Abstract
This paper is intended to provide a general overview of the emergent field of teleimmersion. We detail some of the major issues facing teleimmersive application developers and suggest seven significant factors that determine how data should be distributed. In particular the impact of data size on delivery of quality-of-service requirements is examined. A flexible model for data sharing in distributed STL-style containers is proposed. In this model containers and iterators are used as abstractions for update propagation and memory consistency models.

1 Introduction
Teleimmersion is a term used to describe networked or distributed cooperative Virtual Environments. Teleimmersion is an emergent area of research, which encompasses issues from networking, graphics, visualisation, databases, real-time systems and human computer interaction. The vision for teleimmersion is to immerse the user in a rich, three-dimensional environment where they may interact with other users, explore large data sets and work with a variety of computational tools. Within such an environment complex interactions should be possible. Participants may collaborate to achieve common objectives, or compete for unique resources to achieve conflicting goals. The potential of such environments is one reason Teleimmersion is seen as a key driver for future computing infrastructure projects, such as the i-GRID [DeFanti and Stevens 1998].

However grand though this vision may be, the reality of teleimmersion currently falls somewhat short. In particular there are significant research challenges to building such applications. The diversity of data and media types required for teleimmersion presents one set of challenges: the real-time requirements of interactivity present another. For distributed computing researchers these two sets of issues may be expressed as follows. First, how to design a single distribution model that is appropriate for the diverse set of teleimmersive data and media types. Second, how to meet the demanding, diverse, sometimes even contradictory quality-of-service requirements of these data types with unreliable, best-effort networks.

Although the techniques of teleimmersion are applicable to a variety of different domains, the outstanding research challenges and emergent nature of the field have so far limited its application. Currently, the principle drivers of teleimmersion are:

- **Scientific Visualisation** – where multiple participants cooperatively explore large data sets, or interactively steer a simulation running on a remote high performance computing (HPC) resource [Papka and Stevens 1996]

- **Cooperative CAD** – where multiple designers and engineers work in a shared space to build virtual prototypes rather than physical models. Cooperative CAD tools are being developed by a number of large vehicle manufacturers, including General Motors and Caterpillar [Lehner and DeFanti 1997]

- **Education** – networked virtual environments provide an obvious vehicle for distance learning applications, or as part of institutional outreach programs. The Narrative Immersive Collaborative Environment (NICE) is a teleimmersion application designed for young children. It places the children in a virtual garden, where they may interact with a range of objects and explore a number of learning themes [Leigh, Johnson and DeFanti 1997]

† The author wishes to acknowledge that this work was carried out within the Cooperative Research Centre for Advanced Computational Systems (ACSys CRC), established under the Australian Federal Government’s Cooperative Research Centres Program
• **Entertainment** – multi-player video games commonly support hundreds and even thousands of players interacting in a shared virtual world. For example, Ultima-Online immerses players in a fantasy world where they undertake a variety of different tasks and may adopt quite sophisticated roles and survival strategies [Fisher and Fraser 1998]

• **Military Training and Simulation** – the armed forces have a long history of using computer simulations to train personnel. They have well defined protocols and architectures for building scalable simulations and virtual battlefields, including the Distributed Interactive Simulation (DIS) protocol [IEEE-1278 1995]. In subsequent sections of this paper we will refer to the example of a military flight simulator, in which multiple human participants fly combat missions against each other, and against computer-controlled (synthetic) pilots.

This paper is intended to provide an overview of teleimmersion and to explore the challenges it presents for traditional approaches to distribution. We begin by presenting a set of factors we believe affect the choice of distribution model for a teleimmersion data type. We then describe our approach to building teleimmersive applications, including a distribution model that uses data sharing to provide low-latency access to data structures. We identify two fundamental issues for shared data distribution, memory consistency model and update propagation protocol, and describe how these issues are managed in our model.

2 The Challenges in Distributing Teleimmersion Data

Teleimmersive applications have been described as “the ultimate synthesis of networking and media technologies” [Smith and Weingarten 1997]. Certainly, a great diversity of media and data types must be combined to produce rich, compelling virtual experiences. Data types commonly used in teleimmersion include: streams of audio and video, 3D geometry, images and textures, haptic information, results from database queries, and results from high performance computing simulations [DeFanti and Stevens 1998]. Each of these data types has different quality-of-service requirements. The challenge for designers of teleimmersive applications is to choose an appropriate distribution model for each data type used by an application.

The quality-of-service requirements of a data type will have a significant effect on the model used to distribute that type [Leigh and Johnson 1998]. Quality-of-service requirements may be described in a variety of ways, but we characterise three different service requirements for teleimmersive data: bandwidth, reliability/consistency, and latency/jitter. Bandwidth represents a raw throughput requirement, and is typically specified in terms of constant and peak bit rates. Many teleimmersive data types have well defined bandwidth requirements and recent network protocols such as ATM and IPv6 provide support for constant bit rate services. Reliability represents the extent to which a data type is robust in the face of packet loss, while consistency is an issue for shared data types. Given a fundamentally unreliable delivery mechanism (such as IP), reliable delivery will always incur additional cost. Data consistency is a significantly more involved issue, and we consider it in detail in section 3.1. Latency represents the delay associated with accessing a data structure, while jitter represents the variation in latency. Data structures used in the presentation and rendering of a virtual environment typically have a high sensitivity to latency and jitter. Latency and jitter are significant problems over unreliable packet switched networks (such as IPv4), which drop packets during periods of high contention. Furthermore, certain distribution models, such as RPC/RMI/ORB, introduce additional latency that may make them unsuitable for some teleimmersive data.

In this section we present a set of factors that contribute to the quality-of-service requirements of a data type, and so affect the choice of distribution model. This section also provides a review of general issues facing teleimmersion developers.

2.1 Model/View distinction

There are fundamentally two separate sets of data maintained in a teleimmersive application. First there is an underlying model of a virtual environment, which represents the state of the application. Second there are representational data structures, which are used to present the environment through one or more display devices. The distinction is essentially that of a model and a view in the familiar Model-View-Controller Design Pattern [Gamma et-al 1995]. An obvious example of this distinction can be seen in the military flight simulator described in the introduction. Two sets of data structures must be maintained for each aircraft present in the simulation: an abstract model which represents the aerodynamic properties of the aircraft, damage, ordinance

1 see section 2.1
The distinction between model and view data is significant for a number of reasons. The limited time available during a rendering cycle, mean that view and model data structures tend to have very different quality-of-service requirements. Visual rendering typically occurs 25 –50 times per second, which translates to a time limit of 20 – 40 milliseconds per frame. Bi-directional audio channels typically require a latency of 100-200 milliseconds\(^2\). Haptic rendering is most demanding of all, and typically occurs about 500 times per second, affording only a few milliseconds per update. Because of these stringent time constraints, most teleimmersive applications decouple the rendering (view) processes from the world update (model) process. This is essential so that a global notion of world time can be maintained for the model, without restricting the frame rates of fast machines. Clearly then, the quality-of-service requirements of view and model data structures are quite different. In general, view data structures are highly sensitive to latency and/or jitter, but do not necessarily require reliability or consistency. Model data structures typically require reliable delivery and strong consistency, but are less sensitive to latency and jitter.

The distinction between model and view is significant for other reasons. Both sets of data must be distributed to all participants, but the mechanisms used may vary significantly. In many applications the view data structures are essentially static. Consequently, they can be distributed to a client machine before it has entered into a teleimmersion session, and may be cached and reused between sessions. Model data structures change constantly and so must be continually updated and redistributed. Furthermore, the view data structures may vary from one client to another: the polygon models used on a low-end PC may be significantly simpler than those used on a high-end visualisation workstation.

Obvious as the model/view distinction may seem, it is often overlooked in teleimmersion literature. This may be due, in part, to a desire to combine model and view to avoid maintaining two separate data structures. Furthermore, certain data structures may be used by both model and view operations. Polygon-based geometry is fundamentally a view data structure, but it is frequently also used to perform accurate collision detection - a model level operation. The Avocado [Tramberend 1999] system takes this approach to its extreme by completely merging all model data into the view’s scene graph. Although this blurring of the lines may simplify the construction of simple teleimmersive applications, we do not believe that it will work for significant applications. Scene graphs were not designed as general-purpose data containers, and should not be used as such.

### 2.2 Data size

Leigh, Johnson and DeFanti [1997] categorise the size of teleimmersion data types as *small, medium or large*. Although these descriptions sound rather simplistic on first reading, they turn out to be remarkably useful and we adopt them without alteration. Small data is anything small enough to be distributed to participants in real

\(^2\) Uni-directional audio channels can typically afford longer delays. This allows time to buffer data to minimise jitter due to network variation (see section 2.6)
time, or broadcast to all participants without too much concern for its relevance. Medium sized data is too large to be distributed indiscriminately, but still small enough to fit within the memory of a user display machine. Large data is beyond the memory capacity of a display machine, and is typically maintained on a remote server such as a mass-storage facility. Small data lends itself to distribution by multicast-style mechanisms. Access to large data usually requires a tiling or partitioning strategy with hierarchies of resolution and client caching.

2.3 Access Patterns

The patterns by which machines access a data structure within a teleimmersion session also have a major impact on the distribution model used. We describe access patterns in terms of the number of simultaneous readers and writers of a data structure. Table 1 presents a number of common access patterns with examples of when they might occur in a teleimmersive application and appropriate distribution models. Note that these access patterns may have a significant impact on the consistency mechanisms used to maintain a data structure.

2.4 Computational Cost

Another significant factor determining how, and more importantly where, to distribute a data structure is the computational cost associated with maintaining that structure. The simple task of rendering a three dimensional environment can consume a majority of compute cycles of a client workstation. For complex data structures and sophisticated algorithms it may make more sense to maintain the structure on a single server or high performance compute resource. Conversely, if there is a low cost associated with the structure it may be better to replicate the computation on all client machines. It may even be practical to parallelize the computation across client machines.

Note there is a strong correlation between computational cost and the data size characteristic identified above. Large sized data structures are too costly in space to maintain on client machines; large/complex computations are too costly in time. Small sized data structures are cheap enough to replicate on all machines; small computations are cheap enough to perform on all machines.

2.5 Time dependency and Rate of Change

Certain data types are highly dependent on time and/or have a constant rate of change. Such data types are often naturally distributed as streams. Obvious examples are audio and video media, both of which have constant rates of change and lend themselves to stream representations. A less obvious example is DIS-style object position information, which changes almost constantly and may also be modelled as a stream. The significance of timing dependent and constant change data types is twofold. First, they may have quite significant quality-of-service requirements in terms of latency or jitter. Second, they form a natural constant-bit-rate component of any network connection. By identifying the constant components of network traffic you simplify bandwidth management and quality-of-service delivery.

A related issue is the buffering or queuing requirements of a data type. Streaming technology for time dependent data, such as audio, often uses extensive buffering to try to compensate for burstiness in the underlying network. Leigh, Johnson and DeFanti [1997] noted that queuing is also necessary for data types that require that all elements of a stream be delivered. Buffering is also typically used to ensure reliable delivery over unreliable networks. However, buffering is not a panacea. It dampens the effect of jitter at the cost of added latency, so buffering may not be appropriate if latency is more important than jitter. Buffering may also be unnecessary for robust data types that can be delivered over unreliable networks. DIS-style object positions are an example of rapidly, even constantly, changing data that does not require reliable delivery or buffering.

2.6 Tiling and Spatial Metaphor

The term spatial metaphor describe the representation of a data set in one or more dimensions. For example, a digital terrain model will typically include an explicit three-dimensional coordinate system as part of the basic model, a bitmap image has an implicit representation in two dimensions, while the result of an SQL query on a payroll database has no obvious spatial representation. Most teleimmersive applications place their users in a three dimensional environment. Within this environment the client typically has a limited field-of-view. For

---

3 See section 2.6

4 Some applications offer a two-dimensional interface in addition to, or instead of, the usual three-dimensional interface. The same argument applies for such applications.
data sets with a strong spatial metaphor it is possible to use the client field-of-view as a mechanism for managing the data set.

This technique is particularly significant in two situations: when dealing with large data sets; and when dealing with large numbers of objects. A common technique for managing large data sets is to partition the data into tiles of a more practical size. By distributing only those tiles which appear within a user’s field-of-view, it is possible for an application to reduce a very large data set to a manageable size [Taylor 1997]. However, this reduction is dependent on the spatial orientation of the user’s field-of-view. Hence, tiling is only appropriate for large data sets with a strong spatial metaphor.

The second situation where a strong spatial metaphor is significant is when dealing with large numbers of objects. Current popular on-line computer games may involve thousands of participants [Fisher and Fraser 1998], while future military simulations are expected to involve hundreds of thousands of participants. In such environments it is impractical to keep all machines informed of the position of all objects and all participants at all times. The most common solution is to use area-of-interest management to partition the environment [Abrams, Watsen and Zyda 1997]. By distributing information about only those objects within the current field-of-view, it is possible to reduce the environment to a manageable size. Again, this solution is only appropriate for data structures with some form of spatial metaphor.

3 HADES: A Flexible Model for Data Distribution

The focus of our current work is on exploring flexible models for sharing teleimmersive data among multiple machines. We are developing a framework for teleimmersive applications known as HADES (High-performance Architecture for Distributed Environments). The foundation of this framework is a set of flexible data containers that automatically replicate their contents to all machines sharing the data. A unique feature of HADES is the way the familiar Iterator and Container constructs from the C++ Standard Template Library (STL) [Musser and Saini 1996] are used as abstractions for memory consistency models and update propagation protocols. A HADES container is instantiated in terms of both a contained data type, and an iteration type. By allowing different combinations of container and iterator, a broad range of quality-of-service parameters can be supported. By simply placing an object within a container, the application specifies the quality-of-service parameters used to deliver that object. In this section we consider the mechanisms used to enforce consistent views of shared data, discuss how these mechanisms can be used to represent quality-of-service requirements, and outline the HADES approach.

3.1 Consistency Models and Update Propagation Protocols

Whenever data structures are replicated or copied to multiple computational units, keeping the copies consistent inevitably becomes a problem. This is true at all levels of data sharing, from high level software based Distributed Shared Memories (DSMs) to low level cache coherency on symmetric multi-processors. For systems that attempt to provide a consistent view of shared data there are essentially two issues that need to be considered: determining when two copies of a data structure have become inconsistent, and determining how to restore consistency. We use the term consistency model to refer to the process of determining when consistency must be restored, and the term updated propagation protocol to refer to the mechanism by which consistency information is delivered to all copies.

Perhaps the most intuitively obvious consistency model is known as sequential consistency. Sequential consistency essentially represents the behaviour of memory on a uni-processor. However, although sequential consistency is a simple model of memory behaviour, it is expensive to enforce and precludes many common optimisation techniques [Adve and Gharachorlo 1995]. For this reason sequential consistency is rarely enforced, even at the hardware level. Countless relaxed consistency models have been proposed, each with different engineering trade-offs. All try to minimise the cost of enforcing consistency, while maximising the extent to which data may be shared. We note that a common theme in almost all relaxed consistency models is to use program synchronisation primitives, such as mutexes and semaphores, to define consistency points.

3.2 Update Propagation, Consistency Models and Quality of Service

One point that should be obvious from the proceeding discussion is that different combinations of update propagation protocols and consistency models have different network delivery requirements, and data access overheads. Sequential consistency models typically require reliable delivery mechanisms, and impose a significant latency on data access. Relaxed models, such as Lazy Release Consistency [Keleher 1995], use false
sharing to minimise the latency of accessing data, but typically introduce very large jitter at release consistency 
points. Other consistency models will, inevitably, involve similar trade-offs.

Whatever consistency model is used, access to distributed synchronisation primitives and propagation of 
updates will inevitably introduce latency into a distributed shared memory. However, as noted above, some 
teleimmersive data structures don’t necessarily require strong consistency: low latency and jitter are often more 
significant. In some cases it may even be appropriate to allow the end user to choose the consistency model 
which is used. For example, most web browsers allow the user to explicitly control the cache coherency policy 
(consistency model) used to manage locally cached web pages. By choosing to have pages verified every time 
they are accessed, once per session, or never, a user makes a direct trade-off between data consistency and 
performance. For certain teleimmersion data types, particularly view/presentation structures, a similar trade-off 
may be appropriate.

Ideally then, application developers would like to be able to vary the consistency model applied to each of 
their shared data structures. Fundamentally, the selection of consistency model will be based on the quality-of-
service requirements of a particular data structure. Hence, update propagation protocols and consistency 
models can be seen as abstractions for different sets of service requirements.

3.3 The HADES Model for Data Sharing

The HADES distribution model is based on the container and iterator constructs from the STL. Within the 
STL, containers are used as generic data managers that maintain aggregations of a single data type. Iterators 
are essentially an abstraction for pointers, and serve two purposes: to provide a means of traversing a container, 
and to provide a mechanism for de-referencing individual objects within a container. In HADES we extend 
these constructs to include notions of update propagation and consistency model, respectively.

A HADES container is responsible for both managing a collection of objects, and for propagating changes 
to these objects to all replicas of the container. How a container effects this propagation is a property of the 
container alone. Hence, the container is an abstraction for the update propagation protocol. A range of different 
containers are provided, which support a variety of different update propagation protocols. For example, our 
MulticastContainer propagates changes using simple IP Multicast. This supports very efficient, low-latency 
update propagation, but does not guarantee delivery. Hence the MulticastContainer is only appropriate for data 
types which are robust in the face of packet loss. Other containers are used to represent more sophisticated 
update propagation protocols, at the cost of more significant network overheads.

A HADES iterator is an abstraction for a consistency model. Traditional iterators are used to de-reference 
objects within a container, but HADES iterators are given the added responsibility of managing concurrent 
access to an object. Effectively an iterator becomes the interface to the underlying synchronisation primitive 
used to manage access to an object. An application programmer acquires and releases the synchronisation 
primitive that protects an object, through the same iterator he/she uses to de-reference the object. We call this 
extended primitive the synchronising iterator. As noted above, synchronisation primitives are a fundamental 
part of most relaxed consistency models. Hence a HADES consistency model is simply a synchronising iterator 
which coordinates the updates of the container it iterates. When an iterator determines that an object needs to 
be made consistent, it informs the underlying container, which in turn is responsible for propagating update 
information. Within HADES we provide a range of different iterators which represent different consistency 
models.

As noted above, update propagation and consistency models can be used to represent different quality-of-
service requirements. So, the basis of the HADES distribution model is to use shared containers and 
synchronising iterators as abstractions for different quality-of-service requirements. When an application 
developer needs to distribute a data structure, they simply select an appropriate combination of container and 
iterator that meet their particular service requirements. We believe the simplicity and generality of this model 
are quite compelling.

3.4 A Simple Application

To demonstrate this approach to distributing teleimmersion we are building a simple application that presents a 
static three-dimensional environment, through which multiple participants may move. This application serves 
to demonstrate the basic concepts of teleimmersion, and might be used to provide simple architectural walk-
throughs or guided tours of virtual spaces. A simple HADES MulticastContainer is used to distribute the 
positions of participants within the environment. The container uses IP Multicast to distribute updates and does 
not enforce any consistency model, instead relying on the fact that there is only one writer for each data object. 
The basic object position data structure is robust in the face of packet loss, so the unreliable service provided by
IP Multicast is appropriate. The very simple nature of the application means that it should scale to support very large numbers of participants. Although crude in form, this demonstrator provides an example of how other, more sophisticated HADES applications will be constructed.

4 Conclusion

This paper was intended to provide an overview of the emergent field of teleimmersion. We reviewed a number of factors which affect how teleimmersion data is distributed, and considered how these factors relate to traditional distribution models. We have outlined our approach to building teleimmersive applications, and proposed a simple model for distributing some teleimmersion data types. The basis of our model is to use STL-style containers as shared data spaces, with a flexible set of memory consistency models and update propagation protocols. Although our implementations remain at an early stage, we believe this model provides a flexible architecture for data distribution. We look forward to implementing a variety of different containers and iterators, to provide a rich framework for teleimmersion.

Acknowledgments

The author wishes to acknowledge that this work was carried out within the Cooperative Research Centre for Advanced Computational Systems (ACSys CRC), established under the Australian Federal Government’s Cooperative Research Centres Program.

Bibliography


