geography of the virtual world (the term used in the literature for duplicate worlds is *shards*). As regionalisation is associated to virtual world geography, this approach is closely related to NPSNET's approach of sub-dividing the virtual world geography. A duplicate world is itself supported by a cluster of servers, with regions used to aid in allocating the processing requests originated from player actions amongst such servers as and when required. Due to the similarities in game play and the existence of duplicate worlds; one may assume that all commercial approaches to implementation of distributed player load across the application tier to be similar. There is no player interaction allowed across duplicate worlds although players may pass from one region to another.

Duplicate worlds and geographic regionalisation present a three step approach to identifying localised game play: (i) players do not interact across different duplicate worlds; (ii) players do not interact across different regions; (iii) players interact intricately with other players they specifically target (e.g., click on with mouse). This approach provides two distinct forms of interaction: (i) a general, viewing type style, where players can see the actions of others in their region (assuming appropriate line of sight); (ii) an intricate manner where players directly interact with each other in a user directed way. The latter form of interaction requires consistency to be greater as ordering of events are usually crucial in determining the outcome of an intricate gaming scenario (the server must resolve player interaction). The consistency can be weaker in the general style of interaction as summary information could be propagated between players. For example, in a fight between two players in a virtual world attacks must be regulated (e.g., ordered, not lost in transit) between engaged players (e.g., spells, hitting, shooting) to provide an outcome (e.g., decreased health, loss of inventory). However, for players watching a fight between other players there is only a need to view a series of fighting moves and the end result (that may or may not reflect the actual fight moves as enacted between the fight participants).

Initiating intricate play is via a handshake protocol at the start of an intricate interaction request (specifics vary slightly across commercial implementations, but there is a need for player identification made by the server to initiate such interaction). In the case of player P_1 attacking player P_2, the server will poll P_2 to ensure that intricate interaction may commence. This is to ensure P_2 is actually in a state to which it can respond appropriately and, possibly carrying out some check to ensure P_2 is not at a disadvantage due to inconsistencies between P_1 and P_2's views of the virtual world. This is especially the case if P_1's actions could have an important impact on game play if not responded to in a timely manner (e.g., P_1 attacking P_2). This protocol may be manifested as part of game play itself to ensure players are fully aware of a requested interaction, (e.g., a request is provided to P_2 that may be declined or accepted – either at a player's discretion or transparently by a player's console based on local game state).

An interesting observation in implementation similarities between asymmetric ordering and intricate interaction may be made. Clearly, commercial solutions are relying on sequencer (the server) to regulate intricate interaction (ordering of events) between players. Indeed, direct communication between player consoles is to be avoided in this respect; therefore no leader election protocol is required between player consoles if a server fails. If a server fails one may assume failover may be employed (but there have been instances that show this may not be the case [69]). The importance of allowing server failover specifically for persistent virtual worlds has recently been recognised as a serious problem and is an aim in Sun Microsystems' Darkstar Project [70]. This project looks to hold some promise of bringing a general purpose middleware platform to market that eases the creation of online persistent virtual worlds.

3.3 Synchronization

In section 2 gaming scenarios are discussed with reference to the model used for describing distributed computation. This approach was shown to allow a degree of reasoning when considering the validity of gaming scenarios. As mentioned earlier, this model is primarily used in the domain of fault-tolerant computing where assurances of a system's correctness are paramount and every effort is taken to ensure message reliability and delivery requirements may satisfy such assurances. Unfortunately, probably due to the lack of timing considerations in the model and the failure of ISIS to provide an environment of any "useful" scalability, research into message ordering and reliability protocols for online gaming held little interest to the fault-tolerant community. Instead, the research community that has proceeded to make progress in this area has been the *parallel and distributed simulation* community.

Although the end goals of these two research communities are dissimilar, there are similarities between work carried out in the fault-tolerant community and the parallel and distributed simulation community. They share the same basic model of events, messages and processes and the same concern for preserving causality. However, the parallel and distributed simulation community does not contend with the same rigorous requirements associated to reliable systems (e.g., total ordering, atomic multicast, virtual synchrony). For this fact alone, their algorithms will undoubtedly have a lower message passing overhead and less of a delay between receiving a message and delivery of a message, providing more opportunity for scalability than that witnessed by utilising ISIS in DIVE.

Synchronisation is the term used to describe the end-goal of algorithms designed within the parallel and distributed simulation community. There are two basic approaches described in the literature to achieve synchronisation [59]: (i) *conservative* – messages are received but delivery delayed until delivery guarantees can be satisfied; *optimistic* – messages are delivered as they are received with a possibility that messages may become "undelivered" and then "re-delivered" to correct violations of delivery guarantees realised at a later date (i.e., when receiving a logically stamped later message).

Work on conservative approaches can be traced back to the 1970s (e.g., [60] [61]) and appear at a similar time to Lamport's paper on logical clocks (however, Lamport's interest in this area reaches back to the 1960s). One may assume conservative approaches share a great deal in common to the approaches carried out in the fault-tolerant community as delayed delivery is utilised. Therefore, the developers of virtual worlds find the optimistic approach more inviting than conservative approaches as messages may be delivered without delay, favouring real-time requirements (e.g., [62]). In addition, the virtual world may be capable of a degree of prediction (e.g., dead-reckoning), allowing the application developer to either pre-judge certain ordering irregularities or hide them in gaming scenarios when they occur.

The Time Warp [63] mechanism is a well known optimistic approach. In simple terms, when a process receives a message with a (logical) timestamp lower than a message that has previously been delivered, delivery of such a message (or messages if there are more than one) are rolled back and re-delivered together with the recently received message in the appropriate timestamp order. To limit rollback to an appropriate level there is an identification placed on the length of history possible for rollback.

Conservative and optimistic approaches have been used to attempt synchronisation in gaming scenarios with Ferretti and Roccetti providing a convenient discussion of the state-of-the-art together with some interesting comparisons made between the techniques [64]. Optimistic approaches tend to favour scenarios that can provide a degree of determinism, and may not be suitable for intricate interaction where roll-back is not feasible. For example, when two players are engaging in intricate interaction in a persistent virtual world (as described in 3.2) the notion that some results may be rolled back may deter from an appropriate gaming scenario. In addition, the overhead of roll-back may provide a processing burden that is detrimental to the overall performance of a virtual world if it occurs sufficiently often enough. This may outweigh the alternative approach of delayed delivery found in conservative approaches. The decision on the approach used by a developer is not always straightforward.

Ferretti and Roccetti have pioneered a number of optimistic synchronisation techniques specifically for use in online gaming. A series of works clearly demonstrate the scalability of their optimistic approaches for use over the Internet [106] [108]. Their schemes are based on loose synchronisation of physical clocks [109] and the ability, through negative acknowledgements, to discard events considered "obsolete" in the virtual world. As their approaches are optimistic they deliver messages as they are received, gaining scalability by preventing the sending of some messages due to recognition of the obsolete events with which they are associated.

Including wall clock time (as opposed to logical time) when attempting to gain some ordering guarantees for message delivery has been shown to provide value. Delta-Causality (Δ-Causality) places a limit on the degree of causality between messages by identifying a window of time within which causal relations are maintained [65]. Attempts are not made to ensure causality for messages that arrive too late to be of use. This approach is particularly useful for streamed data (such as voice over IP – VOIP), where out of date message delivery only detracts from the perceived quality of the output stream. [65] is set in the context of streamed media and implemented as *Multi-Flow Conversation Protocol* (MCP). The authors argue that wall clock time is a useful mechanism and practical synchronisation is possible (measured in milliseconds) given modern clock synchronisation techniques (even across the Internet) [66].

The field of distributed real-time systems have provided substantial amounts of research in an effort to maintain causality in message delivery guarantees. The two basic approaches to satisfying such guarantees are via *clock-driven* and *timer driven* techniques. Clock driven is associated to clock synchronisation (as discussed in Δ-Causality) whereas timer driven relies on local timers only and requires some form of acknowledgement message. Verissimo [67] has documented these two approaches, providing an interesting comparison. Probably the two most popular works relating to clock-driven approaches are Δ-protocol [71] family and MARS [72]. The Δ-protocol family and Δ-Causality are not to be confused, as each work appears quite distinct in the literature (Δ-Causality having been attempted in the context of multimedia streams without acknowledgement of the earlier work associated to the Δ-protocol family).

Other attempts at preserving causality in message delivery have been suggested that may be directly, or indirectly, related to modelling gaming scenarios that exploit application level knowledge. For example, Δ-protocol family has been extended for use in small scale distributed embedded systems [73] and an attempt to use application knowledge to "ignore" causal relations between some messages has been suggested [74]. Another approach proposes the notion of, and coins the phrase, *critical causality* [75]. Critical causality identifies that the only causal relation of concern is the directly proceeding event. For example, assume P_1 receives a message m_1 from P_2 informing of event E_1. P_1 then receives a message m_2 from P_3 informing of event E_2. P_1 then generates an event E_3 and disseminates this event to P_1 and P_2 via a multicast m_3. In this scenario E_2 and E_3 are said to be critically ordered (meaning that m_2 must be delivered before m_3 where appropriate). One must note that critical causality is not transitive and the authors assume that critical causality is appropriate for modelling gaming scenarios. The authors of this approach suggest an algorithm which does not place delay on delivery by making sure a process sends both messages that share a critical ordering (in our example P_1 will send m_2 and m_3 together). This algorithm does not guarantee that critical causality is maintained, but is a best effort approach with experiments indicating that critical causality will be preserved 99% of the time in a realistic setting [75].

3.4 Load Balancing

Assuming the approach described in figure 9, the problem of scalability becomes one of load balancing of resources. Furthermore, to avoid wasting resources load balancing must be achieved in an efficient manner (i.e., not have substantial amounts of overprovision at the server side). Load balancing schemes basing their approaches on virtual world geography for online gaming described in the literature may be classified as follows: (i) duplicate; (ii) distinct; (iii) partial duplicate. These approaches are shown in figure 10. Please note that in general purpose clustered solutions for scalable service provision the duplicate and distinct approaches are commonly termed *homogenous* and *heterogeneous* clustering respectively.

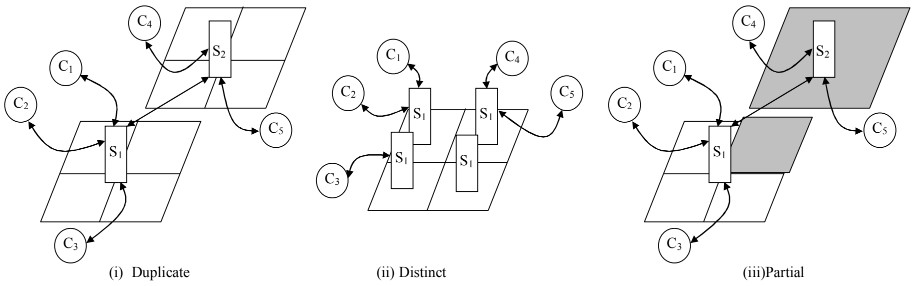


Figure 10 – Load balancing options

In the duplicate server approach each server holds a complete duplicate of the virtual world containing all artefacts. A server will assume responsibility for "ownership" of an artefact, and inform all other servers of updates carried out on such artefacts (relating to messages originated at a player's console). This requires servers to pass messages between themselves to ensure gaming scenarios are modelled appropriately. With the volume of data present on each server there may be opportunities to make use of dead reckoning techniques to ease the message dissemination overhead. Synchronisation of servers is required and so some protocol governing message delivery guarantees will be desirable in this approach. A detailed description of an implementation of this approach has been demonstrated [103] with a number of other works (e.g., [106] [107]) advocating this approach due to a number of possible benefits: *fail-over* – if one server fails other servers may assume responsibility for the failed server's clients (as all servers have some knowledge of cumulative game state); *scalability* – increased client numbers is satisfied by increased server numbers; *responsiveness* – local servers may satisfy the demands of local clients.

In the distinct server approach different areas of a virtual world are maintained by different servers (possibly defined in a similar manner as regions in NPSNET). There is minimum inter-server communications and a single gaming scenario is executed on a single server. The main benefit of this approach is that consistency of interaction becomes an issue to be resolved between a single server and associated player consoles. There is no need for inter-server communications to model interaction as synchronisation between duplicate servers is not warranted to model a gaming scenario. However, a limiting factor is the problem of "full" regions: there is a processing limit dictated by server resource availability on any one particular region. For example, in Second Life this manifests itself as areas been incapable of supporting a level of activity in the virtual world defined by server resources [76] (this is actually on a per duplicate world/shard basis and manifests itself as disjoint islands). Full regions exhibit themselves in the virtual world as "crowding". Unfortunately, crowding is not an uncommon occurrence as players tend to gravitate towards popular events in virtual worlds. In addition, there is a requirement to handle process resource handover between servers when players move from one region to another. This problem in itself has warranted a number of research papers (e.g., [76] [77])

The partial duplication approach is similar to the duplicate server approach apart from the fact that not all the servers are aware of all virtual world interactions. Localised game play is used to identify where synchronisation requirements need to be satisfied across servers. In essence, this approach lies between duplicate server and distinct server approaches. In this approach regions may be allocated to servers dynamically at run-time to alleviate the "full region" problem found in the distinct server approach. Alternatively, load balancing may be achieved by players being assigned to servers and full synchronisation between servers is required to model interaction appropriately, possibly using auras as a basis of determining interaction. This approach has been demonstrated successfully in the context of auras identifying areas of interest to aid in dictating which servers should enact inter-server communications to satisfy gaming scenarios [78] [79].

In section 3.1 the aura and region approaches were described and in section 3.2 Everquest's approach to identifying localised game play was described. The reader should by now recognise that Everquest (and similar commercial products) use regions as a form of load balancing. The identification of localised game play is conveniently used to identify load distribution across the application tier and implement the distinct server approach. Unfortunately, commercial solutions, like Everquest, do exhibit the crowding phenomenon resulting in exhaustion of server resources and full regions. Left unchecked, the effects of crowding may result in a slowdown in game play or, in worst case scenarios, a complete inability to enact player interaction. This may be considered the same problem of consistency management that the distinct server approach is attempting to alleviate: without regionalisation the virtual world itself (single region) may become populated by a sufficiently large number of players as to make the consistency problem unmanageable. In commercial solutions, the number of players allowed into a duplicate world is rarely above 2,500 to offset the problem of resource exhaustion; better to prevent failure in player interaction than allow it. In essence, players load balance themselves by choosing duplicate worlds to enter and are barred from entering those duplicate worlds that are "full" or not available due to maintenance issues.

In the presence of server clustering, there is an opportunity to alleviate the crowding problem by dynamically associating processing requirements generated by player actions during runtime. This takes the form of load balancing player activities across servers with respect to regions and assumes the partial duplicate server approach. The literature provides a number of solutions to load balancing across server clusters suitable for MMORPGs. Regions may be reduced in size by sub-dividing them further (allocating servers to these additional sub-divisions) [80]. Other methods distribute responsibility for region execution to a particular server at runtime based on the volume of players in a region [81], while other methods dynamically resize regions during runtime [82]. Such approaches may be fine tuned further to ensure that the cost of moving responsibility for execution to another server is minimised [83].

EverQuest also describes runtime allocation of resources from within small clusters of servers responsible for a duplicate virtual world. Although no great technical detail is provided on how this is achieved [58], the premise of this approach appears to be player driven: when player enacts a particular action (e.g., opening a door, entering into battle) processing resources are allocated to satisfy the increased processing requirements.

Commercial approaches aside, there are a number of other works in the area of scalable server side solutions that may be appropriate. A notable contribution is work carried out by IBM. IBM has produced region based services that make use of standards such as Web/Grid services [84]. Regions are again used in this work, providing a platform that would allow a similar approach to implementation than would be expected in the commercial approach already discussed. Other works (e.g., RING [85]) do employ multiple servers, allocating regions of virtual worlds to different servers, providing a similar approach to scalability (regions to servers) as advocated in the common commercial approach. Recently The Darkstar project from Sun Microsystems is tackling the scalability issue without dependency on duplicate worlds and instead advocates the scalability problem be solved by distributing tasks over a collection of servers (irrelevant of geographic location of the world). Experimental results are not yet available demonstrating scalability, but this is a project that should be monitored for results in the future [70].

4. Core Problems

There has been a large spectrum of work that is directly or indirectly related to scalable online gaming. In recent years the volume of research related to this area has increased rapidly; many papers on scalability have appeared in the annual ACM SIGCOMM workshop on *Network and system support for games* (*NetGames*), an excellent resource for the latest developments in the area. Many other works have appeared sporadically in a variety of other conferences, ranging from graphics to networking. Correlating such work is a non-trivial task as useful knowledge related to online gaming research may be found in a number of different genres not necessarily produced by researchers primarily concerned with online gaming (e.g., distributed simulation, fault-tolerance, real-time systems, streamed multimedia, human computer interaction). The different genres within online gaming themselves produce their own focussed works (e.g., non-persistent first person shooter [86] [113], streamed content for game artefacts [76]).

Considering the wide spectrum of research activity associated to online gaming one must not lose fact of the basic requirements that need to be satisfied when constructing large scale online gaming worlds. We can structure these requirements into three logical steps that must be achieved to make large scale online games a reality. In their simplest abstraction these basic steps are described as follows:

1. Determine, based on virtual world state, which artefacts are and are not interacting
2. Enable required message dissemination between nodes in a network (e.g., servers, player consoles) while prohibiting needless message dissemination between nodes
3. Manage message delivery to attain appropriate synchronisation to afford intended gaming scenarios

Each of these steps is now considered in turn, concentrating on a number of open questions that each requirement highlights.

4.1 Where am I?

Primarily, virtual world geography is used to determine which artefacts should be interacting. This is not as simple as defining a virtual world distance within which interaction between two artefacts becomes possible. For example, when a virtual world is divided into static regions an artefact close to a region boundary may actually be closer to artefacts in neighbouring regions than artefacts in their own region. The aura/nimbus approach appears more appealing as this issue does not arise. Furthermore, the aura/nimbus model allows areas of interest to be specified on a per-artefact basis (allowing for varying types of artefacts to express varying degrees of influence and interest). Unfortunately, this requires additional processing requirements to determine what interaction is occurring at any one point in time. The only option at this point is to employ a real-time collision detection algorithm to identify such interaction; a substantial processing overhead if in excess of a million artefacts exist at any one time in a virtual world [104]. Furthermore, this is a distributed computation in itself to be carried out across multiple nodes. How does one achieve such a service in a timely manner [102]?

In commercial solutions the problem is tackled by using regions and simply constructing the virtual world to hide the hindrance of interaction found in the static region approach. For example, constructing a large wall preventing players from seeing what is within other regions is a simple, if not elegant, solution. Even with regions and an appropriately constructed virtual world the process of moving from one region to another still requires processing time to allow a server cluster to allocate processing resources effectively. Basically, a player must be slowed down in some way when process resource allocation is changed at the server side due to region changes. Maybe the player can ride on a train between regions, or maybe a door between regions takes time to open. Either way, some game play element must be seamlessly incorporated into a virtual world in a less as intrusive way as possible. When in excess of a few million players are changing regions frequently how can timely requirements be satisfied?

A major drawback with static regions is the possibility that they may become full, hindering player participation. Research has been associated to this problem, with regions been able to spread their processing resources across multiple server nodes if required. However, this may be more process intensive and, therefore, time consuming to achieve than simply allowing inter-server message passing in the first place and distributing load on a per-player basis between servers [78]. If this is the case, what trade-off must be made between the provision of a free roaming virtual world that players enjoy and the strictly regulated transition of region boundaries?

4.2 Who to tell?

Assuming the non-trivial problem of determining where all players are in a virtual world is solved, identifying which events should be propagated to which players needs to be addressed. Once a single oracle type service that identifies interaction in a timely manner exists, this information then has to be implemented using some (approximated) group membership protocol to ensure messages may then be disseminated appropriately across nodes in a network to allow interaction. The less accurate the group membership protocol is the more needless messages will be sent. However, the more accurate the protocol is the more time, and message passing, will be required to achieve identification of message recipients. During runtime, how can such a trade-off be monitored and tailored to guarantee that player expectations associated to gaming scenarios are to be achieved?

The simple solution would be to send all messages to all artefacts within a particular region or to send messages to all artefacts that a player may influence. Unfortunately, this solution is not ideal, especially if a large number of players are present and player consoles receive messages that they are simply not interested in. Witnessing players move around a virtual world is required to aid immersion, but just because a player can view other players does not necessarily indicate that such a player is interested in all events generated by visible players. Therefore, one may envisage that all messages are not to be treated the same. For example, assume a player, say P_1, generates two events, say E_1 and E_2, and another player, say P_2, is only interested in E_2 but not E_1. A group based system modelling this approach will require two distinct groups (message dissemination of E_1 and E_2 is handled separately). Add another player, say P_3, who is interested in E_1 and E_2 and not only do we have another group, but there exists a causal relationship that may be maintained for P_3's view of P_1's actions. This may be modelled via overlapping groups, but the more groups we add the more the processing burden increases and the more time is used up determining message recipients. At what point does group management become so burdensome as to hinder interactivity?

4.3 How to inform?

Consider a virtual world within which the mechanism of realising player locations has been achieved and the identification of suitable message recipients accomplished both in a timely manner. All that is left is to enact message delivery in a manner that satisfies the ordering and reliability guarantees that satisfy the desired player interaction requirements. Such ordering and reliability guarantees will be based on the relevance of messages, and their associated events, to players. For example, intricate interaction between two players will require sufficient ordering and reliability guarantees to ensure game play scenarios progress appropriately. A third player may still be interested in viewing this progression in game play, but may be indifferent to the actual details, possibly requiring summary type information using aggregated messages (e.g., which player won a particular battle). If this is the case then how are aggregated messages related to the ordering and delivery guarantees of the messages they represent? What type of protocol could manage such relationships between messages when different recipients have differing ordering and reliability requirements for the same set of messages?

Treating all messages with the strongest deliverable and reliability guarantees possible has been shown to be problematic when attempting to achieve scalability (ISIS in DIVE); yet modern commercial persistent virtual worlds require something similar for modelling intricate game play. The existence of a server can ease the ordering burden (utilising asymmetric approaches), but such a server must provide some form of failover to ensure a robust environment. An added complication occurs when attempting to avoid resource exhaustion, requiring multiple servers to satisfy message ordering and reliability guarantees to model a single gaming scenario. How does one balance load efficiently yet minimise time consuming message delays that are the result of spreading load over multiple servers? How can failover be achieved while ensuring real-time requirements are satisfied?

5. Conclusions and Further Work

Engineering a scalable virtual world is a non-trivial task that requires a broad range of skills from different areas of computing science. Although commercial virtual worlds exist and have been successful (accounting for over $1 billion in revenue in the USA and Europe by 2006, not including Asia [90]); these worlds can become ultimately more successful. This statement is made as the research accomplished so far, although admirable, needs to expand and become inclusive of a number of fields of computing science.

In the first part of this chapter gaming scenarios are described using the common model for identifying progression in a distributed computation. This provides a convenient and readily understandable description of possible gaming scenarios. A number of errors in gaming scenarios were highlighted that could occur if progression of a distributed computation is not regulated in some way. An attempt is made to relate the problems in gaming scenarios to the more general problems found in distributed computation. This allows a reasoned discussion on what is and what is not possible when developing online games and the possible research direction for addressing the problem of ensuring appropriate gaming scenarios are achieved. This highlights the advances made in the distributed systems community as worthy of serious scrutiny from the online games developer when attempting to create appropriate gaming scenarios for large scale virtual worlds.

In related work a number of academic, military and commercial efforts are listed that have made progress towards the current-state-of-art for scalable online gaming. Works are described that may provide the essence of a number of solutions for advancing the state-of-the-art of scalable online games. This is an important issue to address, as there are a number of research efforts that can make significant contributions to the development of large scale, highly interactive, virtual worlds yet are rarely considered in the gaming literature.

After considering related work, a section is provided that attempts to highlight a number of significant issues to be addressed if advancement in large scale, highly interactive, virtual worlds is to be made. This is represented in a simplified, three stepped approach that appears obvious at first glance, but conceals non-trivial problems that provide a focus for the online gaming researcher. A series of questions are posed without answers to bring to the fore a number of research issues. However, these questions are not exhaustive and may find their solutions by combining solutions highlighted in the related work section of this chapter.

This chapter is now concluded with a number of research problems that go beyond the basic problem of scalability while maintaining highly involving gaming scenarios.

5.1 Advanced interest management

Identifying which artefacts are interacting is a non-trivial problem, however, one can envisage ever more elaborate schemes for actually defining and describing the influence and interest associated to such interaction. Interaction in the real world may consist of a highly complex series of events and creates complex relationships between artefacts which may last sometime (e.g., parcel in plane makes plane heavier). Describing the manner of interaction between participants requires a language capable of expressing a variety of techniques. Although some languages have been proposed (e.g., [91]) they tend to be limited in their expressiveness [28]. Existing solutions use fixed interaction patterns (server based or uses direct communications between user consoles). Varying this choice at runtime has never been considered. Judiciously exercising a choice regarding such patterns may be the key to achieving interactivity and scalability.

To accommodate a wide variety of interaction requirements, an interest management scheme must combine location and discovery services with interaction techniques from a variety of gaming genres:

- ***Discovery*** – given the scale of a virtual world there is a need to provide users with the ability to find scenarios they wish to participate in.
- ***Abstract*** – allow users to exert far reaching degrees of influence on a virtual world, say moving an army of 20,000 soldiers, via minimal effort (e.g., a few mouse clicks).
- ***Realistic*** – to heighten the sense of realism users interact with the each other in a manner similar to that of the real world.

Envisage combining the expressions of interest exhibited in discovery, abstract and realistic interaction into a single interest management solution. For example a discovery service may utilise knowledge of realistic and abstract interaction services to allow a player to locate an appropriate gaming scenario (e.g., finding communities of players collaborating on a task). Developing a single interest management solution for abstract and realistic services will provide a highly interactive virtual world for participants and raises interesting questions. For example, how does a single player's management of a whole city (abstract services) influence the realistic services supporting interacting players inhabiting such a city?

The area of research concerned with scalable *message oriented middleware* (MOM) may offer some insight into this challenging problem. In MOM systems messages may be propagated between sender and receiver based on the subject matter of a message, rather than the identity of the sender. To aid in this MOM systems can provide scripting languages that allow receivers to express their interest in particular message types. This approach has already been experimented with [105] [79] with some success. However, apart from these works the tailoring of MOM systems for use in highly interactive large scale virtual worlds is rarely considered.

Work by Minson and Theodoropoulos has tackled the problem of interest management in a novel way and provide further insights into gaining advanced interest management systems of the future. Their work centres on the investigation of event dissemination via push and pull methods of inter-node communications [111]. They show by considering interest management from the "bottom-up", intricacies are present that affect performance that are often hidden when solely concentrating on interest management as purely an in-world problem. They demonstrate their approaches quite successfully via first person shooter architectures using cell division to attain scalability [112].

5.2 Standardisation and inter-organisational issues

With the commercialisation of virtual worlds a practical engineering solution to scalability must consider two additional issues:

- *Inter-organisational* – delivering a commercial solution to end users requires the cooperation of a number of different organisations (e.g., content providers, hosting provision, Internet service providers).
- *Middleware* – to ensure development costs remain acceptable, a commercial solution must be constructed using readily available middleware tools and services (e.g., security, reliability, persistence, scalability).

An example of inter-organisational complexities is highlighted by the diagram in figure 11, taken from IBM [92]: A games software house produces a game that is made available by some hosting entity supported by a number of service providers that cooperate to deliver a gaming experience to players via some, possibly propriety, gaming device. This inter-organisational approach results in service provision that crosses organisational boundaries, requiring the emulation of electronic equivalents of contract based business management practices. *Service level agreements* (SLAs) provide an opportunity to define such contracts in a way that inter-organisational information sharing may be defined, monitored and enforced.

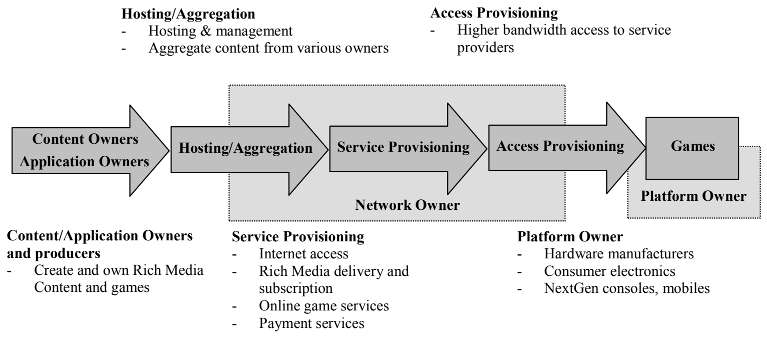


Fig. 11 – Business value chain for online gaming

The cost of developing distributed applications is reduced if existing middleware may be exploited efficiently by a developer. For example, by using an implementation of the J2EE component architecture, say JBoss [93], a developer may engineer a scalable server side application. Services such as transactions, persistence, security, and load balancing may be incorporated into an application by a JBoss application server with configuration guidance from an application programmer.

Inter-organisational and component middleware research has focused on e-commerce client/server style interactions (e.g., stock purchase): there are no J2EE component type architectures for online game developers. Work carried out by IBM to demonstrate Grid technologies does provide introductory work in this area [84] [94] [95] and Sun's Darkstar Project may well provide such a platform in the future [70]. However, the use of SLAs and standard middleware may not be sufficient for modelling advanced virtual worlds.

At the moment virtual worlds are quite disjoint environments. For example, the commercial model dictates that whatever is achieved in World of Warcraft is non transferable to other vendor's sites (vendors do not wish to encourage departures from their own worlds). However, with the advent of standardisation and SLA governed interaction, the future may provide a more unified vision of a virtual world allowing artefacts to be seamlessly transferred between vendors. Ultimately, this may result in a single virtual place where vendors primarily become content providers as opposed to world developers. Artefacts currently have value in commercial virtual worlds, and are regularly traded for real money. However, once the virtual world becomes as accessible and standardised as the modern day Internet then it will be content that will be the most valuable asset.

The ease of access and standardisation of virtual worlds together with associated game engine technologies may yield exciting possibilities. For example, a player may purchase the latest car from one vendor and pay another vendor to race this car around a purpose built arena in a virtual world. Another player may purchase an aircraft and fly over the race track and witness the other player driving their car. Should we let the players interact? Would they want to? Could gaming scenarios grow from disjoint gaming scenarios? Would they make sense? Could they be regulated? Any number of questions may be raised. Linden Labs do allow user derived content development in their Second Life product [55]; however, this is a different proposition to allowing Activision to create the next Gotham Racing in the same virtual world as Rockstar's Grand Theft Auto. Even in the arena of Second Life, where products usually cost less than $1,000 there has been legal issues raised [96]. Consider what legal issues may be raised if a company invested over $30 million (typical cost of top selling game title) in such a world and that investment came to be worth in excess of $400 million (sales value of Grand Theft Auto 4 in first week of release [97]).

5.3 Content Management

To ensure financial success in commercial virtual worlds, player interest must be maintained over prolonged periods of time (measured in years). Therefore, a virtual world must continue to provide new and challenging scenarios to encourage user participation. This can be achieved by periodically introducing new content (e.g., artefacts, rules, stories, areas) and ensuring all content exhibits a degree of persistence to provide a heightened sense of continuing community. In the previous section the discussion centred on the content ownership and management of gaming scenarios created by multiple vendors. However, a more difficult problem may be the actual maintenance and continual improvements to virtual world content. Even now, many commercial virtual worlds are struggling to maintain in excess of 10 to 20 million separate items of content. The prospect of evolving such artefacts and gaming scenarios into ever more elaborate environments seems an insurmountable problem.

Faced with the problem of content management, companies are restricted to manual updates by their own developers or by players. Companies may have good reasons to manage content: coherent storylines and directing the overall look and feel of a gaming scenario. However, this is a burdensome task when millions of artefacts exist. Therefore, an alternative approach has arisen where players are encouraged to create such content, albeit at the expense of a company's ability to direct gaming scenarios [55].

When companies manage content the use of client side updates coupled with additions at the server side is common (e.g., [53] [54]). Updates to client's software are an additional revenue stream for a company. Such updates are achieved by the company releasing "expansion packs" (software updates) which the user must purchase to participate in new gaming scenarios. To ensure existing users may continue to participate without "expansion packs" the company isolates new scenarios from existing content. This is achieved by adding a new area to a virtual world. In reality, existing content is not evolved, but increased in the form of additional areas.

Second Life [55], by Linden Labs, allows player content creation with a financial revenue model based on real-estate and trading: the main type of revenue for Linden Labs relates to the purchase of land and paying of ground rent. An innovative aspect of Second Life is the ability players have for creating content. Such content may then be traded between users. No client side updates are required to access new content (beyond the original downloading of the client game software itself). A scripting language allows artefacts to be instilled with behaviour, allowing players to provide their own virtual world scenarios. This approach provides Second Life players with the most powerful content creation tool available today for online virtual worlds with players providing a wealth of content. Content creation via players has been achieved before Second Life in Active Worlds [115], but it is Second Life's scripting language that provides the dynamic content required to create gaming scenarios. However, even Second Life's approach has its limitations. The following example is used to highlight such limitations.

In a virtual world that already allows players to navigate ships between ports, there is a desire to evolve an economic market by introducing "trade" and "cargo". Once introduced, players will be able to trade between ports via ships carrying cargo. There is a requirement to modify the artefact ship to enable the carrying of cargo. The new concept of trade will require modification to the rules governing the virtual world itself. Ports will assume the role of trade hubs and must be enhanced to recognise their role in trading.

In this example it is not sufficient to just add content, but existing content (ships, ports) must also be altered to enhance them with the ability to participate in trade. This requires updates in the data-store tier (e.g., amount of cargo that a ship can carry) and updates in the application logic tier to enhance functionality (e.g., unload/load cargo) – see figure 9. Furthermore, other artefacts not mentioned in the example must be designated as cargo. This in itself will require updates to other artefacts in the data-store tier (e.g., weight, size, and owner) and additions in the application logic tier (e.g., in transit, set owner, and change value). Finally, the concept of trade itself is quite fundamental and not easily captured within one single artefact, requiring recognition in the rules governing a virtual world (e.g., supply, wealth, and exchange).

The core research problem is the need to ease creation and amendment of existing program code together with changes in persistent data representations to allow far reaching evolutionary change in virtual worlds. This has to be achieved by limiting manual intervention (automating change) without disruption to the virtual world (runtime safe content management).

Existing approaches to company and player derived content evolution can't realise the trading example as existing content cannot be changed appropriately to accommodate new content. In Second Life, propagation of change from one artefact to another is limited and inhibited between artefacts belonging to different owners. Even using such an inhibitive approach Second Life has been plagued by problems (failure of simulation due to erroneous scripts [98]). The more controlled approach used in company driven content change has faired better in terms of virtual world correctness (but failures still happen [99]). This safety has come at the expense of limiting existing content updates to simple bug fixes and only allowing new content distinct from existing content. Fundamentally, all existing approaches severely limit content evolution in favour of safety and the programming burden is immense.

A new code fragment representing an artefact may be manually created. However, the adaptation of the system to accommodate the new artefact should be sufficiently automated to lessen the development burden and ensure safety. One avenue of exploration that may be useful for engineering evolutionary change in virtual worlds is *reflection*. One use of reflection is to allow the self-reorganisation of a system. In essence, reflection could be employed to enable self-reorganisation of code fragments and associated attributes to allow far reaching evolutionary change in a safe manner. Work at Lancaster University in the UK identified the role that reflection may play in online game construction for satisfying scalability, persistence and responsiveness requirements [110]. Reflective middleware platforms may play a significant role in the future of server side virtual world development.

In the end, a virtual world that dates and is unable to keep pace with player expectations will eventually become financially unsustainable. When this occurs, the vendor has no option but to turn the virtual world off.

Acknowledgements

The author gratefully acknowledges members of his team here at Newcastle who have contributed to research related to MMOs over the past few years: Fengyun Lu, Kier Story, Simon Parkin and Dan Martin. Their research was funded by a number of sources, but primarily by the *Engineering and Physical Research Council* (EPSRC) of the UK.

References

[1] Berne, E., "The Games People Play", Ballantine Books, 1979

[2] Chandy, K. M., Lamport, L., “Distributed Snapshots: determining global states of distributed systems”, *ACM Transactions on Computer Systems*, 3(1), 1985, 63 – 75.

[3] Birman, K., “Reliable Distributed Systems: Technologies, Web Services, and Applications”, Springer, 2005

[4] Kshemkalyani, A. D., Singhal, M., “Distributed Computing: Principles, Algorithms, and Systems”, Cambridge Press, 2008

[5] Shatz, S. M., “Communication Mechanism for Programming Distributed Systems”, *IEEE Computer*, 1984, 21-28

[6] Lamport, L., Shostak, R., and Pease, M. 1982. “The Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems*, 4, 3 (Jul. 1982), 382-401

[7] Lamport, L., “Time, Clocks and the ordering of events in a distributed system”, *Communications of the ACM*, 21, 1978, 558-564.

[8] Birman, K., Joseph, T., “Reliable communication in the presence of failure”, *ACM Transactions on Computer Systems*, 5(1) 1987, 47-46

[9] Schlichting, R. D. and Schneider, F. B., “Fail-stop processors: an approach to designing fault-tolerant computing systems”, *ACM Transactions on Computer Systems* 1(3), 1983, 222-238

[10] Schneider, F. B., “Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computer Survey* 22(4), 1990, 299-319

[11] Moser, L. E., Melliar-Smith, P. M., Narasimhan, P., “Consistent object replication in the Eternal system”, *Theory and Practice of Object Systems*, Volume 4 Issue 2, 2000, 81 – 92

[12] Morgan, G.; Shrivastava, S.K., "Implementing flexible object group invocation in networked systems ," *In proceedings on Dependable Systems and Networks*, 2000. (DSN 2000), 2000, 439 – 448

[13] Birman, K. P. “The process group approach to reliable distributed computing”. *Communications of the ACM*, 36,12, 1993, 37-53

[14] Fischer, M. J., Lynch, N. A., and Paterson, M. S., Impossibility of distributed consensus with one faulty process”, *Journal of the ACM*, 32, 2, 1985, 374-382

[15] Chandra, T. D. and Toueg, S., “Unreliable failure detectors for reliable distributed systems”, *Journal of the ACM*, 43, 2, 1996, 225-267

[16] Chandra, T. D., Hadzilacos, V., and Toueg, S., The weakest failure detector for solving consensus. *In Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing* (Vancouver, British Columbia, Canada, 1992. PODC '92. ACM, New York, NY, 147-158

[17] Gray, J., “A Transaction Model”, Lecture Notes in Computer Science, V. 85, Springer Verlag, 1980, 282-298.

[18] Gray, J., “Notes on Database Operating Systems: An Advanced Course”, Springer Verlag, 1979,

[19] Lampson, B. Sturgis, H., “Crash recovery in distributed storage systems”, *Technical Report, Computer Science Laboratory*, Xerox Parc, Palo Alto, 1976

[20] Mostefaoui, A., Raynal, M., “Causal multicast in overlapping groups: towards a low cost approach”, *IRISA/INRIA Research Report* #710, 1993, 13

[21] Dolev, D., Malki, D., and Strong, R., “A framework for partitionable membership service”, *In Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, 1996. PODC '96. ACM, New York, NY, 343

[22] Ezhilchelvan, P. D., Macedo, R., and Shrivastava, S. K., “Newtop: a Fault-Tolerant Group Communication Protocol”, *Technical Report. UMI* Order Number: BROADCAST#TR94-48., University of Bologna

[23] Garcia-Molina, H., “Elections in a Distributed Computing System,” IEEE Trans. Computers, vol. 31, no. 1, 1982, 47-59

[24] Stoller S. D., "Leader Election in Asynchronous Distributed Systems," *IEEE Transactions on Computers*, vol. 49, no. 3, 2000, 283-284

[25] Fich, F. and Ruppert, E., “Hundreds of impossibility results for distributed computing”, *Distributed Computing*, 16, 2-3, 2003, 121-163

[26] Veríssimo, P. and Casimiro, A., “The Timely Computing Base Model and Architecture”, *IEEE Transactions on Computing*, 51, 8, 2002, 916-930

[27] Delporte-Gallet, C., Fauconnier, H., "An example of Real-Time Group Communication System," *In Proceedings of 21st International Conference on Distributed Computing Systems Workshops (ICDCSW '01)*

[28] Singhal S., Zyda, M., “Networked Virtual Environments, Design and Implementation”, Addison Wesley, 1999

[29] Miller, D. C., Thorpe, J. A., “SIMNET: The advent of simulator networking”, *In Proceedings of the IEEE*, volume 8 of 83, 1995, 1114--1123

[30] Zyda, M., Pratt, D., "NPSNET: A 3D visual simulator for virtual world exploration and experimentation", 1991 *SID International Symposium Digest of Technical Papers*, Volume XXII, 1991, 361-364

[31] Neyland, D, L., “Virtual Combat: A guide to distributed interactive simulation”, Stackpole Books, 1997

[32] Hofer, R.C., Loper, M.L., “DIS today [Distributed interactive simulation]”, *In Proceedings of the IEEE*, 83, 8, 1995, 1124-1137

[33] Id Software's Original README.TXT File for Shareware Doom v1.8, can be viewed at: http://www.classicdoom.com/doominfo.htm

[34] Macedonia, M. R., Brutzman, D. P., Zyda, M. J., Pratt, D. R., Barham, P. T., Falby, J., and Locke, J., “NPSNET: a multi-player 3D virtual environment over the Internet”, *In Proceedings of the 1995 Symposium on interactive 3D Graphics* (Monterey, California, United States, April 09 - 12, 1995). SI3D '95. ACM, New York, NY

[35] Singhal, S.K., Cheriton, D.R., "Exploiting Position History for Efficient Remote Rendering in Networked Virtual Reality," Presence, Vol. 4, No. 2, 1995, 169-193

[36] Holbrook, H. W., Singhal, S. K., and Cheriton, D. R. 1995. Log-based receiver-reliable multicast for distributed interactive simulation”, *SIGCOMM Computer Communications,* Rev. 25, 4 (Oct. 1995), 328-341

[37] Hagsand, O. 1996. Interactive Multiuser VEs in the DIVE System. IEEE MultiMedia 3, 1 (Jan. 1996), 30-39

[38] Carlsson, C., and Hagsand, O., "DIVE—A Platform for Multiuser Virtual Environments", *Computers and Graphics*, Vol. 17, No. 6, 1993, pp. 663-669

[39] Birman, K. P. 1986 Isis: a System for Fault-Tolerant Distributed Computing. *Technical Report*. UMI Order Number: TR86-744., Cornell University

[40] Macedonia, M. R. and Zyda, M. J. 1997, "A Taxonomy for Networked Virtual Environments", *IEEE MultiMedia* 4, 1 (Jan. 1997), 48-56

[41] Benford, S. and Fahlén, L. 1993, "A spatial model of interaction in large virtual environments", *In Proceedings of the Third Conference on European Conference on Computer-Supported Cooperative Work* (Milan, Italy, September 13 - 17, 1993). G. de Michelis, C. Simone, and K. Schmidt, Eds. ECSCW. Kluwer Academic Publishers, Norwell, MA, 109-124

[42] Greenhalgh, C. and Benford, S. 1995., "Virtual reality tele-conferencing: implementation and experience", *In Proceedings of the Fourth Conference on European Conference on Computer-Supported Cooperative Work* (Stockholm, Sweden, September 10 - 14, 1995). H. Marmolin, Y. Sundblad, and K. Schmidt, Eds. ECSCW. Kluwer Academic Publishers, Norwell, MA, 165-180

[43] Greenhalgh, C., Purbrick, J., and Snowdon, D. 2000, "Inside MASSIVE-3: flexible support for data consistency and world structuring", *In Proceedings of the Third international Conference on Collaborative Virtual Environments* (San Francisco, California, United States). E. Churchill and M. Reddy, Eds. CVE '00. ACM, New York, NY, 119-127

[44] Benford, S., Greenhalgh, C., Rodden, T., and Pycock, J. 2001, "Collaborative virtual environments", *Communications of the ACM* 44, 7 (Jul. 2001), 79-85

[45] Barrus, J.W., Waters, R.C. and Anderson, D.B., "Locales: Supporting large multiuser virtual environments", *IEEE Computer Graphrics and Applications*, 16, 6 (Nov.1997), 50–57

[46] Barrus, J. W., Waters, R. C., and Anderson, D. B. 1996. "Locales and Beacons: Efficient and Precise Support for Large Multi-User Virtual Environments", *In Proceedings of the 1996 Virtual Reality Annual international Symposium* (VRAIS 96) (March 30 - April 03, 1996). VRAIS. IEEE Computer Society, Washington, DC, 204

[47] Singh, G., Serra, L., Png, W., Wong, A., and Ng, H. 1995, "BrickNet: sharing object behaviors on the Net", *In Proceedings of the Virtual Reality Annual international Symposium* (Vrais'95) (March 11 - 15, 1995). VRAIS. IEEE Computer Society, Washington, DC, 19

[48] World of Warcraft Site, http://www.worldofwarcraft.com

[49] Star Wars Galaxies Site, http://starwarsgalaxies.station.sony.com/

[50] City of Heroes Site, http://www.cityofheroes.com/

[51] Eve online Site, http://www.eve-online.com/

[52] Everquest Site, http://everquest.station.sony.com/

[53] Trials of Obi-Wan Site, http://starwarsgalaxies.station.sony.com/trialsofobiwan/

[54] Wrath of the Lich King Site, http://www.worldofwarcraft.com/wrath/

[55] Linden Lab, Second Life Site, http://secondlife.com/

[56] Castronova, E., “Synthetic Worlds: The Business and Culture of Online Games”, *University of Chicago Press*, 2007

[57] Anecdotal evidence from Computer and Video Game Survey.com, *http://www.video-games-survey.com/online_gamers.htm*, 2008

[58] Kushner, D., “Engineering Everquest”, *IEEE Spectrum*, April 2005

[59] Tyrer, H. W., “Advances in Distributed and Parallel Processing: System Paradigms and Methods”, *Ablex Publishing*, 1994

[60] Bryant, R. E., “Simulation of Packet Communication Architecture Computer Systems”, *Technical Report. UMI Order Number: TR-188., Massachusetts Institute of Technology*, 1977

[61] Chandy, K.M. Misra, J., “Distributed Simulation: A Case Study in Design and Verification of Distributed Programs”, *IEEE Transactions on Software Engineering*, Volume: SE-5, 1979, Issue: 5, 440- 452

[62] Wang, X., Turner, S. J., Low, M. Y., and Gan, B. P., “Optimistic Synchronization in HLA-Based Distributed Simulation”, *Simulation* 81, 4, 279-291, 2005

[63] Jefferson, D. R., “Virtual time”, *ACM Transactions on Programming Languages and Syst*ems, 7, 3, 404-425, 1985

[64] Roccetti, M., Ferretti, S., and Palazzi, C. E., “The Brave New World of Multiplayer Online Games: Synchronization Issues with Smart Solutions”, *In Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC)* - Volume 00, May 2008, IEEE Computer Society, 587-592

[65] R. Yavatkar, "MCP: A protocol for coordination and temporal synchronization in multimedia collaborative applications," *in Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, 1992, 12, 606--613,

[66] Lamport, L., “Using Time Instead of Timeout for Fault-Tolerant Distributed Systems”, *ACM Transactions on Programming Languages and Systems,* 6, 2, 254-280, 1984

[67] Veríssimo, P., “Causal delivery protocols in real-time systems: a generic model”, *Real-Time Systems*, 10, 1, 45-73, 1996

[68] Dahmann, J. S., Fujimoto, R. M., and Weatherly, R. M., “The Department of Defense High Level Architecture”, *In Proceedings of the 29th Conference on Winter Simulation, Atlanta, Georgia, United States, December 1997, S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, Eds. Winter Simulation Conference. IEEE Computer Society*, Washington, DC, 142-149

[69] Terdiman, D., “'World of Warcraft' battles server problems”, *CNET News.com, Published: April 2006,* http://news.cnet.com/World-of-Warcraft-battles-server-problems/2100-1043_3-6063990.html

[70] Sun Microsystems “Project Darkstar” *Web Site, http://www.projectdarkstar.com/*

[71] Cristian F., Aghili H., Strong R., "Atomic broadcast from simple message diffusion to Byzantine agreement", *International Symposium on Fault-Tolerant Computing (FTCS),* June 1985, 15, Ann Arbor, Michigan, USA, p. 200-206

[72] Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C., and Zainlinger, R. 1989, "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach", *IEEE Micro* 9, 1, Jan. 1989, 25-40

[73] Zuberi, K. M. and Shin, K. G. 1996. "A causal message ordering scheme for distributed embedded real-time systems", *In Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS '96), October 1996,* SRDS. IEEE Computer Society, Washington, DC, 210

[74] Roberts, D. J., Strassner, J., Worthington, B. G., & Sharkey, P. (1999), "Influence of the supporting protocol on the latencies induced by concurrency control within a large scale multi user distributed virtual reality system", *International Conference on Virtual Worlds and Simulation (VWSIM), SCS Western Multi-conference'99*, 1999, 31, 70-75

[75] Zhou, S., Cai, W., Turner, S. J., Lee, B., and Wei, J. 2007, "Critical causal order of events in distributed virtual environments", *ACM Transactions on Multimedia Computing and Communications*, Appl. 3, 3, August 2007, 15

[76] Rosedale, P., and Ondrejka, C., "Enabling player-created online worlds with grid computing and streaming", *Gamasutra*, September 2003.

[77] De Vleeschauwer, B., Van Den Bossche, B., Verdickt, T., De Turck, F., Dhoedt, B., and Demeester, P. 2005 "Dynamic microcell assignment for massively multiplayer online gaming", *In Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support For Games, Hawthorne, NY,* October 2005, *NetGames '05. ACM*, New York, NY, 1-7

[78] Lu, F., Parkin, S., and Morgan, G. 2006, "Load balancing for massively multiplayer online games", *In Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support For Games, Singapore, October 2006, NetGames '06. ACM*, New York, NY

[79] Morgan, G., Lu, F., and Storey, K. 2005, "Interest management middleware for networked games", *In Proceedings of the 2005 Symposium on interactive 3D Graphics and Games (Washington, District of Columbia, April 2005, I3D '05. ACM*, New York, NY, 57-64

[80] Vleeschauwer, B., et al. "Network and System Support for Games", *In Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, *NetGames '05*, 2005, Hawthorne, NY, 1 – 7,

[81] Das, T. K., Singh, G., Mitchell, A., Kumar, P. S., McGee, K. "NetEffect: a network architecture for large-scale multi-user virtual worlds", *In Proceedings of the ACM Symposium on Virtual Reality Software and Technology (Lausanne, Switzerland). VRST '97*. ACM, New York, NY, 157-163

[82] Hori, M., Iseri, T., Fujikawa, K., Shimojo, S., Miyahara, H., "Scalability issues of dynamic space management for multiple-server networked virtual environments", *In proceedings of IEEE Pacific Rim Conference on Communications, Computers and signal Processing*, (PACRIM), August 200, 2001, 1:200--203,

[83] Chim, J., Lau, R., Leong, H.V., Antonio Si, "CyberWalk: A Web-based Distributed Virtual Walkthrough Environment", *IEEE Transactions on Multimedia*, 5(4):503-515, Dec. 2003

[84] Shaikh, A., Sahu, S., Rosu, M.-C, Shea, M., Saha, D., "On Demand Platform for Online Games", *IBM Systems Journal*, Vol 45, No1, 2006

[85] Funkhouser, T. A.,"RING: A Client-Server System for Multi-User Virtual Environments", *Computer Graphics, 1995 SIGGRAPH Symposium on Interactive 3D Graphics,* 1995, pp 85-92, Monterey, CA,

[86] Cronin, E., Kurc, A. R., Filstrup, B., and Jamin, S. "An Efficient Synchronization Mechanism for Mirrored Game Architectures", *Multimedia Tools and Applications, Kluwer,* May. 2004, 23, 1, 7-30

[87] Schiper, A. and Raynal, M. 1996, "From group communication to transactions in distributed systems", *Communications of the ACM*, 39, 4, April 1996, 84-87

[88] Bartle, R., "Early MUD history", *http://www.mud.co.uk/richard/mudhist.htm*

[89] Morningstar, C. and Farmer, F. R. 1991, "The lessons of Lucasfilm's habitat", *In Cyberspace: First Steps, MIT Press, Cambridge, MA*, 273-302

[90] Screen Digest report by Piers Harding-Rolls, "Western World Massively Multiplayer Online Games Market: 2006 Review and Forecasts to 2011", www.screendigest.com

[91] Powel, E. T., Mellon, L., Watson, J. F., Tarbox, G. H., "Joint Precision Strike Demonstration (JPSD) Simulations Architecture", *In Proceedings of 14th Workshop on Standards for the Interoperability of Distributed Simulations*, 1996, 807 – 810, USA,

[92] Sharp, C., IBM, "Middleware to enable new business models in the online games industry", *http://www-106.ibm.com/developerworks/webservices/library/ws-intgame*, as viewed May 2004

[93] Fleury, M., Reverbel F., "The {JBoss} Extensible Server", *ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, 344-373, LNCS, Springer-Verlag

[94] Shaikh, A., Sahu, S., Rosu, M., Shea, M., Saha, D., "Implementation of a Service Platform for On-Line Games", *In Proceedings of Workshop on Network and Systems Support for Games (NetGames), 2004 Workshop*, 2004.

[95] Saha, D., Sahu, S., Shaikh, A., "A Service Platform for On-Line Distributed Games", *In Proceedings of Workshop on Network and Systems Support for Games (NetGames) 2003*, May 2003

[96] Second Life News Centre, Reuters, "Judge rules against 'one-sided' TOS in Bragg lawsuit", May 31, 2007

[97] The BBC, "GTA game 'to break sales records'", *http://news.bbc.co.uk/1/hi/technology/7372736.stm*, 29 April 2008

[98] Linden Lab, "Security and Second Life", *http://blog.secondlife.com/2006/10/09/security-and-second-life/*, viewed August 2007

[99] CNet, “World of Warcraft' battles server problems”, http://news.com.com/World+of+Warcraft+battles+server+problems/2100-1043_3-6063990.html

[100] Carpenter, A., “Applying Risk Analysis To Play-Balance RPGs”, *Gamasutra*

June 11, 2003, http://www.gamasutra.com/features/20030611/carpenter_01.shtml

[101] Zenke, M., “Land of fire: The rise of the tiny MMO”, *Game Developer Magazine, United Business Media*, June/July 2008-

[102] Parkin, S. E., Andras, P. and Morgan, G., “Managing Missed Interactions in Distributed Virtual Environments” *In Proceedings of Virtual Environments 2006: 12th Eurographics Symposium on Virtual Environments*, Portugal, May, 27-3, 2006, Eurographics Association,

[103] Müller, J. and Gorlatch, S. 2006, “Rokkatan: scaling an RTS game design to the massively multiplayer realm”, *Computer Entertainment*, 4, 3, July 2006, 11

[104] Morgan, G. and Storey, K, “Scalable Collision Detection for Massively Multiplayer Online Games”, *In Proceedings of the IEEE 19th International Conference on Advanced Information Networking and Applications (AINA '05)*, 2005,Taiwan Volume 1, 873-878, IEEE Computer Society

[105] Bharambe, A. R., Rao, S., and Seshan, S. 2002. “Mercury: a scalable publish-subscribe system for internet games”, *In Proceedings of Workshop on Network and System Support for Games*, Bruanschweig, Germany, April 16 - 17, 2002, NetGames '02. ACM, New York, NY, 3-9

[106] Ferretti, S. and Roccetti, M. 2005, “Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games”, *In Proceedings of the 2005 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology* (Valencia, Spain, June 15 - 17, 2005). ACE '05, vol. 265. ACM, New York, NY, 405-412

[107] Cronin, E., Kurc, A. R., Filstrup, B., Jamin, S., “An Efficient Synchronization Mechanism for Mirrored Game Architectures”, Multimedia Tools and Applications Volume 23, Number 1 / May, 2004, 7-30, Springer Netherlands

[108] Ferretti, S, Roccetti, M., “Fast Delivery of Game Events with an Optimistic Synchronization Mechanism in Massive Multiplayer Online Games”, *Proc. 2nd ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE2005)*, Valencia (Spain), June 2005, 405-412

[109] Cristian, F., "Probabilistic clock synchronization," *Distributed Computin*g, vol. 3, 146-158, 1989, Springer

[110] Okanda, P. and Blair, G. 2003, “The role of structural reflection in distributed Virtual Reality”, *In Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (Osaka, Japan, October 01 - 03, 2003). VRST '03. ACM, New York, NY, 140-149

[111] Minson, R. and Theodoropoulos, G. 2005, “An Adaptive Interest Management Scheme for Distributed Virtual Environments”, *In Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation* (June 01 - 03, 2005). *Workshop on Parallel and Distributed Simulation*. IEEE Computer Society, Washington, DC, 273-281

[112] Minson, R. and Theodoropoulos, G. 2008, "Push-Pull Interest Management for Virtual Worlds", In Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC) - Volume 00, May 2008, IEEE Computer Society, 189-194

[113] Martin, D., van Moorsel, A., Morgan G., 2008, "Efficient Resource Management for Game Server Hosting", *In Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing* (ISORC) - Volume 00, May 2008, IEEE Computer Society

[114] Blizzard Entertainment press release, "World of Warcraft@ Reaches New Milestone: 10 Million Subscribers", *PARIS, France. – 22 January, 2008,* http://eu.blizzard.com/en/press/080122.html,

[115] Active Worlds Site (http://www.activeworlds.com/)