A SOFTWARE ARCHITECTURE OF DISTRIBUTED VIRTUAL REALITY SYSTEM FOR FORMATION FLIGHT VISUALIZATION

Vasily Y. Kharitonov

Department of Computers, Systems and Networks
Moscow power engineering institute (technical university), Russian Federation
E-mail: KharitonovVY@gmail.com

Abstract

Among problems facing today's computer science one of the most challenging is designing of a distributed virtual reality system (DVR system) which are used in many applications of human activity including the aerospace industry. Such systems should ensure consistent interaction of many geographically separated users in a shared virtual environment. This requires complex approach involving creation of dedicated software architecture. In paper we examine basic DVR system design principles and propose such an architecture. Application of proposed architecture for formation flight visualization is discussed.

1. INTRODUCTION

At present, with the growth of computational power of modern computer systems the aerospace industry has witnessed a great increase in the number of various training simulators. These simulators allow to solve a wide variety of challenges facing today's astronautics and aviation, both civilian and military purposes, while providing significantly lower material costs and safety requirements. One of the foundations of modern training simulators is the technology of virtual reality (VR) that enables user to "immerse" in a highly-realistic artificial world or environment, providing interaction of user and virtual environment.

Meanwhile, the group trainings of many crews are becoming more popular, that is of great importance in solving a series of practical tasks, such as formation flight, refueling in the air etc. Solving of such tasks requires the creation of a qualitatively different system, namely *a distributed virtual reality system* (DVR system), allowing to reproduce temporal-spatial interaction of many objects managed by remote users. In such a system users should experience a sense of presence in the shared indivisible *virtual environment* (VE) despite how far they are distanced on a computer network scale.

The realism of virtual world, experienced in DVR systems, depends not only on the quality of graphics, but also on the underlying distributed middleware and its networking mechanisms. This middleware must provide consistent representation and visualization of the virtual environment for all users (pilots) allowing to observe as mush identical VE states as possible (though, possibly, from different view points), regardless hardware limitations imposed by the communication lines and nodes hardware. Maintaining of consistent VE representation is particularly significant in the problem of formation flight simulation, where it is very important to ensure the accurate and consistent spatial positioning of aircrafts relative to each other for all pilots.

The essence of this work is to develop software architecture of DVR system providing a consistent VE representation for many users. To do this we should answer to several questions. First, what architectural principles should be built into DVR system to ensure a consistent interaction of large numbers of users? Secondly, what limitations affect the consistent displaying of virtual reality moving objects and how they can be overcome? And third, since from the program point of view DVR system consists of multiple interacting processes, how to effectively organize data exchange to provide users with experience of being in a shared virtual world.

In paper we first look at main DVR system features comparing DVR systems with classical distributed systems. Then we examine basic DVR system architectural principles considering them at three abstraction layers and formulate main requirements determining the qualitative nature of users interaction.

Finally, we propose a software architecture of DVR system and discuss its application to formation flight visualization.

2. DVR SYSTEM ARCHITECTURAL PRINCIPLES

First, let's consider main virtual reality terminology. The term Virtual Reality is used to describe a computer-generated, highly-realistic artificial world or environment (called a Virtual Environment), allowing the user to interact with it in real-time by interfacing some of his actions in the real world back into the virtual environment and providing visual, acoustical and, sometimes, haptic feedback. The soft hardware allowing geographically remote users to interact in the shared virtual environment is referred to as the Distributed Virtual Reality system:

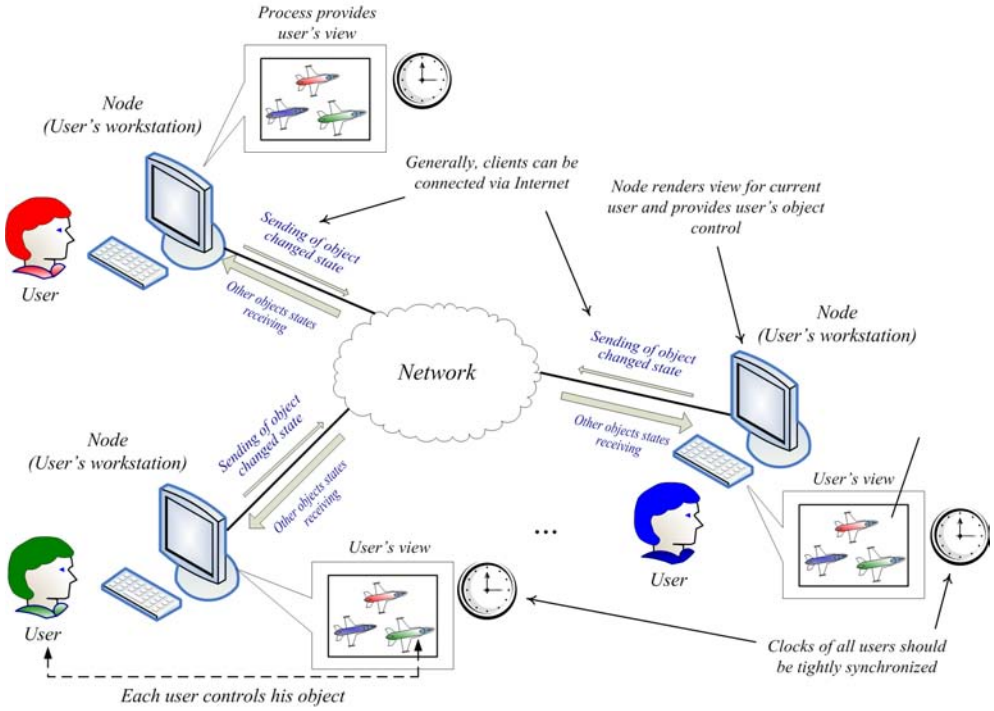

Figure 1. Generalized representation of DVR system

Virtual environment is a collection of virtual objects with certain sets of attributes. Attributes determine the properties and behavior of each object, and together form an *object state*. The *VE state* is a tuple of all states of its objects. The *view* is a rendered image of VE observed from the given position in the virtual space.

2.1 DVR systems features

DVR systems are a subclass of distributed systems and, therefore, inherit all of their properties [1]:

- *No global physical clock.* All system processes interact asynchronously (but the system can still maintain some clock synchronization model).
- *No shared memory.* All communications are carried out only through the message exchange between processes (though there may be a shared memory abstraction).
- *Geographical separation*. Processing nodes of DVR system can be located within the local area network (LAN), as well as on a wide-area network (WAN).
- *Autonomy and heterogeneity*. Processing nodes are loosely coupled in the sense that they may have different performance and running various operating systems. They are usually not part of a dedicated system, but interact with each other to solve common problems.

However, DVR systems have their own features:

- *All computations are performed in real time.* This imposes special requirements on computational nodes which should be able to handle large amounts of information in short time, as well as on communication channels which should ensure instant access to the most actual VE state;

- *Specific nature of distributed computations in DVR systems.* Most of the computations in DVR systems are concentrated on the VE global state calculation which is constantly changing, not only in response to users' actions, but also with the progress of time.
- *Requirement of clock synchronization.* Although there is no common system clock, each process has its own local clock and in DVR system it is very important that these local clocks are well synchronized with each other.

2.2. Representation of DVR systems at different abstraction layers

We consider the design of DVR system at three abstraction layers: *user*, *software* and *hardware*. This allows a particular developer to divide system-building process into separate stages.

At the highest, user layer of abstraction DVR system should be indivisible and transparent for the user, hiding all of its distributed nature and implementation details (see fig. 2). Each user (pilot) should be given a view to the virtual environment and a *logical interface* to interact with it.

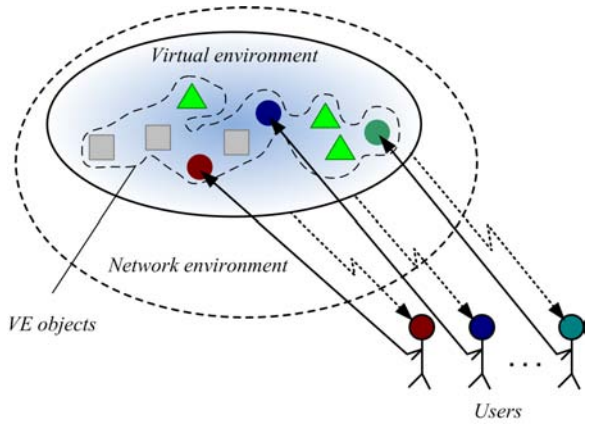

Figure 2. DVR system for user

Examining architectural principles, we primarily keep in mind the software architecture since the software layer is fundamental in DVR system design. At the software layer, DVR system is a collection of asynchronous processes, interacting with each other based on a certain type of *communication architecture* using some kind of a *high-level protocol* (see fig. 3). The main types of communication architecture are *client/server* and *peer-to-peer*. There are also combined architectures, such as multi-server architecture. Depending on the chosen architecture, *client*, *server* and *peer* processes can be distinguished. In client/server architecture client processes are focused on the individual user's view visualization and state control of his object, while server process provides interaction of multiple users. In peer-to-peer architecture peer processes include both functions. The high-level protocol is based on general network protocols, such as TCP/IP protocols, and makes it possible to transmit data between processes taking into account specific communication architecture.

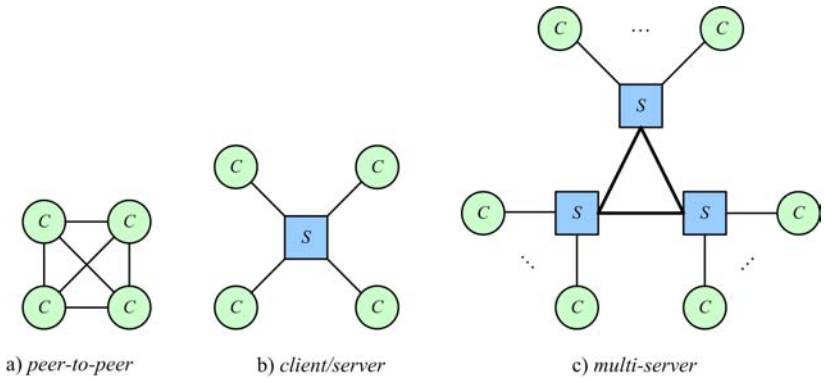

Figure 3. DVR system representation at the software layer

When considering hardware layer, DVR system consists of computing nodes (in the simplest case, PCs), connected via data network. At this layer a particular type of network is chosen, on which basis the system is to be built, and hardware requirements for the nodes are specified. Also input and display devices for user-to-system interface are determined (*man-machine interface*).

2.3. The main requirements to DVR system — consistency and responsiveness

The main requirements for DVR systems that determine the qualitative nature of users interaction are *consistency* and *responsiveness* [2]. Generally, consistency requirement means that all users of DVR system should have identical data on the VE state at every moment. At the same time information on state changes (update messages) should be distributed between users in the minimally possible time. When named conditions are satisfied, it is said about *consistent interaction* (in terms of transferring data between nodes). One of the main objectives achieved through consistent interaction is to provide a *consistent displaying* of VE objects, allowing all users to observe as mush identical VE states as possible, though, possibly, from different view points. It is impossible to achieve absolute consistency since in a real system local VE state copies necessarily are to differ, for example, because of data transmission delay (latency). To ensure high consistency, each user, having performed any action, should wait before undertaking next action until data safely reach other users. Also, at the software layer processes should be tightly coupled that requires high bandwidth and low latency, as well as imposes restrictions on the number of users.

The responsiveness of the system is the time required for the user action to make the result in the virtual world and become noticeable for the user. To ensure high responsiveness each user's process should not wait for other remote users to be notified about user's actions. Instead, it should change the local VE state copy so that the user immediately would become aware of his actions result. Therefore, processes should be loosely coupled, making a large amount of local calculations. The decision on how action will affect the virtual environment should make a user's process itself.

However, the user may not always determine the result of his actions alone. For instance, to perform collision detection of multiple users' objects in a right way (for example, missile and air target), it is necessary to take a collective decision, which is contrary to the requirement of high responsiveness, because such a decision requires some time on the data exchange between users. If each user attempts to detect collision independently of the others, all users can come to different results, and the consistency of the system may be disrupted. Thus, the requirement of responsiveness can be in opposition to the requirement of consistency. In most cases it is impossible to achieve both these requirements at the same time and trade-off have to be found.

3. PROPOSED DVR SYSTEM SOFTWARE ARCHITECTURE

3.1. Overview

The proposed architecture is presented in fig. 4 that depicts the main, in our view, components which real DVR system should consist of, as well as the relationships between them.

According to the picture, DVR system is considered as many separate processes (shown as large bars) interacting with each other on the basis of a certain type of communication architecture (in this paper we are limited to client/server architecture, but multi-server architecture also possible). All processes consist of a collection of computing units and a data storage. Each client process is focused on the individual user's view visualization and state control of his aircraft (which can be an airplane, a helicopter etc.), while server process provides interaction of multiple users. Now consider main DVR system components.

User input handling unit reads data from the user input devices and converts them to control actions that are transmitted to *Local VE state simulation* unit. Client process keeps (in *Local VE state storage*) and simulates (in *Local VE state simulation* unit) only part of VE state needed to visualize VE within user's visibility scope. VE state simulation involves modeling of VE objects states in accordance with their physical models. *Global VE state simulation* unit tracks changes in the *Global VE state storage* and ensures collective objects interaction.

When needed, the data not presented in local storage are loaded from the *Global VE state storage*, located on the server. This need can be related either with swap of new VE parts, or with state change of remote users' objects. In turn, modification of any VE object state performed by current user must also be reflected in the global storage.

Thus, the *Global VE state storage* accumulates all the changes in the VE and contains complete information about VE state at any instant of time. There is a bidirectional data exchange between global storage on the server and local storages on the clients all the time. Together server and client storages form distributed data storage.

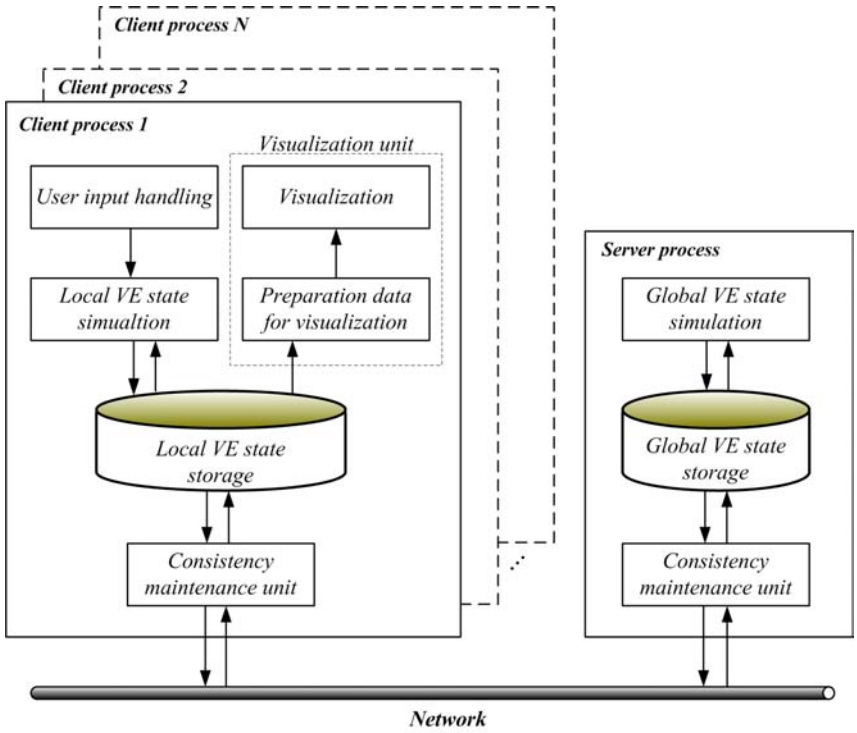

Figure 4. Proposed software architecture of Distributed Virtual Reality System

Data from the local storage on the client process is input to the *Visualization* unit which renders user view. The *Preparation data for visualization* unit optimizes data from the local storage for rendering and transforms them into the appropriate format. Then optimized data come to *Visualization* unit which can be either an individual computer or a complex rendering system.

The most important unit of the DVR system is a *Consistency maintenance unit*. It is a part of each process and resolves the following issues:

- communication of current process with other processes (a high-level protocol);
- applying methods for hardware limitations compensation;
- data replication management;
- clock synchronization;
- object ownership management (defining ways to access and modify VE state).

Below we look at some of them.

3.2. Overcoming hardware limitations

One of the major obstacles taking place when organizing a consistent interaction are hardware limitations imposed by the communication lines and nodes hardware. The main limitations are *network bandwidth*, *latency* and *node processing power*. As it is not possible to overcome these limitations completely [2–4], it is reasonable to use dedicated software-based approaches that enable more efficient use of computing and network resources. Among them are the following:

1. *protocol optimization* — assumes minimizing the number of messages transmitted over the network, using data compression, as well as grouping several messages in one

2. *relevance filtering (interest management)* — allows to adjust the flow of data between users in accordance with certain criteria. For example, it is often used visibility-based filtering in which data are transferred between users only if they are in visibility of each other.
3. *dead reckoning algorithms* [2,4,5] are applied for predicting objects states at any given moment of time that makes it possible to transfer data less frequently as well as reduces reliability requirements to data channels and network protocols;
4. *dynamic VE state distribution among several servers*.

All of these approaches minimize network traffic transmitted between nodes. Moreover, the approach 3 reduces the impact of latency, and the approach 4 allows to decrease the amount of calculations performed by individual nodes.

3.3. Data replication strategies and distributed scene graph

There are three main data replication strategies: *centralized*, *replicated* and *distributed* (see fig. 5). When using a centralized architecture, all VE state data are stored on a dedicated process. In a replicated architecture each process keeps a copy of the entire VE state. In a distributed architecture VE state is distributed among several processes.

In our software architecture none of these strategies is used in its original form. Instead, we use hybrid strategy. On the one hand, there is a centralized data storage, located on a dedicated server or group of servers, where the most relevant VE state is stored. On the other hand, data from the central storage are replicated on the client processes to reduce the number of requests to a central storage and to increase data access speed (or, in other words, to increase responsiveness of DVR system). In addition, in case of large and extensive virtual world it makes sense to replicate on the clients not a complete VE state, but its separate parts. The other parts of virtual world can be swapped when necessary, with the movement of a particular user in the VE. This approach allows to significantly improve the scalability of DVR system. However, for its effective implementation a special form of VE state representation is required. For our purposes it is convenient to organize information in data storage in a hierarchical structure which is shared between users, called *distributed scene graph* (*DSG*) [6].

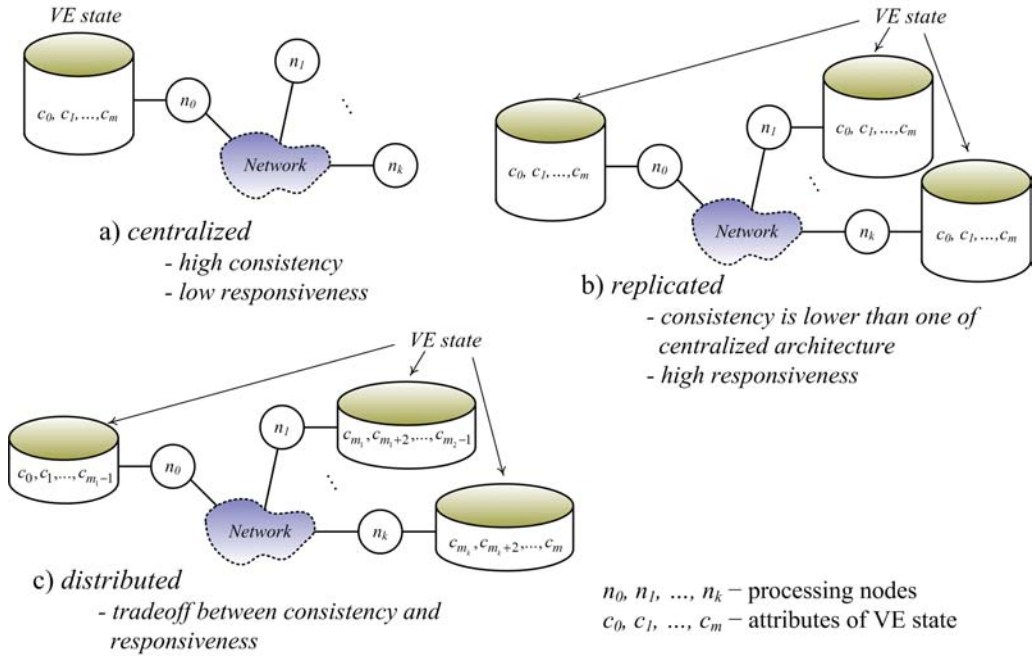

Figure 5. Main data replication strategies

DSG establishes logical and spatial relationships between VE objects. It is headed by a top-level root node which encompasses the whole virtual world. The world is then broken down into a hierarchy of nodes representing either spatial groups of objects, settings of the position of objects, animations of objects, or definitions of logical relationships between objects. The leaves of the graph represent the VE objects

themselves, their renderable geometry and material properties. Formally speaking, a scene graph can be defined as a directed acyclic graph with a random number of children. Main benefits of DSG include:

- fits well with the object structure of the VE and is a natural representation of the virtual world for the human perception;
- provides means for effective 3d-rendering of VE state by implementing various graphics acceleration algorithms (such as *culling* and *level-of-detail techniques*), and collision detection algorithms;
- increases the scalability of DVR systems by distributing the computational load on VE state processing between the processes;
- increases flexibility of data replication;
- simplifies VE objects state management;
- easy programmable.

3.4. Object ownership management

When building DVR system it is very important to correctly handle requests of multiple users to the same data. This situation often occurs, for example, when multiple users attempt to simultaneously manipulate the same object. The actions of different users overlapping in time should not interfere with each other. To do this a mechanism is needed to control access to the objects states.

The proposed architecture uses an approach based on object ownership transferring. Each VE object can be either free or owned by a single process. This ownership can be transferred between processes. Any other process may try to seize the object by sending corresponding request to the server process. If the requested object is free (i.e. not occupied by another process), then the server marks it in a global storage as “owned” and sends the user a positive response. After this only owner process is able to alter object state in the global storage.

If the requested object is owned by another process, the server sends ownership request to this process. The owner process may either meet this request and grant membership or reject request, depending on his needs. Owner process may also block access to the object. In this case any membership request is automatically declined.

3.5. Clock synchronization

Clock synchronization is the technique of ensuring that physically distributed processes have a common notion of time. Clock synchronization assumes availability of a *reference clock* in the system that is stored on a dedicated process called *time server*. In our architecture, each newly connected client synchronizes its clock with the server clock using a special procedure, similar to Cristian algorithm [7] or algorithm used in Network Time Protocol [8]. Reference clock set the course of the internal *model time* which is common for the entire DVR system. All events in the VE are related to particular instants of model time. Each global VE state is calculated based on model time.

3.6. High-level communication protocol

Proposed architecture assumes applying of a specialized high-level protocol. It provides process interaction and includes a set of messages defining various operations in VE and interactions between processes. It is based upon two transport level protocols of TCP/IP stack: TCP and UDP, combining reliability features of the former and speed of the latter. Individual messages that require reliable delivery are transferred via TCP (for example, messages on creation VE objects). For often transmitted messages that require fast delivery and are not critical for packet loss UDP is used. Such combination of protocols allows to use network bandwidth in more efficient manner.

Protocol messages are divided into groups. The main of them are: world messages, object messages, service messages. World messages define global operations on VE modification such as objects creating/deleting, models loading, weather effects control, time management, camera control and others. Object messages include operations on object state altering, DSG structure modification, ownership management etc. Service messages provide additional communication operations on top of underlying transport protocols, such as address send from between any two processes or batch send allowing to group several messages into a single one.

3.7. Practical use of the architecture

Proposed architecture is employed as a basis for a software framework allowing to create DVR systems for specific application areas. This framework is a cross-platform programming library with the C++ API. It represents a middleware for DVR system providing easy and transparent interface to work with VE state as well as has built-in means for 3d-rendering based on OpenScenGraph [9].

The framework can be used for various applications requiring interaction of many users in real-time. One of them is formation flight visualization. In this case virtual environment should include such objects as terrain, buildings, ground targets and various aircraft objects. Each pilot' work place should have interface to control his aircraft, visual, acoustical components providing pilot with experience of being at the controls of real aircraft. There also should be simulating component computing aircraft physics. These components can be implemented as parts of a single aircraft simulator. Each pilot's simulator is connected to others through provided framework API by creating aircraft representation in VE. The framework ensures pilots interaction in a shared VE.

Precise aircraft physical model may be rather comprehensive and its state can include large set of attributes. However VE object model shouldn't contain all of these attributes to be communicated to other pilots. It is enough to keep only resulting parameters of physical model in the VE object state, such as aircraft position, orientation, velocity and acceleration. Using dead reckoning algorithms makes it possible to provide smooth aircraft movement on remote site.

An example of formation flight visualization system based on the developed framework is shown on fig. 6.

Figure 6. The formation flight visualization system

4. RELATED WORK

There are three main branches of DVR systems at present time:

- simulating equipment;
- networked virtual environments;
- multiplayer computer games, MMOGs and metaverses.

The first type of systems is mainly used in various military simulations: from aircraft simulators to large-scale battlefield simulations. The well known representatives are SIMNET (SIMulation NETwork) and its followers that had become industry standards — DIS (Distributed Interactive Simulation, IEEE Standard 1278) and HLA (High Level Architecture, IEEE Standard 1516) [10].

Networked virtual environments are mainly used for academic and educational purposes. Such systems are applied in many research projects that require remote interaction of many participants (e.g., distance learning). Examples of such systems: DIVE, RING, NPSNET [11, 12].

Multiplayer networked games are the most widespread type of DVR systems and use similar to other types of mechanisms. Among the most successful game series there are Unreal Tournament, Counter Strike (Half-Life Source) [13], and Quake. There is growing interest in the new genre of network games — Massively Multiplayer Online Game (MMOG), which support especially large number of users (currently, up

to hundreds of thousands). More general and global concept is used in metaverses. Metaverse is more than game; it a lifelike complex open-ended virtual world where “winning” simply means exploring, communicating, and creating objects. Second Life is one of the examples of metaverse [14].

The closest analogue of the proposed architecture within existing systems is HLA. The High Level Architecture (HLA) for Modeling and Simulation was developed to facilitate interoperability among simulations and promote reuse of simulation components. Using HLA, computer simulations can communicate to each other regardless of the computing platforms. Communication between simulations is managed by a Run-Time Infrastructure (simulation) (RTI). However, the HLA specification only defines a distributed runtime infrastructure (RTI) but not its implementation. Therefore, heterogeneous RTIs can not interoperate. In addition, various RTIs provide different performance which in most cases insufficient for simulating VE at high rates. HLA supports only peer-to-peer communication architecture which does not scale up well for large scale geographically distributed simulations and can be effectively used only on LAN network. Moreover, there are no built-in mechanisms compensating hardware limitations.

5. CONCLUSION AND FUTURE WORK

In this paper we studied a number of issues taking place in DVR systems. The main components that real DVR system can be constructed of, were considered. As a result the software architecture of DVR system was proposed. It was shown that this architecture could be used as a basis for creating formation flight visualization system.

In the near future, we plan further development and expansion of the proposed architecture and the framework by improving existing mechanisms, as well as by adding new ones, such as multi-server communication architecture and QoS for clients.

ACKNOWLEDGEMENTS

This work was supported by the State Analytical Program “The higher school scientific potential development” (project no. 2.1.2/6718).

REFERENCES

[1] A. Kshemkalyani and M. Singhal. Distributed Computing: Principles, Algorithms, and Systems, Cambridge University Press, New York, NY, USA, 2008.
[2] J. Smed, H. Hakonen. Algorithms And Networking for Computer Games. UK, Chichester: John Wiley & Sons, 2006.
[3] V. Y. Kharitonov. Methods of efficiency enhancement of network interaction in distributed systems of virtual reality // Proceedings of 2nd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2007. IEEE Computer Society, Los Alamitos, CA, USA, 2007. pp. 305-308.
[4] S. Singhal, M. Zyda, Networked virtual environments: design and implementation, ACM Press/Addison-Wesley Publishing Co., New York, NY, 1999.
[5] V. Y. Kharitonov. An Approach to Consistent Displaying of Virtual Reality Moving Objects. Proceedings of 3rd International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008, IEEE Computer Society, Los Alamitos, CA, USA, 2008, pp. 390-397.
[6] M. Naef, E. Lamboray, O. Staadt and M. Gross. The blue-c distributed scene graph. Proceedings of the Workshop on Virtual Environments 2003, EGVE '03, Vol. 39. ACM, New York, NY, 2003, pp. 125-133.
[7] F. Cristian. Probabilistic clock synchronization. Distributed Computing, Vol. 3, 1989, pp. 146-158.
[8] D. L. Mills. Internet time synchronization: the network time protocol. IEEE Transactions on Communications, 39(10), 1991, pp. 1482-1493.
[9] OpenSceneGraph — an open source high performance 3D graphics toolkit. http://www.openscenegraph.org
[10] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules, IEEE Standard 1516-2000 (2000).
[11] T. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments // Symposium on Interactive 3D Graphics, April 1995. pp. 85-92.
[12] M. Capps, D. McGregor, D. Brutzman, and M. J. Zyda. NPSNET-V: A New Begining for Dynamically Extensible Virtual Environments // IEEE Computer Graphics & Applications, vol. 20, 2000. pp. 12-15.
[13] Source Multiplayer Networking. Valve Software, 2009. http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking.
[14] Second Life Project. www.secondlife.com.