

A DEMAND-DRIVEN APPROACH FOR A DISTRIBUTED VIRTUAL ENVIRONMENT

Markus Fakler
S. Frenz, M. Schöttner, P. Schulthess
Department of Distributed Systems
University of Ulm, Germany
e-mail: markus.fakler@uni-ulm.de

Abstract

The relevance of distributed virtual environments (DVE) and interactive 3D worlds for entertaining and commercial purposes is steadily increasing. DVEs are typically implemented as traditional client/server based scenarios in which a central server system controls the state of the distributed data and sends update messages to all clients.

In this paper we describe the benefits of a demand-driven approach for a DVE. A distributed operating system is used to host the DVE application. The common scene graph of the DVE resides in shared memory and offers direct access to the participating nodes – no explicit update messages are needed. The shared memory concept allows for an easier, more intuitive and less fault-prone way of creating distributed applications like a DVE and it offers inherent consistency and efficiency because the nodes do not have to process unneeded update messages.

An existing prototype demonstrates the benefits resulting from this concept.

Keywords: *Virtual Reality & New Media, Common Scene Graph, Transactional Consistency, Computer Networks & Systems*

1. Introduction

The relevance of distributed virtual environments (DVE) used for multi-player games, virtual reality chat systems and interactive 3D worlds is steadily increasing for entertaining, educational and commercial purposes. Most DVEs are still realised within a traditional client/server based scenario and within standalone operating systems. Network communication between clients is performed by specialised middleware components and often the distributed application must explicitly manage synchronisation and consistency of the rendered scene. Intricate frameworks are advocated to handle these consistency issues but all to often they fail to simplify the task of programming the distributed applications.

We argue that using a distributed operating system as a foundation for distributed applications offers significant advantages. It can offer implicit synchronisation and consistency of the distributed data, thus reducing the complexity for the programmer and therefore providing a more intuitive and less error-prone way of writing distributed applications.

To demonstrate the benefits of this concept we have

implemented an interactive virtual 3D world on top of a distributed operating system.

In section two of this paper we describe the structure of our common scene graph. Section three and four present the advantages gained from the demand-driven approach and section five introduce our implemented prototype. In section six we characterise the underlying distributed operating system and the subsequent sections discuss related work and our conclusions to complete this paper.

2. Common Scene Graph

In traditional multi-user environments only a server or a server cluster (hereinafter just 'server') is allowed to directly modify and control the scene graph describing the layout of the virtual environment. In doing so it is possible that the server is situated on the same machine as a client or that different parts of the scene graph are controlled by different servers. However, the single parts of the scene graph (or even the whole scene graph) are controlled typically by only one explicit and responsible server instance respectively. The server then receives modification requests by the clients, must interpret these messages, modify the scene graph accordingly and send update messages back to all connected clients (figure 1). For this purpose the server has also to maintain a list of the participating clients and further administrative data.

To speed up the rendering process the clients keep partial copies of the scene graph in local memory. If a client wishes to alter some objects it will ask the server to do the modification for it. In a second step the client will then receive the updated objects in the scene. Thus the control of objects is only possible in an indirect way for the clients and can take some time depending on the workload of the server.

In our demand-driven approach we use an advanced type of

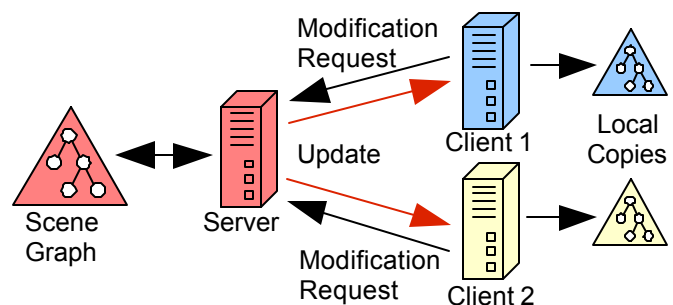


Figure 1: Traditional client/server architecture for controlling the scene graph

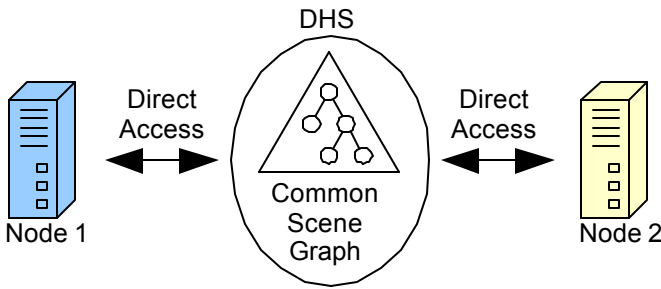


Figure 2: Common scene graph within a distributed heap storage (DHS)

shared memory provided by the underlying distributed operating system. We have placed the scene graph completely into the shared memory area making it a common scene graph and thus enabling every node in the cluster to directly access it for modifications (figure 2). This means if a node needs to change some data belonging to the scene graph (for instance the position of an avatar which is controlled by this node) it can immediately write the new and actualised data into the common scene graph without first sending the appropriate modification request to the server and then waiting until the update response will be received.

The consistency of the objects in the common scene graph is guaranteed by a transactional consistency protocol common to all stations (see section 6.2). Possible accessing conflicts when two or more nodes wish to update the same data at the same time are solved completely transparent by the operating system.

The availability of a consistent distributed heap storage facility frees the programmer also from the necessity to distinguish between local and shared data. Every object will appear in the common scene graph only once and by default it is shareable. We believe that accessing data instead of formatting and receiving read/write messages is an easier and more intuitive way of creating distributed applications and the programmer can now concentrate on the primary task of writing a distributed application instead of being distracted by complex synchronisation and communication issues which need attention and make the whole task more error-prone.

3. Demand-Driven Approach

Our demand-driven approach of the DVE is consequence of combining the direct accessibility of the common scene graph by all nodes and the transactional consistency offered by the underlying memory and operating system. A node reads and writes the desired data from the scene graph without placing locks and without special consideration. No explicit server instance to control a certain part of the scene graph is needed and therefore no update messages are sent, broadcast, received and interpreted just to realise perhaps that the update message was only relevant for another node. Using the demand-driven approach a node accesses the data only then when it is going to use it.

The shared memory concept lets the nodes work without explicit replication and maintenance of local copies of the scene graph. Even during the rendering process when a node is

about to send for example the shape of an object to the graphics adapter it accesses the data directly from the shared scene graph. The transactional consistency ensures that every access into the scene graph will always return the most actual data - no further consistency issues need to be considered by the application.

4. Dedicated Computing

In a client/server based scenario the clients will receive the update messages only at certain intervals or at modification points on the server side. But a user of a DVE will not accept that for instance a computer controlled avatar or an animation visualised on its client pauses just because the update message with the next position of the animated objects is not yet received. Traditionally each client will calculate these animations on its own by extrapolating the new position of the animated objects from the last update messages. Depending on how long ago the last update message has arrived a drift between the extrapolated data and the next update message when received may occur. This possible drift might have to be considered and corrected by the client (e.g. via dead reckoning algorithms [1]). This whole procedure will be expensive as every single animation is adjusted on every participating client resulting in a tremendous effort and computational cost for animations and computer controlled objects within a DVE.

With the utilisation of a shared common scene graph only one single node needs to calculate an animation for the whole DVE. The results will again reside in the scene graph allowing all other participating nodes to access directly the actualised positions of the modified objects when they are in view and need to render it.

An additional advantage of this concept is that the calculation of an animation or other computer controlled objects must not be done on a certain node but can be done by just any one node having enough resources left to do this computation. This allows the use of dedicated nodes not participating explicitly in the DVE but only offering their computational resources within the cluster. So if it is desired to have ever more complex animations within the DVE it is easily possible to add additional nodes and the rest of the cluster will benefit from the extended resources (figure 3). Even the migration of such computations from nodes with high workload to other nodes with more free computational resources is possible.

Conversely this concept additionally allows older or less

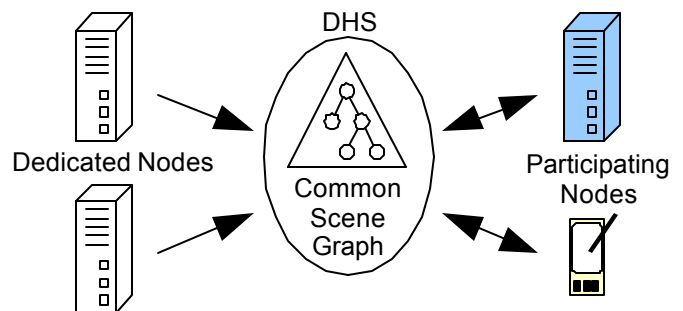


Figure 3: Adding dedicated nodes for additional computing (e.g. animations)

powerful hardware (like a PDA or mobile phone) to participate in the DVE. The basic requirement for these devices is merely that they must visualise the data during their render process in adequate time. All other computation will be done by faster nodes or dedicated computing nodes.

5. Wissenheim – A DVE prototype

We have built a prototype of an interactive 3D virtual environment called 'Wissenheim' [2]. It is running on top of a Plurix cluster implementing the described concepts. Wissenheim is intended as a virtual presence platform in which students and teachers from our university have the possibility to meet and discuss and also to deepen the learned content of the lectures. It is already used for teaching purpose and student work. For this reason we have visualised some educational content of the lectures by means of 3D animated objects and scenes and they are now running within Wissenheim.

To further demonstrate the full capabilities of our prototype and to make our virtual world more dynamic we have also created some autonomous virtual agents within it [3] like they are known from the area of artificial intelligence. These autonomous agents can act inside our world, evade obstacles in their way and can develop in a limited way. They use a set of (virtual) sensors and actuators to orient and move within Wissenheim and behave by the computed results of a neuronal controller network immanent to each agent.

This work was used to test an evolutionary approach for a self-organising population. The computational effort needed for each agents is much higher than for normal animations. Our DVE platform offered the possibility to easily combine the visualisation of the behaviour of these agents during their further development along with providing enough computational resources to do the needed calculations for their behaviour. Thus this work benefits more from dedicated compute nodes than normal animations do and is therefore a good example for the advantages gained from our concept.

6. Plurix - A distributed operating system

As a base for our prototype and further distributed applications we use the Plurix® distributed operating system. The Plurix project [4] implements a lean operating system for PC clusters. It is working in a fully object-oriented way and it is written almost entirely in Java. This enables Plurix to utilise the type-safety of Java and offers along with it a widely spread and known high-level programming language (even on the system layer) for the programmer. The proprietary Plurix Java Compiler (PJC) compiles the Plurix system and the applications directly into native machine code thus enabling the system to use the full power of the installed hardware.

6.1 Memory Model

Plurix offers an advanced variant of distributed shared memory implemented as distributed heap storage (DHS) [5]. The DHS offers to all nodes within the cluster the same view

of the shared objects and thus implements a single-system-image (SSI). Communication between nodes is implicit and can be easily achieved by placing data into the DHS to be possibly accessed by any other node. A global name-service is used for publishing new objects to the cluster. The distribution of objects from one node to another when accessed is done automatically by the memory management system.

Persistence within the Plurix system is offered by an optional page-server taking incremental checkpoints and accomplishing automatic fallbacks in case of a failure in less than a second [6].

6.2 Consistency Model

To achieve deterministic computation Plurix introduces a novel consistency model named transactional consistency [7], i.e. all calculation done by a node is encapsulated within a transaction (TA) much like it is known from database systems. When a TA finishes and wants to commit its modified data, it is validated against all other nodes in the cluster. If no other node wants to modify the same data at the same time, the changes can directly be accepted. Otherwise if there is a conflict between multiple modifying nodes on the same data Plurix solves this collision with a first wins strategy [7]. The first TA committing is always allowed to propagate its modifications into the DHS. Other involved TAs will get aborted and restarted with the now actualised data. This is done by the Plurix system and is transparent for the user and also for the affected TAs. The only noticeable effect is a slight delay before these TAs finish. As a consequence all calculations done by the TAs are performed with the newest versions of the used data.

This makes Plurix an ideal platform not only for distributed parallel applications but also for applications with an inherent need for heavy communication and therefore synchronisation and consistency requirements like those of a DVE [8].

7. Related Work

Traditionally DVEs are implemented as classic client/server based scenarios. The main differences to shared memory systems have already been described within this paper. Other systems implementing a shared memory system have almost without exception specialised on scientific computing and are therefore not fully suitable for the realisation of a DVE with its implicit communication needs.

A large number of formats and architectures for scene graphs have already been proposed. Among the more popular ones are VRML and Java-3D. However, these are unfortunately geared to single station systems and lack the possibility to be easily expanded into a distributed design.

There are other approaches to implement DVEs (Avocado [9], DIVE [10], Repo-3D [11], ...) or distributed scene graphs (blue-c Distributed Scene Graph [12]) which often use the term „common scene graph” (like Distributed Open Inventor [13]) but they either use replicated scene graphs on the participating nodes according to some relaxed consistency models or again use explicit update messages to

achieve synchronisation between parts of the scene graph. Therefore these concepts are not completely suitable for use with a shared common scene graph scenario and can not benefit from the use of a distributed memory system offering transactional consistency like in Plurix.

8. Conclusions

Our research shows that the use of a DHS is an elegant and viable concept for distributed applications with implicit communication requirements like for instance a DVE. The placement of the scene graph inside the DHS reduces the complexity of the applications and provides a more intuitive and less error-prone approach to designing and writing such applications.

The concept of a common scene graph with direct accessibility from all participating nodes offers benefits such as avoiding redundant computation and scalability by adding additional compute nodes.

The use of a distributed operating system as a foundation for distributed applications also reduces the burden on the programmer. He no longer needs to worry about synchronisation and consistency.

Therefore we suggest that our demand-driven approach offers an interesting alternative to the traditional client/server based architectures currently in use.

8.1 Future work

Currently there is further work to do for our prototype Wissenheim. We have already revised the first implementation and are now going to finish the adaption of the existing content (like the autonomous agents, [3]). Afterwards we are again able to do extended measurements and to show significant and actual data. The results of these measurements will be presented in a following paper. We are also going to expand our DVE and make more educational content available to provide a whole interactive lecture inside Wissenheim.

References

- [1] L. Pantel, L. C. Wolf, "On the suitability of dead reckoning schemes for games", in: Proceedings of the 1st workshop on Network and system support for games, Braunschweig, Germany, 2002
- [2] Project Wissenheim, www.wissenheim.de
- [3] P. Pichler, "Populating virtual worlds: an evolutionary approach", Diploma Thesis, University of Ulm, 2005
- [4] The PLURIX Project, www.plurix.de
- [5] R. Goeckelmann, "Speicherverwaltung und Bootstrategien für ein Betriebssystem mit transaktionalem verteilten Heap", Dissertation, Universität Ulm, Abteilung Verteilte Systeme, 2006
- [6] M. Schoettner, S. Frenz, R. Goeckelmann, P. Schulthess, "Checkpointing and Recovery in a transaction-based DSM Operating System", in: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks, Innsbruck, Austria, 2004
- [7] M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer, P. Schulthess, "Optimistic Synchronization and Transactional Consistency", in: Proceedings of the 4th International Workshop on Software Distributed Shared Memory, Berlin, Germany, 2002
- [8] M. Schoettner, M. Wende, R. Goeckelmann, U. Schmid, P. Schulthess, "A Gaming Framework for a Transactional DSM System", in: Proceedings of the Workshop on Distributed Shared Memory on Clusters, IEEE International Symposium on Cluster Computing and the Grid, Tokyo, Japan, 2003
- [9] H. Tramberend, "Avocado: A Distributed Virtual Environment Framework", Dissertation, Universität Bielefeld, Technische Fakultät, 2003
- [10] C. Carlsson, O. Hagsand, "DIVE - A Platform for Multi-user Virtual Environments", *Computer and Graphics*, 17(6), pp. 663-669, 1996
- [11] B. MacIntyre, S. Feiner, "Repo-3D - A Distributed 3D Graphics Library", in: Proceedings of the ACM SIGGRAPH '98, Orlando, Florida, 1998
- [12] Martin Naef, Edouard Lamboray, Oliver Staadt, Markus Gross, "The blue-c Distributed Scene Graph", in: Proceedings of the IPT/EGVE Workshop, Zurich, Switzerland, 2003
- [13] G. Hesina, D. Schmalstieg, A. Fuhrmann, W. Purgathofer, "Distributed Open Inventor: A Practical Approach to Distributed 3D Graphics", in: Proceedings of the ACM Symposium on Virtual Reality Software and Technology, Vienna University of Technology, 1999