Multi-Agent Software Architecture for Distributed Virtual Reality Systems

Volodymyr Duchenchuk
Institute of Software Systems
Kyiv, Ukraine
dvbua@ukr.net

Volodymyr Boublik
National University Kyiv-Mohyla Academy
Kyiv, Ukraine
boublik@ukma.edu.ua

***Abstract* — Interactive distributed applications with a shared virtual environment, also known as distributed virtual reality (DVR) systems, have been examined in this paper. DVR systems are efficiently used in applications for education, simulation, training, and entertainment purposes. They impose a challenge for a high level of real-time interaction between users in an efficient way in terms of network usage. The aim of this paper is the development of a special multi-agent software architecture for distributed virtual reality systems. Main components of a multi-agent system as well as communication mechanisms between the agents have been suggested and evaluated. On the base of multi-agent systems, consistent visualization of objects as well as their interactions within the system has been achieved.**

Keywords—distributed, virtual reality, multi-agent, mobile

I. INTRODUCTION

The networked world in the age of information society completely depends on the quality of software designed to accompany all areas of human activity. Existing networks and moreover networks of the nearer future dictate extremely complicated challenges for software development within distributed systems based on numerous hardware and network platforms. A subject of special interest in this case is the development of distributed sophisticated virtual reality systems. They have proved their effective applicability in many areas of education, science, engineering, entertainment, and medicine. Virtual reality systems allow their users to immerse themselves in virtual worlds by enabling a high level of interaction with the virtual environment. Extensive computer networks involving rigorous numbers of user in man-computer interaction demand development of flexible and scalable distributed virtual reality systems. They enable multiple users concurrently immersed in virtual reality to simultaneously interact with each other.

The main aim of this work is to develop a multi-agent software architecture enabling the development of distributed virtual reality systems. Section II gives a brief overview of the main features of distributed virtual reality systems and the main issues arising from their development. Section III describes the main components of multi-agent systems and their applicability to distributed virtual reality systems. Developed multi-agent software architecture for distributed virtual reality systems is introduced in Section IV.

II. DISTRIBUTED VIRTUAL REALITY SYSTEMS

Virtual Reality is a computer-generated realistic simulation of a world, which is often referred to as a virtual environment, which enables the users to interact with the environment in real-time in a seemingly real or physical way by a person using special electronic equipment.

A virtual environment is a collection of virtual objects with certain sets of attributes that determine the properties and behavior of each object, forming an object state. The state of the virtual environment is a combination of the states of all virtual objects contained in the virtual environment [1].

A distributed virtual reality (DVR) system is a virtual reality system that enables remote users to interact in a shared virtual environment. Typically, users in DVR systems connect from different geographical locations using a wide-area network, nevertheless a DVR system may also be running in a local area network.

A DVR system is composed of a set of asynchronous processes that communicate by messages passing over the communication network; these processes do not share a global memory and communicate solely by sending messages to each other [2]. The communication delay between the processes is unpredictable; there is no shared clock between the processes.

The global state of a DVR system is composed of the states of the related processes and also the communication channels used. Each of the processes of a DVR system consists of the local state of the virtual environment, which together with network messages form the global state.

DVR systems are a subclass of distributed systems, so they inherit all of their properties [2]:

- No global physical clock. Asynchronous system processes introduce the distribution element to the system.
- No shared memory. The processes must communicate using message passing. An abstraction of shared memory can be introduced, but it will just be achieved using messaging behind the scenes.
- Geographical separation. The processes can run in a local area network (LAN) as well as in a wide-area network (WAN).
- Autonomy and heterogeneity. The processors are loosely coupled, which means they can be running at different speeds and have different operating systems installed.

DVR systems also have their own unique characteristics [3]:

- Real-time computations. As all computations are performed in real-time, the computational nodes

978-1-7281-6760-2/20/$31.00 ©2020 IEEE

should be able to handle large amounts of operations on the information in a short time. This also imposes requirements on communication channels which should provide access to the actual state of the virtual environment as soon as possible, in order to avoid desynchronization.

- Specific nature of distributed computations. The computation takes place mostly around the global environment state calculation.
- Clock synchronization requirement. The processes don't have a shared physical clock, but they should synchronize their local clocks with each other.

Main motivational factors for using distributed systems are resource sharing among the processes, an enhanced reliability and an increased performance/cost ratio. Reliability consists of several aspects: availability, integrity, and fault-tolerance. Usage of distributed systems also increases the scalability rate as well as modularity of the system.

There are several functions that must be addressed when developing a distributed system [2]:

- Communication. This function involves communication mechanisms between processes. Some examples are remote procedure call and remote object invocation.
- Naming. Designing efficient ways of naming, addressing and identifying is important for resource locating.
- Synchronization. A mechanism for the synchronization and coordination of the processes must be established. Also, the local clocks of the processes must be synchronized.
- Consistency and replication. Replication of data objects provides fast access to data and scalability. But this also creates a challenge for consistency among replicas.
- Security. Usage of cryptography, access control, and key management to ensure the secure state of the distributed system.

III. Agent Technologies

The term "agent" has been frequently used in software engineering; however, without a generally accepted definition. The definition provided in [4] seems to be one of the most promising ones: "agent - an entity that resides in environments where it interprets sensor data that reflect events in the environment and executes motor commands that produce effects in the environment."

An agent is a software object in an execution environment acting on behalf of a person or a human institution.

The agent possesses a combination of unique characteristics:

- Reactive. An agent is sensible to the environment and makes decisions based on changes to the environment.
- Autonomous. It can control its own actions and make decisions based on its own current knowledge of the environment.
- Proactive. The behavior of an agent is objective-oriented. The agent has its own goals to achieve.
- Continuity. The agent is continuous in time, which means it is constantly in the execution state.

Optional agent features are:

- Communicative. Ability to communicate with other agents and parties.
- Learning. Ability to adapt to the environment based on previous experiences.
- Mobility. Ability to migrate from one execution environment to another.

Different types of agents according to [4] are:

- Information agents. Agents that collect the information and efficiently process it to answer user queries.
- Autonomous agents. Agents that act autonomously on behalf of the user in the process of achieving goals stated by the user.
- Entertainment agents. Agents that provide entertainment to a user by interactive simulation of worlds.
- Interface agents. Agents that provide assistance to users in their connection with certain applications.

A multi-agent system is a software environment involving a diversity of running agents. It enables software agents to execute applications, access resources and control the lifecycle of agents. A multi-agent platform defines standards for the development of multi-agent systems. A multi-agent system may consist of multiple execution environments, which are usually called agencies of the system. Some agencies may consist of places, independent execution environments [5]. Agent systems can be grouped into domains, each containing a registry that maintains information about registered agents.

Multi-agent platforms usually realize a distributed agent environment (DAE) which consists of several components a [6]. A DAE can consist of multiple domains, multi-agent systems and places.

An agent is stationary if it stays in one execution environment, not being able to leave it. On the other hand, mobile agents are able to move to other execution environments transporting their code, data, and state of execution.

There are several features that a modern multi-agent system needs to support [6]:

- Agent execution. The agent system needs to be able to execute the code of an agent and provide access to the requested resources.
- Transport. The agent system has to support the network mobility of the agents with their data, code, and state of execution.

- Unique identification. Multi-agent systems and all of the agents inside of them must be uniquely identifiable. This imposes the requirement of a mechanism for the unique naming of agents and agent systems.
- Communication. For an effective achievement of their goals the agents need to be able to communicate with each other.
- Security. Agent systems have to provide security features such as authentication, access control and cryptography for data encryption and privacy protection.
- Management. Agent administrators should be able to control agent execution by being able to monitor the full state of an agent as well as to interrupt agent execution on demand.

The usage of mobile agents decreases network resources usage by executing the program on a remote location, instead of downloading all the necessary data to a specific location. This also leads to a decrease in network latency. Mobile agents can also provide fault-tolerance. a

Some of the well-known mobile agent systems are Aglets, Odyssey, Concordia, and Voyager. Most of the agent platforms are developed using Java programming language as it enables easy portability over heterogeneous platforms and provides efficient tools for security and serialization [7].

The usage of multi-agent technologies for the development of distributed systems provides several benefits [6]:

- Asynchronous/autonomous task execution. Agents can act autonomously and use itineraries to determine their next actions. After an agent creation, the user who initiated the agent does not need to control the agent and can continue to perform other tasks.
- Reduction of network traffic. An agent can move to the data instead of the data first moving to the process, which results in a faster speed of processing and decreased network traffic, as well as a decrease of the negative effects of network latency.
- Flexibility and extensibility. Software distribution can be on-demand by encapsulating the service software inside mobile agents, which then are downloaded by the user.
- Decentralized control and management. Agent cloning and agent migration mechanisms allow the efficient development of decentralized software.
- Increased robustness. The usage of multi-agent systems reduces the interdependence of software components which benefits the fault-tolerance of the whole system.

It is important to state some of the drawbacks of using multi-agent technology. When an agent migrates to another host, it moves with the code associated to it, which could lead to security issues. Also, agent migration may sometimes be less effective than message passing between agents. Moreover, applications that use mobile agents suffer from slower execution due to agents being written in relatively slow interpreted languages such as Java, mostly for security and portability reasons.

The security of mobile agent systems has been recently studied a lot. Hind Idrissi [8] provides a comprehensive review of existing approaches to securing mobile agent systems. He also proposes robust security mechanisms that preserve flexibility, autonomy and mobility features of mobile agent systems.

IV. MULTI-AGENT SOFTWARE ARCHITECTURE

In this paper we suggest a structure of the main components of a multi-agent environment designed for distributed virtual reality systems.

Each of the DVR system users run an agent execution environment which enables agent creation, identification and communication between the agents.

Let us first describe the way users connect to an application. For this purpose, a LobbyAgent will be used. A LobbyAgent is a stationary agent that allows the discovery of applications and user sessions. It contains information on the specific session running: application identifier, number of currently active users and maximum number of users. These agents are going to be used in a connection step to retrieve up-to-date information about running sessions.

Next, ConnectionAgent is a stationary agent that is acting on behalf of its user by communicating with LobbyAgents to retrieve information on current active sessions. Specific filters can be implemented for easier search of the desired session. The user can then select one of the sessions and initiate a connection to it. The ConnectionAgent gets access to the registry associated with a running session.

Then, we will use UserAgent, which is an agent associated with a session user. It contains information about the user and communicates with other agents that may want to retrieve information such as the user id. The UserAgent is a stationary agent that is created in the user's execution environment. By asking the session registry for the UserAgent list, it is possible to determine the current number of active users in the session.

ObjectAgent is a mobile agent responsible for objects in the virtual environment. It has an owner, who is the user that has initiated the creation of the agent. It can be an original agent or a shadow agent, which means an agent that was copied from the original. ObjectAgents can have multiple attributes provided by the developer. These attributes can have different access rights.

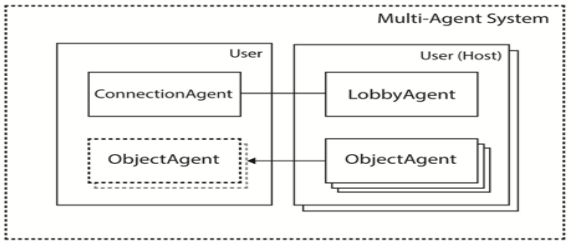

Fig. 1. Multi-agent environment during connection phase

One of the main challenges of distributed virtual reality systems is achieving the consistency of the virtual environments for the users. To achieve consistency, the ObjectAgent creates a clone of itself and then the newly

created copy migrates to the execution environment of the user. The copy is identified as a shadow of the original agent and maintains a link to it. Original agents and their copies are grouped for quick access by the agents using the session registry. Original agents are also grouped for quick access to non-shadow agents.

Agents of another user can interact with the shadow ObjectAgent by sending messages and calling procedures as governed by the access policy to achieve security of agent code execution. Once an ObjectAgent is modified, it sends an update message to its group containing the original and shadow agents, which helps maintaining the consistency among states of local virtual environments of users.

An ObjectAgent represents a specific service of an application and can contain resources to achieve its goals. For example, TerrainAgent could contain 3D models and textures for the virtual world representation as well as collision data and code that provides the ability to navigate the terrain. An ObjectAgent is migrated with the code, resources, and state of execution once moved to another user's execution environment.

If a shadow ObjectAgent crashes during the execution, the ConnectionAgent asks the original ObjectAgent on the remote host to send a new copy, which will then restore the consistency of the virtual environment. If the original ObjectAgent crashes, it is recreated using stored local data.

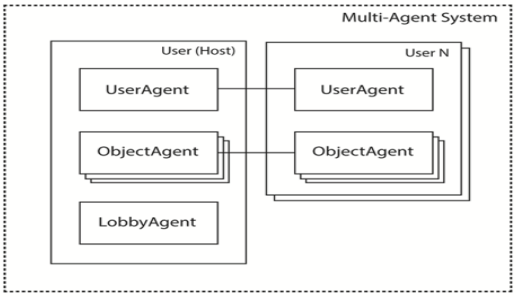

Fig. 2. Multi-agent environment during session phase

When users connect to the session, they need to get all the information about the virtual environment. Once the ConnectionAgent gets access to the session registry, it sends requests to the original ObjectAgents to create copies of themselves and migrate the copies to the user's execution environment. The ConnectionAgent waits for all the requested agents to migrate and then changes its state to connected.

Once the ObjectAgent has been migrated, the ConnectionAgent can cache some of the resources of the agent to speed up migration next time, as resources such as 3D models, textures and sounds are usually taking most of the space of an application. Together with the original ObjectAgent, the ConnectionAgent can verify the integrity of saved resources and send an optimized request for migration.

If the user who is an initiator of the session leaves or disconnects from the application session, all the agents associated with the session are destroyed and all other users are disconnected from the application. If another type of user leaves the session, all the agents that are associated with this user are destroyed.

All the agents that are non-mobile, as described in this section, can be recreated in case of a crash. Their state will be restored using local data or data from the session registry.

In order to achieve a common notion of time between the distributed processes, a clock synchronization technique is used. The reference clock is stored within the TimeAgent, which is accessible to all other agents. Newly connected clients synchronize their clock with the reference clock using the Network Time Protocol.

A modified version of a consistency model based on the selective consistency principle is used [3]. Instead of having centralized data storage as proposed in the model, each session has its own data storage managed by the initiator of the session.

V. Conclusion and Future Work

The multi-agent architecture proposed in this paper meets the requirements of a DVR system: by leveraging agent technologies it enables real-time communication between users, it enables consistent interaction in a virtual environment, and it provides basic security mechanisms using access control in a heterogeneous environment.

Some improvements could be done to the solution. For example, once the user that initiated the session leaves, the ownership of its agents could be transferred to another user. Agents would need to communicate with each other to select the user who will be selected as the next host.

The approach suggested in this paper is now being evaluated and tested in the development of the next releases of distributed virtual reality systems developed by one of the authors and widely used now in the world global network: Dragon Sim Online [9] and Cat Sim Online [10].

References

[1] V. Y. Kharitonov, "An Approach to Consistent Displaying of Virtual Reality Moving Objects," in Proc. of 3rd Int. Conf. on Dependability of Computer Systems, Szklarska Poreba, Poland, 2008, pp. 390-397.

[2] A. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. New York, NY, USA: Cambridge University Press, 2008.

[3] V. Y. Kharitonov, "A consistency model for distributed virtual reality systems," in Proc. of 4th Int. Conf. on Dependability of Computer Systems, Brunow, Poland, 2009, pp. 271–278.

[4] L. Chiariglione, "Rationale." FIPA, http://leonardo.chiariglione.org/standards/fipa/fipa_rationale.htm (accessed Dec. 5, 2019).

[5] S. Robles, "Mobile Agent Systems and Trust, a Combined View Toward Secure Sea-Of-Data Applications," Ph.D. dissertation, Dept. of Math. and Comput. Sci., The Autonomous University of Barcelona, Bellaterra, Spain, 2002.

[6] P. Bellavista, "Mobile Agent Models and Technologies for Distributed Coordinated Applications in Global Systems," Ph.D. dissertation, Dept. of Inf. Technol. and Syst. Electron., University of Bologna, Bologna, Italy, 2000.

[7] D. B. Lange, "Mobile objects and mobile agents: The future of distributed computing?" in Proc. of the Eur. Conf. on Object-Oriented Programming, Brussels, Belgium, 1998.

[8] H. Idrissi, "Contributions to the security of mobile agent systems". Ph.D. dissertation, Dept. of Comput. Sci., Mohammed V University, Rabat, Morocco, 2016

[9] Dragon Sim Online. (2019). Turbo Rocket Games. Available: https://play.google.com/store/apps/details?id=com.turborocketgames.dragonsim

[10] Cat Sim Online. (2019). Turbo Rocket Games. Available: https://play.google.com/store/apps/details?id=com.turborocketgames.catsim