Sensors in Distributed Mixed Reality Environments

Felix G. HAMZA-LUP
Computer Science, Armstrong Atlantic State University
Savannah, GA 31419

and

Charles HUGHES1, Jannick P. ROLLAND1,2
1School of Electrical Eng. and Computer Science
2School of Optics-CREOL
University of Central Florida
Orlando, FL 32816

ABSTRACT

With the advances in sensors and computer networks an increased number of Mixed Reality (MR) applications require large amounts of information from the real world. Such information is collected through sensors (e.g. position and orientation tracking sensors). These sensors collect data from the physical environment in real-time at different locations and a distributed system connecting them must assure data distribution among collaborative sites at interactive speeds.

We propose a new architecture for sensor based interactive distributed environments that falls in-between the atomistic peer-to-peer model and the traditional client-server model. Each node in the system is autonomous and fully manages its resources and connectivity. The dynamic behavior of the nodes is triggered by the human participants that manipulate the sensors attached to the nodes.

Keywords: Mixed Reality, Distributed Systems, Sensor Networks, 3D Visualization.

1. INTRODUCTION

In the majority of network collaborative Mixed Reality (MR) environments information is collected from participants through graphical user interfaces. In a limited number of cases, head tracking information is embedded in a collaborative MR session. Even in these cases the data is used only locally. Advances in sensors and computer networks have triggered an increase in the number of MR applications that entail large amounts of information from external devices (e.g. tracking systems). Sensor data is especially required to render the position and orientation of the virtual components of the scene in MR applications where a smooth blending of the real and virtual is desired.

We propose a new architecture for sensor based interactive distributed MR environments that falls in-between the atomistic peer-to-peer model and the traditional client-server model. The architecture facilitates the development of distributed MR collaborative applications in which data collected from real-time sensors is shared among the users. Each node in the system is autonomous and fully manages its resources and connectivity. The dynamic behavior of the nodes is dictated by the human participants who manipulate the sensors attached to those nodes.

The paper is structured as follows. Section 2 reviews related work. Section 3 presents the main paradigms used throughout the paper and presents a formal categorization of distributed MR application participants and system nodes. Section 4 gives an overview of the proposed architecture representing a distributed system node as a state machine. In Section 5 the behavior agent is detailed together with its load balancing capability, followed by a discussion regarding MR application Quality of Service requirements. Section 6 presents an application example built on the proposed architecture, followed by concluding remarks in Section 7.

2. RELATED WORK

One of the first and most intensive efforts in building a networked simulation was the SIMNET project started in 1983 [1] followed a few years later by the Naval Postgraduate School's NPSNet [2]. The software architecture in both cases was event driven, and a set of predictive modeling algorithms (e.g., dead reckoning) were embedded to compensate for the network delays and to allow the system to scale beyond a local area network (LAN). Important contributions of these systems included the protocol data unit (PDU) that allowed the distribution of simulation data among participants and dead-reckoning algorithms that ensured a reasonable maintenance of dynamic shared state [6]. Later the Distributed Interactive Simulation (DIS) project improved the PDU and this led to the emergence of the IEEE 1278 Standards for Distributed Interactive Simulation and its follow-on IEEE P1516 (see [3] for a discussion of these efforts). A large number of other systems have been created to distribute interactive graphical applications over a set of distributed nodes (e.g. Atlas [4], Paradise [5], DEVA3 VR [7], and MASSIVE-3 [8]).

From the perspective of the results reported here, research in distributed VR can be divided into four categories [9]. The first consists of approaches to optimize the communication protocol through packet compression and packet aggregation [10]. The second is focused on reducing the bandwidth throughout the system, and includes data visibility management, which makes use of area of interest (AOI) management [8] and multicasting [11]. Taking advantage of the human perceptual limitations, like visual and temporal perception [12], constitutes the third category. The fourth category deals with the system architecture. Most distributed virtual environment systems are built on the traditional client-server architecture and are composed of nodes with static behavior, i.e., the server node functions as a data distributor (forwarder) while the client nodes...
act as data producers and/or consumers. Once the distributed application is deployed, the nodes cannot change their core functionality and data distribution is often bottlenecked by the server capacity.

Several research efforts have concentrated on the development of sophisticated middleware on top of client-server models for distributed data sharing through remote method calls using an object oriented approach (e.g. Repo-3D [13], Avocado [14]). A drawback of building the middleware frameworks on the distributed object model is the additional delays caused by the software layers, making it difficult to maintain an interactive behavior. A few research efforts [15] have concentrated on the fundamental system architecture attempting to define a spectrum of communication architectures. While each point in the spectrum offers its own performance characteristics the participating nodes cannot change their core functionality based on the user’s interactions.

With advances in computer graphics and tracking systems the research community has shifted attention to collaborative environments that span the entire virtuality continuum [16], i.e. Mixed Reality and a subset of MR, Augmented Reality (AR) [17] [18]. AR systems were proposed in the mid ’90s as tools to assist different fields: medicine [19], complex assembly labeling [20] and construction labeling [21]. A significant leap towards entertainment based MR systems and applications, was provided by the MR Project [22] developed in Japan and presented at ISMR’99. Projects with similar goals can be found in [23] [24]. These projects brought to life practical collaborative applications for different domains based on MR paradigms. As a common feature, the applications emerging from these projects make extensive use of sensors (tracking, haptic etc) although they are based on a local collaboration paradigm and are built on the traditional client-server architecture. None of the research efforts concurrent to these projects consider the possibility of collecting and distributing real-time sensory information from multiple remote sensors in the collaborative environment.

Considering the preceding systems design’s advantages and disadvantages, we introduce a novel architecture in which nodes have a dynamic behavior dictated by the participants attached to those nodes. A prototype application following the proposed architecture has been implemented as both a proof of concept and as a way of investigating the efficiency of such a distributed system in terms of capturing and sharing the sensory data at interactive speeds.

### 3. SENSORS, NODES AND PARTICIPANTS

Collaborative MR applications usually involve the interaction of several participants. While some of the participants actively modify the shared scene, other participants are passive, in the sense that they do not interact with the shared scene. From this point of view we define two categories of participants: active participants and passive participants. An active participant triggers modifications of the virtual components of the scene from a graphical interface or through a sensor attached to his/her node. Passive participants do not trigger any modifications of the shared scene. They receive visual, haptic and/or audio feedback from the environment. The active/passive attributes of the participants can dynamically change during collaboration. An active participant can become passive and vice versa depending of the collaboration needs.

In what follows, let us define a node in the distributed system as a computing device that allows a participant to interact with the MR environment. In a distributed collaborative environment, participants need to exchange a wide variety of data (e.g. position/orientation data, virtual components attributes etc.). With the advances in sensor technology, we envision that in future systems a significant amount of data will be collected from sensors and devices attached to the participating nodes. Without loss of generality, we will consider sensors that provide position and orientation information and peripheral devices (e.g. mouse, keyboard) that allow interaction with the virtual components of the scene. The discussion can easily be extended to other types of sensors (e.g. haptic) and other devices that can be part of the distributed system’s resources.

Sensors are interaction tools from the point of view of the participant. Sensors (e.g. attached to a glove like device) provide real-time information about the position and orientation of the real objects (e.g. user's hand). This information may be used by the distributed system’s nodes to render the virtual components properly registered (in 3D space) with the real components. An interactive collaborative environment must ensure that the data captured by sensors is distributed with minimum delay to all participants. Moreover each node in the system will need to exchange its sensory data with all or a predefined subset of the system’s nodes. A pure centralized data distribution approach (e.g. client-server) would not be efficient because of the additional delay associated with the data collection stage followed by the data distribution. An atomistic peer-to-peer approach would not fit either because of the additional overhead in data distribution. Each node would have to exchange data with all the other nodes. As a fundamental property, the nodes in a sensor based MR system may act as data producers, consumers and distributors, simultaneously.

Before we introduce the details of the hybrid nodes architecture let us define four types or running modes for the distributed system’s nodes: active nodes, passive nodes, active forward nodes and passive forward nodes:

- **active nodes** - An active node represents an active participant in the MR environment. Each active node collects data from its sensors and is responsible for making the data available to interested peers as quickly as possible.
- **passive nodes** - A passive node does not inject any information into the MR environment. The node uses the information provided by active and forward nodes to render the shared scene.
- **active forward node** - As the name indicates, a forward node forwards data. The forward node can be active or passive. An active forward node injects its own as well as the forwarded data into the system.
- **passive forward node** - A passive forward node does not inject new information in the system. These nodes act as pure data forwarders.

### 4. ARCHITECTURE OVERVIEW

Let us introduce a novel architecture and discuss its suitability for the development of sensor based distributed MR applications. The design falls in-between the atomistic peer-to-peer and client-server model. Each node is autonomous and fully manages its resources and connectivity through a set of software agents: a GUI agent, a sensor agent, a rendering agent, and a behavior agent. These agents run on each node and trigger
the node behavior, i.e. a node can switch among any of the four modes described in the previous section.

The GUI agent is responsible for displaying the interface and collecting the information given by the participant through the GUI. The sensor agent is responsible for collecting the information given by the participant through the sensor(s) attached to the node (e.g. tracking sensor, haptic sensor etc).

When the participant does not interact with the shared scene, the sensor agent and the behavior agent are deactivated and the node becomes passive. The behavior agent triggers the forward state of a node. To clarify the possible states (i.e. modes) of a node and the possible transitions, in the next section we describe a state machine that approximates the hybrid node concept.

**Hybrid Nodes as State Machines**

Let's denote the states of a node as: \{A, P, AF, PF\}. "A" stands for "active", "P" for "passive", "AF" for "Active Forwarder" and "PF" for "Passive Forwarder". Let's denote the conditions that trigger the change in state using a binary representation \{00, 01, 10, and 11\}

- 00 - means that the Sensor agent goes from inactive to active
- 01 - means that the Sensor agent goes from active to inactive
- 10 - means that an activation message has been sent for forwarding on
- 11 - means that an activation message has been sent for forwarding off

Twelve transitions may occur. Table 1 summarizes the possible transitions, the triggering conditions, and the main operations that are executed at the node.

| Current State | Next State | Condition(s) | Action on the Server Thread |
|---------------|------------|--------------|------------------------------|
| A             | P          | 01           | Turn off                     |
| A             | AF         | 10           | -                            |
| A             | PF         | 01 followed by 10 | -                          |
| P             | A          | 00           | Turn on                      |
| P             | AF         | 00 followed by 10 | Turn on            |
| P             | PF         | 10           | Turn on                      |
| AF            | A          | 11           | -                            |
| AF            | P          | 01 followed by 10 | Turn off                   |
| AF            | PF         | 01           | -                            |
| PF            | A          | 00 followed by 11 | -                           |
| PF            | P          | 11           | Turn off                     |
| PF            | AF         | 00           | -                            |

The behavior of a node can be represented as a state machine as illustrated in Figure 3.

**5. THE BEHAVIOR AGENT**

Our design gravitates around the idea that every node has equal responsibilities. There is no central administration or connection arbiter. Each node renders its local scene based on the data that
it receives from its peer nodes. On the other hand each node, while in active or active forward mode, acts as a server multicasting data to the interested nodes. Therefore the design falls in between the atomistic peer-to-peer model and the traditional client-server model.

The behavior agent spawns a thread that controls the activation of the server component of the node. A passive node runs in client mode "consuming" incoming data from the participating nodes. When a passive node becomes active, it means that it acts as a data producer and distributor for the local node and for the other nodes interested in the data.

The atomistic peer-to-peer model has a fundamental bootstrap problem: how to join. Without a central server there is no easy way of determining resource availability in advance. To solve the peer-discovery situation, a new node broadcasts a query to join and awaits response. A complementary problem is: how does a node register itself to receive data from a peer node? It is the responsibility of the behavior agent to advertise that the node has available data from a particular sensor by a short broadcast to all the nodes. If a new peer is interested in the data it will be added by the behavior agent to the node's consumers list. In the next subsection we detail further the producers and consumers list mechanism.

Consumers and Producers, Load Balancing
Each node keeps two lists, producers and consumers as shown in Fig.4. The producers list contains the node's data providers. The consumers list contains the node's current consumers and their connection attributes. An active forwarder adds itself to its producers list because it inserts data in the environment.

An in-depth discussion of the QoS parameters is beyond the scope of this paper. Our goal here is merely to emphasize the architecture's flexibility with regard to the application level QoS parameters. These parameters are considered by the behavior agent for each node when it joins the specific distributed MR application. To maintain a consistent view of the shared scene over all the participants, scene synchronization approaches [29] can be employed up to a certain threshold. Beyond this threshold, application specific QoS requirements are considered.

One of the parameters that has an important effect on distributed applications is the Average Round Trip Time (ARTT) between two participating nodes. Each participating node contains a round trip time table that is frequently updated. This table contains the ARTTs between the node and each of its data consumers. Another parameter that we will consider in future versions is the computational load on each node. The computational load on each node is a factor of the number of sensors it has to manage, the number of producers and consumers it handles, and the rendering complexity of the virtual components in the scene. Each node can measure its average computational load based on the number and complexity of the threads it is executing. Moreover, this takes into consideration the hardware components of the node since the nodes in the system are heterogeneous. The rendering agent is the most sensitive because it directly influences the frame rate. A drop in the frame rate will decrease the quality of the virtual components of the scene and will have negative effects on the user task performance [30].

6. APPLICATION

To investigate the efficacy of the hybrid node concept we have developed a prototype application using the Distributed Artificial Reality Environment DARE [31] and VESS [32] frameworks.
System Setup
The application was deployed in a distributed system containing five nodes interconnected on a 100Mbps local area network. Additionally two optical tracking sensors were used to insert tracking information into the environment regarding the position and orientation of the virtual objects in the scene.

Distributed Interactive 3D Visualization
At each location, the real environment is augmented with floating 3D objects seen through the head-mounted display as illustrated in Fig.6.

7. CONCLUSION
It is clear that distributed collaborative applications involving physically distributed multiple real-time sensors are inherently very complex. It is also clear that interactive distributed MR applications and environments are becoming increasingly common. Thus, the architectural issues in building and organizing the software for such systems must be closely examined.

We have proposed a novel architecture for managing distributed sensors as part of an interactive distributed MR application. The proposed approach allows a dynamic behavior for the distributed systems nodes based on the participants' behavior allowing interactive data capturing and distribution. Furthermore we avoid a complex architecture as we believe simplicity is a key component in developing interactive applications. A subjective assessment of the Distributed Interactive 3D Visualization application interactivity confirmed our expectations. We are in the process of developing a monitoring system that will allow an objective assessment of the proposed architecture performance.

Future work involves refining the architecture and studying its scalability. If the number of remote participants increases we anticipate the need of some centralized control and additional dynamic shared state management techniques to maintain the environment interactivity.

8. ACKNOWLEDGEMENTS
We wish to thank our sponsors: The Link Foundation, NSF/ITR IIS-00-820-16, Office of Naval Research Grant N000140310677, and the US Army Simulation, Training, and Instrumentation Command (STRICOM) for their invaluable support for this research.

9. REFERENCES
[1] D. Miller and J.A. Thorpe, "SIMNET: The advent of simulator networking", Proceedings of IEEE Vol.83, No.8, 1995, pp.1114-1123.
[2] M. Zyda., D. R. Pratt, J.G. Monahan, and K.P.Wilson, "NPSNET: Constructing a 3D Virtual World", Proceedings of ACM Symposium on Interactive 3D Graphics, 1992, pp.147-156.
[3] S. Singhal and M. Zyda, "The origin of Networked Virtual Environments", in Networked Virtual Environments: Design and Implementation, Addison Wesley Pub., 1999, pp.19-53.
[4] M. Fairen, and A. Vinacua, "ATLAS, A Platform for Distributed Graphics Applications", Proceedings of sixth Eurographics Workshop on Programming Paradigms in Graphics, 1997, pp.91-102.
[5] H.W. Holbrook, S.K. Singhal and D.R. Cheriton, "Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation", in Proceedings of ACM SIGCOMM, 1995, pp. 328-341.
[6] S. Singhal and D.R. Cheriton, "Exploiting position history for efficient remote rendering in networked virtual reality", PRESENCE: Teleoperators and Virtual Environments, Vol. 4, No.2, 1995, pp.169-193.
[7] S. Pettifer, J. Cook, J. Marsh, and A. West, "DEVA3: Architecture for a Large-Scale Distributed Virtual Reality System", in Proceedings of ACM Virtual Reality Software and Technology (VRST), 2000.
