Distribution and Partitioning Techniques for NVEs: the case of EVE

Christos Bouras, Eri Giannaka, Alexandros Panagopoulos, Thrasyvoulos Tsiatsos

***Abstract*—The majority of the systems and platforms developed for supporting Distributed Virtual Environments are based on the concept of distribution from the early beginning of their development. In this paper we present the migration to a distributed virtual environment from a traditional client-server architecture. In particular, this paper describes the case of EVE, a Networked Virtual Environment originally aimed to support small-scale applications. EVE started as a standard client-multi server architecture, which could support multiple concurrent virtual worlds with a maximum number of seventeen simultaneous participants in each of these worlds. However, the need to support larger-scale applications revealed that the traditional architecture, upon which EVE was based, is insufficient to meet the needs of these applications, which are large both in the sense of virtual space and graphics and in regard to the number of concurrent participants. This paper discusses the migration of EVE to a distributed platform, which will be able to support large-scale networked virtual environments. In particular, the paper describes the modifications realized in the architectural model of the initial platform for supporting effectively large-scale applications. The basic entities of the distributed model are presented, their operations, as well as the interconnection among them. In addition, the paper presents an initial approach of the algorithm that will be adopted for the efficient partitioning of the virtual world and the assignment of the clients to the entities and resources of the distributed platform. The approach presented is space-object driven, in the sense that both the actual size of the virtual space along with the number of objects with which the user can interact is taken into account during the partitioning.**

***Index Terms*—distributed virtual environments, distributed architectural model, partitioning problem**

Manuscript received February 8, 2006.

Christos Bouras is with the Research Academic Computer Technology Institute, Patras, Greece (phone: +30-2610-960375, fax: +30-2610-960358, e-mail: bouras@cti.gr) and the Computer Engineer and Informatics Department, University of Patras, Greece (phone: +30-2610-996951, fax: +30-2610-969016, e-mail: bouras@ceid.upatras.gr).

Eri Giannaka is with the Research Academic Computer Technology Institute, Patras, Greece (phone: +30-2610-960380, fax: +30-2610-960358, e-mail: giannaka@cti.gr) and the Computer Engineer and Informatics Department, University of Patras, Greece.

Alexandros Panagopoulos is with the Computer Science Department, Stony Brook University, New York, USA (e-mail: apanagop@cs.sunysb.edu).

Thrasyvoulos Tsiatsos is with the Research Academic Computer Technology Institute, Patras, Greece (phone: +30-2610-960316, fax: +30-2610-960358, e-mail: tsiatsos@cti.gr).

I. Introduction

Virtual Reality technology has been evolved from a newborn trend for the achievement of a high sense of realism to a technology that has been widely established and used for a variety of applications and areas. The first Networked Virtual Environments (NVEs) originally developed could support a limited number of users and were defined as Small-Scale NVEs. The maturation of the Internet, the familiarization of the users with this means of communication but mainly the improvement of this dynamic means led to the need for the formation of larger NVEs. In particular, the last few years, the research interest turns to the direction of NVEs that could support efficiently a tremendous number of concurrent users. However, the development of Large Scale Networked Virtual Environments introduced a series of problems, which are mainly related to the debility of the network to host and serve the demanding recourses that these applications require. In particular, these environments are accompanied by rich graphics for the representation of the provided information as well as a variety of provided services. These characteristics, in combination to the tremendous number of users that are called to support, result in the exponential increment of the resources needed for their smooth, natural and efficient operation.

Traditional architectures and algorithms adopted for the development and support of, the so-called, small-scale Virtual Environments seem insufficient to be applied to environments with an importantly larger number of users [13]. In particular, small-scale approaches, from an architectural point of view, fall usually into one of the following architectures: a) client-server architectures, and b) peer-to-peer architectures. Both approaches, when scaling into a larger number of users fail to support the virtual environment efficiently as either, the clients, the server, or both fail due to bandwidth deficiency.

The increased research interest on Distributed Virtual Environments resulted in the development of a number of protocols and platforms. In the area of protocols for Distributed Virtual Environments, SIMNET (Simulator Network) [5] constitutes the very first effort to this direction. The protocols that followed were DIS (IEEE 1278) and the SIMNET DIS that were developed for overcoming the limitations that the former SIMNET protocol presented. NPSNET-IV [20] extends the original DIS architecture to

address many of these problems. Open Community (OC) is a proposal of a standard for multi-user enabling technologies. SPLINE is an OC compliant implementation that provides development APIs. MASSIVE-2 (model, architecture and system for spatial interaction in virtual environments) [11] is a prototype developed in 1997 and its major contribution is the introduction of the third-party objects, which allows a hierarchical dynamic space-based embodiment of multicast groups. SCORE was developed in 2000 and is based on the division of the world in cells as suggested by [12]. Finally, VELVET [4] is an adaptive hybrid architecture that allows a greater number of users to interact through a CVE. This is accomplished through an adaptive filtering scheme based on multicasting.

This paper presents the transformation of a NVE called EVE, which was developed for supporting small-scale virtual environments to a flexible and scalable distributed platform. Platform EVE, as originally implemented, could be used for applications with a limited number of concurrent users. Such application was a virtual learning class, where students and tutors could collaborate using the services of the platform (as chat and audio) as well as interact both with the objects of the virtual world and with the other for facilitating the learning procedure. However, for applications with larger virtual worlds and a much more high number of concurrent users the client-server architecture was insufficient. Examples of such applications that require very large virtual worlds for their representation could be a virtual campus with different departments, different classes and offices per department and different number of users or a virtual city, where the users could navigate around the buildings and public services and interact with servants for realizing real-time transactions. The adaptation of the client-server version of EVE for the above examples would fail as the servers would be overloaded and would constitute a bottleneck point for the network. Thus, the challenges that arise for the transformation of the client-server architecture to a distributed one are mainly related to the effective partitioning of the virtual space as well as to the assignment of the objects and clients of the system to the servers available, so as to optimize the networking performance and maintain the consistency of the applications. To this direction, the paper describes the modifications realized in the architectural model of the initial platform for supporting effectively large-scale applications as well as the partitioning approach that will be adopted for the effective assignment of the workload to the available servers and entities of the platform.

The paper is structured as follows: Section 2 presents shortly the architectural model of EVE platform. Section 3 describes the main reasons that lead to the need of distribution. Section 4 introduces the implementation issues for the first steps realized for the transition of the platform. The Section that follows presents the concept of the partitioning algorithm that will be adopted for achieving and maintaining an optimized performance. Section 6 presents some further consideration that need to be taken into account for a distributed virtual environment application as well as the ways that the distributed version of EVE will handle these considerations. Finally, Section 7 concludes the paper and presents the planned next steps for the optimization of the platform.

II. PLATFORM EVE

EVE's architecture has been presented in a series of papers [1]. However for the facilitation of the reader, which might not be aware of EVE we shortly describe its architectural model.

EVE is based on a client-multiserver platform model. This model offers scalability and flexibility to the EVE architecture, because we can add more application servers in order to offer more functionality and furthermore the processing load is distributed among the above set of servers.

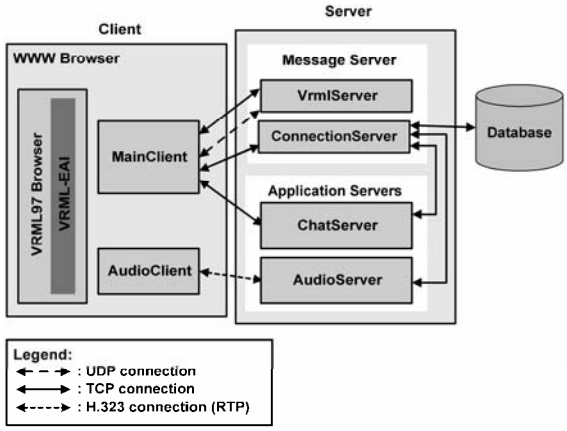

Fig. 1: EVE's client-server architecture

A. Server Side

The servers on which the platform relies, is the message server and two application servers, a chat and an audio server.

1) Message Server

The message server is responsible for the manipulation of the virtual worlds that are visited by the users of the system. In addition, this server creates and supports the illusion to the users that they are participants in the above virtual worlds and that they share a common space by updating the view of the world every time that a shared object is modified. Two servers, each of which is used for a specific sequence of operations, constitute this message server. These servers are the Connection Server and the VRML server.

Connection Server: this server maintains a database, which the system accesses in order to authenticate the user and allow him/her to enter the virtual space of EVE. In addition, the connection server reports every entry or departure that takes place in the platform to all other servers.

VRML server: this server monitors and records every event that takes place in the virtual space and reports these changes to all participant clients of the platform. Thus, by performing these continuous updates the system assures that the users will have the illusion of sharing a common space.

The VMRL server also maintains constantly an updated copy of the world, which is sent to the clients when they enter the system. That way, the new users have the same updated view that the existing users already have.

2) Application Servers

The application servers are responsible for providing specific functionality to the participants of the virtual world. In the current form of EVE there are two application servers available, a chat server and an audio server.

Chat Server: this server is responsible for the text chat support. It allows group chat, which means text chatting between multiple users, or whispering, which allows the one-to-one communication between two users.

Audio Server: this server is responsible for the audio communication between the users of the platform. The audio server uses H.323 as its main protocol.

B. Client Side

As depicted in Fig. 1, in order the users' clients to communicate with EVE's servers and have access to the provided functionalities they need a web browser, a VRML browser, the main EVE client and the audio client.

Web Browser: The web browser is used for the communication with the web server of the system, which provides an initial interface and entry point between the user's client and EVE's environment.

VRML Browser: The 3D environment of EVE is implemented using the VRML language. Therefore, a VRML browser, a plug-in, is essential in order to allow the navigation of the user's avatar in the virtual training space.

Main Client: This client is responsible for (a) the primary connection of the user to the Message Server, (b) the interaction between the user's avatar and the 3D virtual space and (c) the text chat communication between the users of the same virtual space. In particular, the main client, which is a java applet, makes an initial connection to the connection server, which allows it to present the current connection status and when the user is authenticated, it passes on to the vrml server.

III. Towards Distribution

Platform EVE was originally developed for supporting simultaneous small-scale multi-user virtual environments, with a small number of concurrent users (up to 17 for each virtual space). On this version of the platform, which was based on the architecture described in the previous section, a performance monitoring was conducted for evaluating its networking performance. The simulations, experiments and results that were extracted have been presented in [2]. In short, the results indicated that the above-mentioned platform could efficiently support up to two concurrent virtual worlds. However, the platform indicated some limitations in the avatars' movement within the virtual world, in the sense that a small percentage of positioning messages were lost. This was not an important limitation for the previous version of EVE, as the avatars' movement within the virtual space was limited.

The reasons that lead to the need for distribution fall into two directions. The first one arises from the need to make a platform of general use. This implies that the platform should be able to support different kind of applications as games, simulations, etc. In the case of the games and other similar applications, where the probability of the users' movement within the virtual environment is high and based on the results extracted by the performance evaluation conducted, EVE would be unable to serve efficiently such types of applications.

The second reason is the need for a platform that could support large-scale virtual environments. The essence of "large" refers both to the size of the virtual worlds as well as to the number of concurrent users that the platform can support. If we decided to adopt the initial platform for this purpose, and based on the fact that each server can efficiently support up to thirty concurrent users, it becomes clear that the case of thousand of users, where large-scale environments apply, would be completely infeasible, since the centralized components of the old architecture would not be able to handle a large amount of users, no matter what the underlying hardware is.

IV. Distributed Architectural Model

Based on the fact that EVE started as a standard client-multi server architecture, the modification and redesign of the architectural model was the first step to be taken for the process and purpose of distribution. To this direction it became clear that the Message Server (Fig. 1), which was responsible for the critical processes of the EVE platform, could no longer exist with its traditional form. In the following paragraphs we describe the basic components on which the distributed version of our platform relies.

A. Main Entities of the Distributed Model

The main entities on which the distributed platform relies on could be divided in two main categories: the "Kernels", which mainly handle the processes that take place within the virtual environment and the "Kernel Managers", which are mainly related to managing the processes and distributing them over the available machines.

B. Kernels

With the term "kernel" we refer to threads, which are engaged with certain types of processes within the virtual environment. The distributed version of the platform is based on three types of kernels: a) the World Kernel, b) the User Cluster Kernel and finally c) the Parser Kernel. Each kernel is a process responsible for part of the book-keeping, processing or communication load related to a virtual environment. The operations and processes that each of these kernels manages are described in the subsections that follow.

1) World Kernel

The state of each virtual world is stored and maintained by

one or more World Kernels. The World Kernel is a process responsible for the maintenance of the state of the data representing a portion of the virtual environment. These data correspond to the current state of the virtual world and the World Kernel is responsible of handling all state changes related to the portion of the world that it handles, except for the changes of user state, which are handled by the appropriate user cluster.

In particular, this thread implements a simple computation kernel to handle a single part of the virtual world. However, in the current version of the platform, this thread handles an entire world. In future versions of the platform world kernels will handle parts of the virtual space. Furthermore, the World Kernel stores locally the world data and allows for data/event input/output. The basic operations related to this thread are: a) load binary data into the database, b) send binary data describing the whole world, c) send binary data describing a single node, d) apply an event onto the data stored, e) clear the entire database and f) output update events.

As mentioned above, this thread sends out the updates of this world to the clients, while spatial filtering of the outgoing events is also performed.

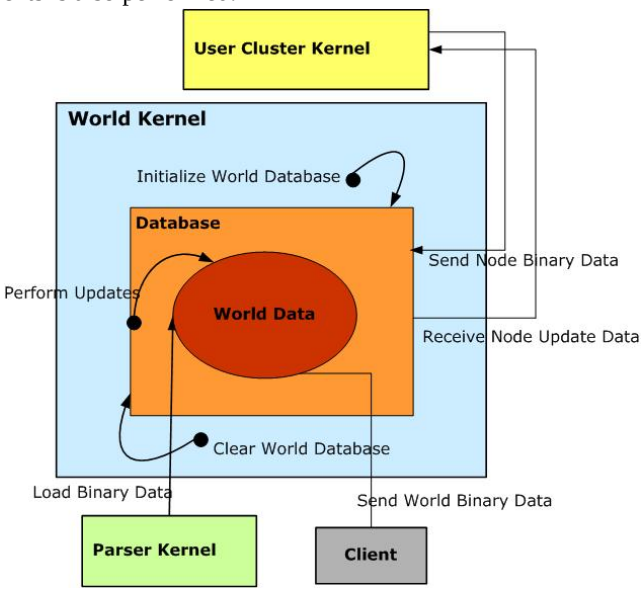

Fig. 2: World Kernel Operations

Clients, in this case, are the User Cluster Kernels, described in the following subsection, which serve the corresponding part of the world and are responsible for forwarding the events received by the World Kernel to the user clients connected to it. Also, spatial filtering can be applied to the outputted events, to significantly reduce the traffic generated by the World Kernel, so that each update is forwarded only to the User Cluster Kernels that lie in its area of effect. This way, better scalability is achieved, since the load on the World Kernel does not depend on the total number of users but only on the number of affected User Cluster Kernels.

2) User Cluster Kernel

For the manipulation of the users' avatar movements within each virtual world "User Cluster Kernels" are introduced. Each User Cluster manages a group of users that participate in the virtual environment. Thus a virtual world may consist of many User Cluster Kernels.

In particular, this kernel is responsible for the entire I/O process with the users connected to it. It handles user movement (without updating the database of the World Kernel) and all other events by forwarding these events to the World Kernel, if necessary, and receiving updates from the World Kernel that are forwarded to the users.

User Cluster Kernels are arranged on a regular grid over the virtual world and each of them undertakes an area which it handles. At this point it should be mentioned that a User Cluster Kernel can also serve users that are outside its area of responsibility, but within some limited distance.

Due to the fact that the distribution aims to support large scale virtual environments, the platform selects a more effective way for handling the events that update the virtual environment. In particular, when an event modifies the state of an object within the area of interest of a specific User Cluster Kernel, this message is not forwarded to all other existing User Cluster Kernels, as it would happen in the client-server version of the platform. In particular, in the client-server architecture an event message would have to be sent to all participating users.

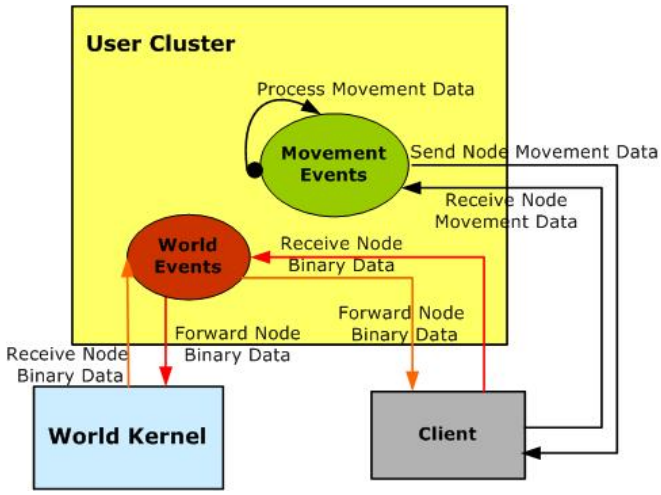

Fig. 3: User Cluster Kernel Operations

However, the distributed approach performs a two-layered filtering: The World Kernel will send the message only to the User Cluster Kernels it involves and the User Cluster Kernels will forward this message to the clients they handle. For example, in the case of a virtual environment with 1000 concurrent users, the World Kernel would have to send 1000 messages for these users. With the filtering adopted the World Kernel will only send two messages (if we assume that there are two User Cluster Kernels). If the User Cluster Kernels serve about 50 users, they will only forward this message to these 50 users. Thus the number of information exchanged is tremendously decreased.

3) Parser Kernel

Each virtual world and all of the objects available in it are saved under a VRML/X3D file format. These files are stored on a server as well as on the platform's database through a reference link. The distributed version of EVE generates a dedicated thread for manipulating these files. In particular, the concept of Parser Kernel is introduced, which constitutes a remote kernel dedicated to parsing VRML/X3D data, since this can be a very time-consuming task.

The Parser Kernel parses the file containing the virtual environment along with the objects contained and sends the parsed data in binary form to the corresponding World Kernels, which will store them. So, the heavy task of parsing and preprocessing the VRML data is now moved away from the potentially heavily loaded World Kernel that handles the affected portion of the world. In fact, the remote kernel manager will chose the less loaded machine in the supporting network to send the parser process.

C. Kernel Managers

This section presents the Kernel Managers that the distributed version of EVE adopts for managing the processes of the platform. These processes are not created once and stay static but instead they are created and intercepted dynamically during the operation of the platform. The platform adopts two kinds of kernel managers: (a) the Local Kernel Manager, which accepts and executes the commands from the Remote Kernel manager for the creation of new processes on the machine it handles and the stopping of other processes and (b) the Remote Kernel Manager, a nodal point of the architectural model, which is responsible for the distribution of all processes and makes the decisions during the operation. These Kernel Managers are described in the following subsections.

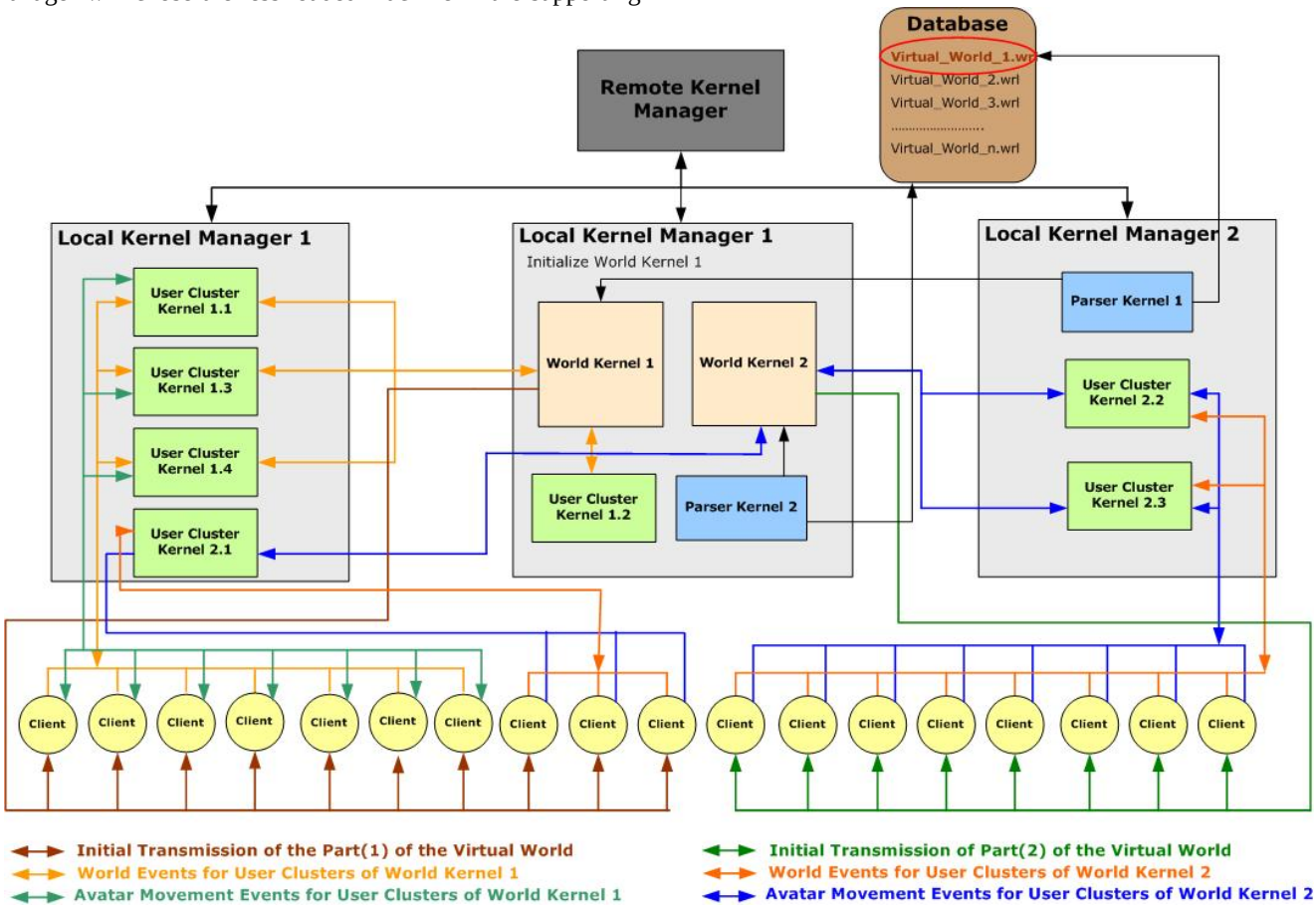

Fig. 4: Distributed Scheme Entities and Interrelations

1) Local Kernel Manager

The entity of the Local Kernel Manager is used for spawning kernels locally on a single machine, controlling their execution and also keeping track of local resources and computational load for the given machine. In particular, a Local Kernel Manager runs in each of the machines that participate in a session and performs the necessary authentication that will allow to the World, Parser and User Cluster Kernels (or: any kernels) to be able to connect to the platform and perform their designated tasks on the world(s) they serve. When the authentication of the Local Kernel Manager is completed, this entity waits for instructions from the Remote Kernel Manager. Sole responsibility of the Local Kernel Manager, after the connection to the platform, is the starting and stopping of processes on the host machine, as designated by the commands received by the Remote Kernel Manager, while also providing the Remote Kernel Manager

with feedback regarding the computational load and available resources on this machine. This way, each local kernel manager acts as a client to the Remote Kernel Manager, which is responsible for the general coordination of kernels.

2) Remote Kernel Manager

The Remote Kernel Manager stands as the administration entity of the platform as it is responsible of controlling all processes that constitute the platform. In particular, the Remote Kernel Manager maintains the information necessary (ip address, port, type and state information) for all connected Local Kernel Managers (which represent participating servers) and all kernels currently executing. The Remote Kernel Manager can remotely spawn new processes on remote machines (by commanding the appropriate local kernel manager to do so), terminate processes remotely and reply to queries made by kernels, which try to find other kernels connected to the platform based on certain criteria such as type and world id. Furthermore, the Remote Kernel Manager acts as the coordinating entity for the whole platform. In particular, during a session, where users join the platform, new processes need to be created for balancing the workload. Similarly, when two processes can be merged (i.e. two User Cluster Kernels for which the number of clients is importantly reduced). The above decisions fall under the jurisdiction of the Remote Kernel Manager. This thread could either run on separate machine or in one of the machines that a Local Kernel Manager runs.

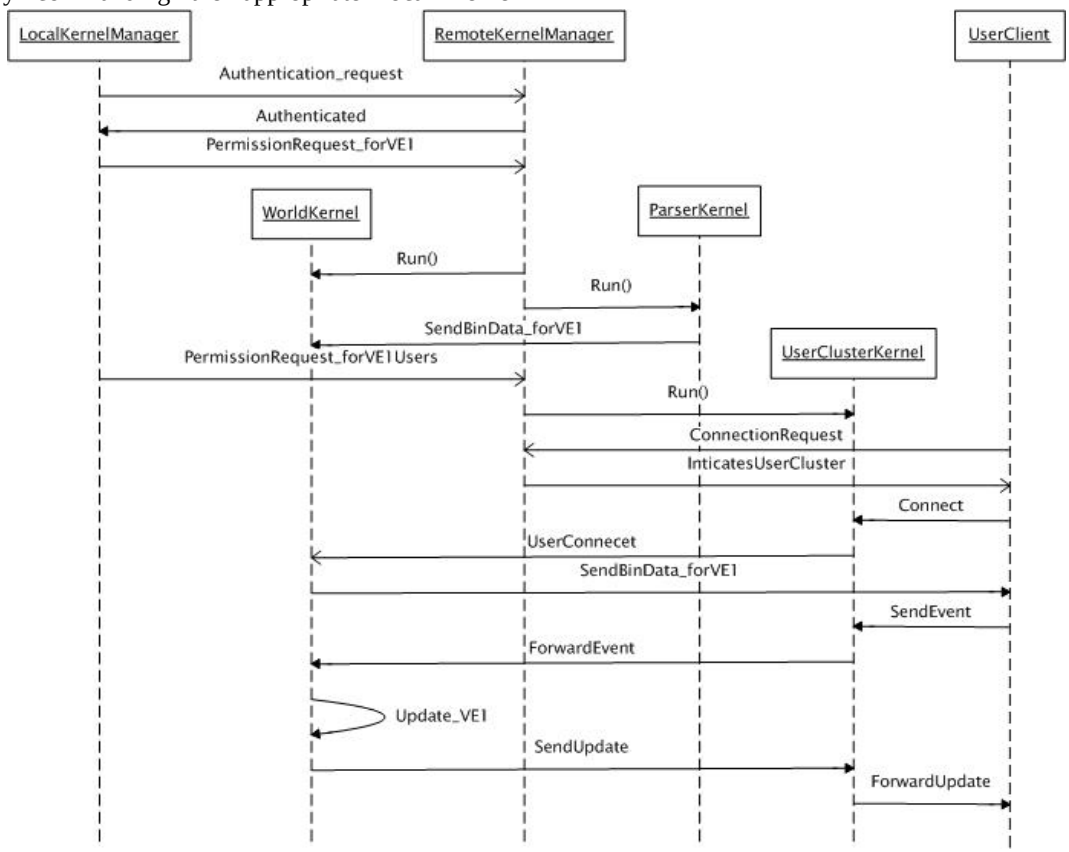

Fig. 5: UML sequence diagram showing world initialization, user connection and updates

D. Deployment Example

We assume that we have two different machines, on which the platform will operate. Each of these machines runs a Local Kernel Manager, while in one of these machines, or even on a separate machine a Remote Kernel Manager maintains the information necessary for all connected Local Kernel Managers.

When a Virtual World is to be loaded, the machine requests from the Remote Kernel Manager to start a World Kernel for this virtual world. The Remote Kernel Manager selects one of the available machines and sends a command to the corresponding Local Kernel Manager to spawn the corresponding process, sending at the same moment a series of information that are related to it, such as the name of the world. When this process is completed, we ask from the Remote Kernel Manager to run a Parser Kernel, which is supplied with the url where the world data reside (.wrl file) and the id of the world that shall be built by these data.

The Parser Kernel parses the file and through the Remote Kernel Manager finds the corresponding World Kernel and sends it the binary data that were parsed. The World Kernel stores this data in the database.

With the same process we create a User Cluster Kernel by making a request to the Remote Kernel Manager. When the

User Cluster is running, users can enter the virtual world. Each client asks the Remote Kernel Manager to which User Cluster it should connect. The data during the loading of the virtual world are sent to the client directly from the World Kernel.

V. Partitioning Algorithm

As Distributed Virtual Environments tend to become a de-facto solution for large-scale application a number of problems need to be taken into account for achieving an effective performance. One of these problems is the partitioning of the virtual space. To this direction both heuristic and on-heuristic approaches have been proposed [14]. At this point it should be mentioned that partitioning, as presented in [15] constitutes a NP-complete problem. Lui and Chan [15] have described the importance of finding a good assignment of the participating clients to the available servers for managing the workload and the communication cost and for achieving a better networking performance. This partitioning algorithm currently achieves the best results for DVEs [16].

As the partitioning algorithm that EVE will adopt will be based on the approach of Lui and Chan, we will describe briefly the solution they proposed and we will then present the modifications that will be performed for improving the performance.

The partitioning algorithm, as presented in [15] has three basic steps:

(a) A Recursive Bisection Partitioning algorithm (RBP), which, based on the concept of divide and conquer, creates the initial partitions of the virtual world.

(b) A Layering Partitioning algorithm, which, based on the computing workload reassigns the clients to the servers so as to reduce the overall cost.

(c) A Communication Refinement Partitioning algorithm (CRP), which reassigns some clients to other partitions (servers) so as to reduce the server-to-server communication cost.

They also propose a quality function (denoted as Cp), for evaluating each assignment of clients to servers. This quality function takes into account two parameters, a) the computing workload generated by clients in the DVE system, which should be shared among the available servers in regard to their computing resources and b) the overall inter-server communication requirements.

The approach of the partitioning algorithm presented in this section takes into account some additional parameters, which can further improve the performance of the algorithm proposed in [15]. These parameters are the following:

- Nob =The number of active multi-user objects, which are the objects that the user can interact with. These active objects could either be (a) *static*, in the sense that cannot be transferred from one location to another and (b) *moveable*, which are the active objects that can be moved from one location of the virtual world to another.
- Nst = The number of inactive objects, which are the objects that the user cannot interact with but are placed within the virtual environment.
- $Os_{i,}$ =The size of each multi-user object, where i =1,.., Nob.
- $Ons_{i,}$ =The size of each inactive object, where i =1,.., Nst.
- VS =The total size of the virtual space.
- As =The average size of the participating avatars.
- $L(a_i, o_j)$ =The amount of information generated by the interaction of an avatar with an active object of the virtual world.
- SP_i =The available (free) space of a given partition P_i

Our approach is based on the observation that there is an upper limit on the number of clients that each virtual world can support, which is not related only to the available resources of the system but also to the actual space of this world. In particular, let us assume a server that handles one small virtual world with dimensions 2×2 units and also comprises some objects. Let us also assume that we have a powerful system with enough computer resources to server up to 30 concurrent users. If the average size of the avatars that participate in this virtual world is As then the actual number of concurrent users that this space can support equals

to $NA\max = (VS - (\sum_{i=0}^{Nst}(c * Ons_i) + \sum_{i=0}^{Nob}(c * Os_i))) \div As$,

where c is a constant, which is related to the collision bound value set for the virtual environment.

Based on the above, we extend the partitioning algorithm of Lui and Chan [15] for improving the general performance.

A. Recursive Bisection Partitioning (RBP) Algorithm

This algorithm creates the initial partitions of the virtual world and for each of the created partitions calculates their size

1. Divide the virtual world into N disjoint cells of area D^2, where D is the average diameter of the Area of Interest (AOI) of the avatars

/* Create a graph for the virtual environment*/

2. For each cell i create a node and calculate:
   - the number of avatars that reside in this cell Na_i
   - the number of active Nob_i and inactive Nst_i objects that reside in this cell
   - the size of the active Os_i and inactive Ons_i objects that reside in this cell
   - Calculate the free space of the cell SP_i
3. For neighboring cells create an edge
4. Assign all cells into one server
5. Based on the graph created reassign cells from one server to another so as to minimize the total communication cost among the servers

6. Select the partition with the largest cost and apply the algorithm again until all servers are assigned with a partition

B. Layering Partitioning (LP) Algorithm

The main objective of this algorithm is to find a new partition based on the results of the Recursive Bisection Partitioning algorithm. The basic steps of this algorithm are the following:

1. For all created partitions P_i, where i=1,2,…,n *check* the available free space /* the free space is known from the RBP algorithm */ {
2. If (P_i *has available* free space) then { /* we start with the creation of a graph /*
3. For each avatar a_i and object o_i in partition P_i *create* a node va_i (representing the avatar) and a node vo_i (representing the object)
4. { For each node va_i *find* all *avatars* (va_j) and *objects* (vo_j) within its AOI which reside in different partitions $P_j, (i \neq j)$ /*these avatars are denoted as border avatars */
5. while ($Num\_va_j > 0$ or $Num\_vo_j > 0$) /*in case there are avatars or objects within the AOI of a_i /*
6. { *create* an edge ea_{ij} between va_i and va_j
7. *assign* a weight to edge ea_{ij}, which equals to the communication cost $I(a_i, a_j)$ /* $I(a_i, a_j)$ represents the amount of information exchanged among avatar a_i and a_j
8. *create* an edge eo_{ij} between va_i and vo_j
9. *assign* a weight to eo_{ij} which equals to the interaction cost $L(a_i, o_j)$
10. *calculate* the overall cost for va_i: $OC_{va_i} = \sum (I(a_i, a_j) + L(a_i, o_j))$ }
11. Find partition number for which: $Ocost = \sum(I(a_i,a_j) + L(a_i,o_j)) = \max\{\sum(I(a_i,a_j) + L(a_i,o_j))\}$ and assign a label with this partition number to va_i }
12. For all other nodes in P_i create edges between these nodes and the newly labeled border nodes /*these "other" nodes are the ones for which both $Num\_va_j = 0$ and $Num\_vo_j = 0$ */
13. calculate the overall cost
14. label them with the partition number which maximizes $O\cos t$ /*in this case the avatar and object nodes reside in the same partition*/ }
15. *calculate* the total number mt_{ij} of nodes that can be moved from P_i to P_j /*we refer to these nodes as candidate nodes*/ }
16. else { /*this is the case that there is no available free space in partition P_i, which means that the number of nodes that can be moved from other partitions to this node $mt_{ji} = 0$, for $(i \neq j)$ }
17. We define t_{ij} as the final number of the candidate nodes that can be moved from partition P_i to P_j.
18. Based on the available space in partitions, estimate the maximum number of avatars that can be reassigned.
19. For all the neighboring partitions try to minimize $\sum_{1 \le i \neq j \le n} t_{ij}$, where n is the number of partitions (servers), trying to approach an ideal workload balanced situation of server i.

C. Communication Refinement Partitioning (CRP) Algorithm

The Communication Refinement Partitioning algorithm is applied for reducing the communication cost among the partitions produced by the Layering partitioning algorithm. The concept of this algorithm is very similar to that of the layering algorithm. In particular, the graph produced by the layering algorithm is used and the following steps are performed:

1. For all border nodes va_i calculate the overall communication cost /*these are the nodes for which there is an edge connecting va_i to either an avatar va_j or object node vo_j in a different partition $(i \neq j)$ */
2. Compare the overall communication cost and label node va_i with the server number for which the maximum cost is achieved.
3. Calculate the total number z_{ij} of nodes that can be reassigned from partition P_i
4. We define b_{ij} as the final number of the candidate nodes that can be moved from partition P_i to P_j.
5. For all the neighboring partitions try to maximize $\sum_{1 \le i \neq j \le n} b_{ij}$, where n is the number of partitions (servers), keeping in mind that the number of nodes that are assigned to server i equals the number of nodes withdrawn from this server, so as to maintain the workload balancing.

VI. Further Considerations

As mentioned in [3], DVEs need to address more serious problems than the ones of single user virtual reality systems.

These problems are related, among others, to the control of network traffic, latency, and reliability. In addition, DVEs mainly aim at supporting Large-Scale applications. The term “large” refers both to the size of the virtual environment, including the size of the graphics, as well as to the number of concurrent participants. These characteristics introduce a series of situations and problems that need to be solved for achieving sustainability and for optimizing the performance. The main considerations that need to be taken into account when designing and developing a DVE are: a) the scalability and b) the heterogeneity. In particular, regarding scalability, there is a certain limit on the number of users that each system can support. When the number of users exceeds this limit malfunctions might occur, which are related with the limited computing resources. Thus methods and mechanisms are needed for avoiding undesirable effects in the interactivity and rendering [3]. The other main consideration is the heterogeneity of equipment of the participants. For facing this problem, there are two solutions which either deploys the simulation based on the slowest and weakest system or setting up a minimum requirement and simply denying access to nonconforming systems [3],[4].

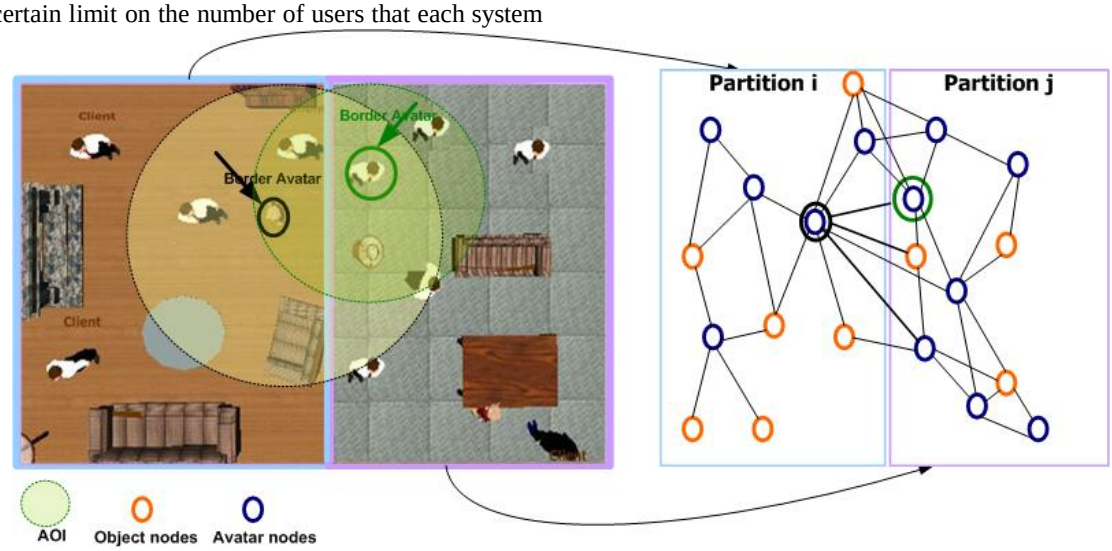

Fig. 6: 3D Space representation in a graph

A. Scalability

As mentioned above, scalability is one of the main considerations for a DVE. As pointed out in [6], the key design issue for achieving scalability is to reduce the number of exchanged messages, which might burden the system, without, however, affecting the consistency of the virtual environment. This section describes the issues that could be addressed as well as the efforts and mechanisms that have been developed for each of them.

Interest management: This issue is related to the notation that users that participate in a virtual environment could only be notified of changes and events in their area of interest, for reducing the number of exchanged messages and for relieving network resources. The interest management could be divided, based on [7], into two methods according to the fidelity of capability of message filtering: a) division of the virtual space into regions and b) localization of the area of interest of the participants. In the distributed version of EVE, the interest management will be handled with a combination of the above two solutions.

Communication Architecture: This issue is related to the model adopted for the exchange of the messages among the participating nodes. The models available are client/server, which however fails for large-scale applications, the peer-to-peer model, where each peer assumes all the responsibility of message filtering and synchronization and the peer/server model. In this case the communication among the participants is achieved using multicast through peer connections, while the server maintains the consistency [6]. In the distributed version of EVE the communication architecture exploits the primitives of the client-server architecture, with the main difference that there is no server, in a physical sense, but instead there are dedicated kernels, thread responsible for the exchanging of the messages.

Data replication: This issue is related to the initial loading and updates of the virtual environment. The most common approach to this is the replication of the data to the client at the beginning of a session and the continuous update of the changes that take place during the session. However, an important problem with this approach is the replication of large-scale environments, especially at the beginning of the session. In these cases the most common solution is to replicate only the objects of interest for the user [6]. As mentioned in [8], for the data replication there are two techniques mainly used: a) the prioritized transfer of objects, and b) a caching and prefetching techniques. In the distributed version of EVE, the data replication is handled as follows: at the initial transmission of the virtual environment, only the area, including the objects, of interest for the user is transmitted. Also, a streaming algorithm, with a level of detail

view is applied that allows the gradual transmission of data. Thus, the user obtains a wider overview of the environment, without large waiting times.

Concurrency control: This issue is related to the need for the maintenance of the consistency of the users' view within a virtual environment. In particular, in multi-user environments, where the number of users is large, which also increases the probability of the interactions and consequently of the events that need to be updated, it becomes clear that concurrency is a major issue for the achievement and maintenance of realism. As mentioned in [6], as communication delay increases, the probability of conflicts between operations does as much. For this reason the concurrency control is of vital importance for the synchronization and the effectiveness of the application. The concurrency control schemes have been broadly categorized into pessimistic, optimistic and prediction scheme [9]. The distributed version of EVE adopts the pessimistic scheme for ensuring that the update messages will be delivered in the order the actions were realized for maintaining the consistency among the connected users' views.

B. Heterogeneity

Another limitation that should be taken into account when implementing a Distributed Virtual Environment is the heterogeneity. To this direction, there are basically two approaches commonly used by existing architectures. On the one hand, for facing heterogeneity problems, the systems either need to meet a minimum performance or all systems have to reduce data transmitted so that the weakest of the systems could support the load. Both solutions are somewhat inadequate because the first prevents some users from joining the CVE session, and the second would under-use resource [4], [10].

VII. Conclusion-Future Work

Large-scale virtual environments are large both on the size of the virtual space as well as on the number of the concurrent users that aim to support. Traditional architectures, either client-server or peer-to-peer, seem unable to support such application. A solution to this comes from Distributed Virtual Environments.

This paper discusses the migration of a networked virtual environment, called EVE, which was based on client-multi server architecture and aimed to support small-scale applications to a distributed platform, which will be able to support large-scale networked virtual environments. The reasons that led to this migration and the problems that need to be faced are presented. Since the platform we adopted was based on client-server architecture, the first step to be taken was to modify the model to a distributed one. The entities of the distributed model, the operations they perform and tasks they undertake as well as the communication among them are described. In addition the papers describes briefly a partitioning algorithm, which based on [15], takes additional space-based parameters as input for improving the overall performance of the partitioning.

Some preliminary results of the algorithm show that the introduction of the objects in the algorithm plays an important role on the initial assignment of the virtual world to the available servers. Our future steps include extensive experiments for monitoring the performance of the partitioning algorithm in regard to the size of the virtual space and the distribution both of the avatars and objects.

References

[1] C. Bouras, E. Giannaka, A. Panagopoulos, T. Tsiatsos, "A Platform for Virtual Collaboration Spaces and Educational Communities: The case of EVE", Multimedia Systems Journal, Special Issue on Multimedia System Technologies for Educational Tools, Springer Verlang, 2006, (to appear)

[2] C. Bouras, E. Giannaka, "Performance Monitoring on Networked Virtual Environments", 5th International Conference on Internet Computing(IC 04) Las Vegas, Nevada, USA, June 21 - 24 2004, pp. 302 - 308

[3] N. Pryce, "Group Management and Quality of Service Adaptation in Distributed Virtual Environments", 4th UK VR-SIG Conference, Brunel University, Uxbridge, UK, November 1997

[4] J.C. Oliveira, N.D. Georganas, "VELVET: An Adaptive Hybrid Architecture for VEry Large Virtual EnvironmenTs", Journal of PRESENCE: Teleoperators and Virtual Environments (MIT Press), Vol. 12, Issue 6, pp. 555-580, December 2003

[5] Calvin, J., Dickens, A., Gaines, R., Metzger, P., Miller, D. and Owen, D., "The SIMNET Virtual World Architecture", IEEE VRAIS, September 18-22 1993, Seattle, Washington

[6] D. Lee, M. Lim, and S. Han, "ATLAS – a scalable network framework for distributed virtual environments", ACM International Conference on Collaborative Virtual Environments (CVE'02), pages 47–54, September-October 2002

[7] Lim, M. and Lee, D., "Improving Scalability Using Sub-Regions in Distributed Virtual Environments", International Conference on Artificial Reality and Telexistence, Tokyo, Japan, December 1999, 179-184.

[8] Park, S., Lee, D., Lim, M. and Yu, C., "Scalable Data Management Using User-Based Caching and Prefetching in Distributed Virtual Environments" ACM Symposium on Virtual Reality Software and Technology, Canada, November 2001, 221-226

[9] Yang, J. and Lee, D., "Scalable Prediction Based Concurrency Control for Distributed Virtual Environments", IEEE Virtual Reality 2000, New Brunswick NJ USA, March 2000, 151-158

[10] Singhal S., Zyda M., "Networked Virtual Environments: Design and Implementation", ISBN 0-201-32557, ACM Press, 1999

[11] Greenhalgh, C., Purbrick, J., & Snowdon, D., "Inside MASSIVE-3: Flexible support for data consistency and world structuring. ACM Collaborative Virtual Environments", 2000 (CVE 2000), 119–127

[12] Hook, D. J. V., Rak, S. J., & Calvin, J. O., "Approaches to relevance filtering", Proceedings of the 11th DIS Workshop on Standards for the Interoperability of Distributed Simulation, 367–369

[13] Don McGregor, Andrzej Kapolka, Michael Zyda and Don Brutzman "Requirements for Large-Scale Networked Virtual Environments," Proceedings of the 7th International Conference on Telecommunications ConTel 2003, Zagreb, Croatia, 11-13 June 2003, pp. 353-358

[14] P. Morillo, J.M. Orduña, M. Fernández, J. Duato, "A Comparison Study of Metaheuristic Techniques for Providing QoS to Avatars in DVE Systems", Lecture Notes in Computer Science, Volume 3044, Jan 2004, pp. 661 – 670

[15] John C.S. Lui, M.F. Chan, "An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems", IEEE Transactions on Parallel and Distributed Systems, Volume 13, No. 1, Jan 2002

[16] P. Morillo, J.M. Orduna, J. Duato, "On the Characterization of Distributed Virtual Environment Systems", Proceedings of European Conference on Parallel Processing (Euro-Par'2003), Klagenfurt, Austria. August, 2003