Book Title
Book Editors
IOS Press, 2003

In Proceedings of Medicine Meets Virtual Reality XIV
(MMVR'06), Long Beach, CA, January 24 - 27, 2006.

GiPSiNet: An Open Source/Open Architecture Network Middleware for Surgical Simulations

Vincenzo Liberatore [1], M. Cenk Çavuşoğlu and Qingbo Cai
Department of Electrical Engineering and Computer Science
Case Western Reserve University

Abstract. In this paper, we present the design and techniques of GiPSiNet, an open source/open architecture network middleware being developed for surgical simulations. GiPSiNet extends *GiPSi* (General Interactive Physical Simulation Interface), our framework for developing organ level surgical simulations, to network environments. This network extension is non-trivial, since the network settings pose several serious problems for distributed surgical virtual environments such as bandwidth limit, delays, and packet losses. Our goal is to enhance the quality (fidelity and realism) of networked simulations in the absence of network *QoS* (Quality of Service) through the GiPSiNet middleware.

Keywords. Surgical simulation, Virtual reality, Network, Middleware, QoS

1. Introduction

Virtual environments receive increasing interest as a new medium for surgical training. The network extension can substantially amplify the accessibility of surgical virtual environments and enable remote continuing education and advanced training. However, this network extension is non-trivial, since the network settings and realities pose several serious problems for distributed surgical virtual environments such as bandwidth limit, delays, and packet losses due to congestion. These network issues can greatly impair the user-perceived simulation quality (fidelity and realism). Thus, the performance of a networked surgical simulation critically depends on the techniques applied to implement an effective remote interaction.

In our previous work [1,2], we proposed *GiPSi* (General Interactive Physical Simulation Interface), an open source/open architecture software development framework for organ level surgical simulations, to accommodate heterogeneous models of computation and interface heterogeneous physical processes. In our ongoing project, we extend GiPSi to network environments by adding a network middleware module *GiPSiNet*, which acts as an intermediary between simulation applications and the network and takes actions

[1]Correspondence to: Vincenzo Liberatore or M. Cenk Çavuşoğlu, Department of Electrical Engineering and Computer Science, Case Western Reserve University, 10900 Euclid Avenue, Cleveland, OH 44106, U.S.A. Tel.: +1 216 368 4088; E-mail: vxl11@case.edu or cavusoglu@case.edu.

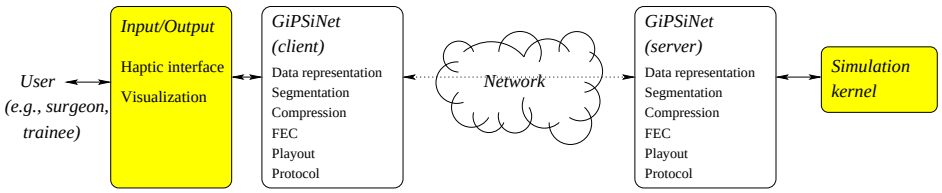

Figure 1. The GiPSiNet middleware architecture. Shaded module are part of the existing GiPSi platform, clear modules are part of the GiPSiNet middleware.

to compensate for the lack of network *QoS* (Quality of Service) [3]. Our goal is to enhance the quality of networked simulations in the absence of network QoS through the GiPSiNet middleware.

Specifically, the design objectives of the GiPSiNet middleware are as follows:

- Providing abstraction to address heterogeneous haptic devices and data representations.
- Ensuring modularity through encapsulation and data hiding.
- Supporting customizability to accommodate various surgical simulations.
- Utilizing network bandwidth and other system resources efficiently. GiPSiNet should be lightweight and efficient for data communication.
- Devising compensation strategies for networks with delay, jitter, and congestion.

In this paper, we present the design and techniques of GiPSiNet, an open source/open architecture network middleware being developed for surgical simulations.

2. GiPSiNet Architecture

In our current implementation of GiPSi, an API is provided between the simulation kernel and I/O (i.e., haptic I/O, visualization, graphical user interface (GUI)). To extend the kernel-I/O API to network environments, we add a middleware layer (GiPSiNet) between the kernel and the I/O module to encapsulate all networking functionality (Figure 1). The resulting simulator has two communication end-points: (1) the *client*, which interacts with the end-user (e.g., surgeon, trainee) through the haptic and visualization interfaces, and (2) the *server*, where the simulation kernel runs and the physical processes are numerically simulated. The GiPSiNet high level client-server architecture and data flow are illustrated in Figure 2.

Accordingly, relevant modules in GiPSi (i.e., haptics I/O and visualization) are restructured so that their functionalities are appropriately distributed over the network. For example, the architecture of the GiPSi haptics I/O module is shown in Figure 3. In this architecture, the Haptic Interface (HI) class provides an abstraction and a uniform API for the physical haptic interface device, and hardware specific low-level function calls. The Haptic Interface Object (HIO) class is the representation of an actual haptic interface in the simulation. Then, the Haptics Server is responsible of creating the instances of the HI class, initializing them, attaching them to the proper HIOs, and when requested by the Simulation Kernel, enabling, disabling, and terminating them. The Haptics Client starts and initializes instances of HIs, starts the real-time schedulers for the physical haptic interface devices, establishes network communication with the Haptics Server, and,

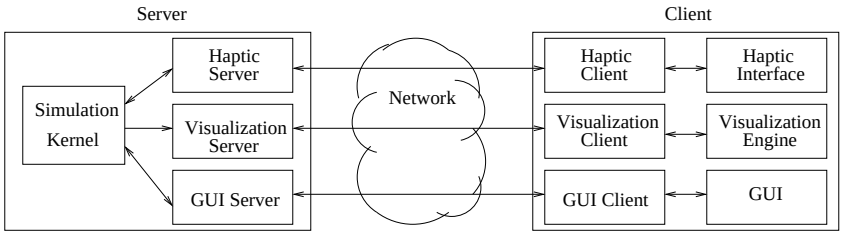

Figure 2. The GiPSiNet client-server architecture and data flow.

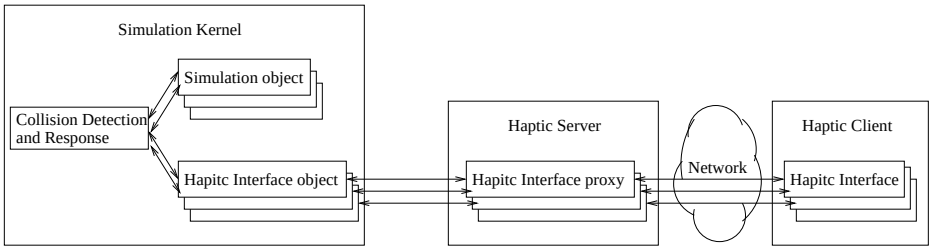

Figure 3. The high level architecture and data flow for GiPSiNet haptics.

when requested by the Haptics Server, enables, disables, and terminates HIs. Each HI at the haptic client side has a corresponding HI Proxy residing at the haptic server which encapsulates the network communications for haptics I/O and consequently is the main API for extending to GiPSiNet.

2.1. Communication Model

GiPSiNet uses the following agreement for the client-server communication. The client sends the server a data unit including: (1) an input x_i, the state of user actions, e.g., a scalpel position, and (2) an integration interval Δt_i. The server computes a linearized approximation V_{i+1} of the system state after the integration interval Δt_i assuming that the input x_i is applied continuously during such interval, and replies to the client with a data unit including the new system state V_{i+1}. GiPSi views such data units as building blocks for its operations. However, for each "unit", several operations are required to make it suitable for network transmission, for example, segmentation and compression as implemented in the GiPSiNet middleware.

Figure 4 shows a canonical simulation client-server communication: the client sends $(x_i, \Delta t_i)$ to the server, and the server replies to the client with the new system state V_{i+1}. The information exchange follows a classical on-demand protocol: the client sends a request $(x_i, \Delta t_i)$ to the server, and the server replies with a response (V_{i+1}). Ideally, the client sends requests at regular intervals of length Δt_i (say, 100ms, as in the GiPSi simulator), and the server reply is delivered to the client before Δt_i. However, in a best-effort network such as the Internet, a server reply may be delayed or lost due to congestion, and cannot be delivered before Δt_i. Therefore, we should take actions to remediate for such network impairments and ensure timely data delivery as discussed in Section 3.

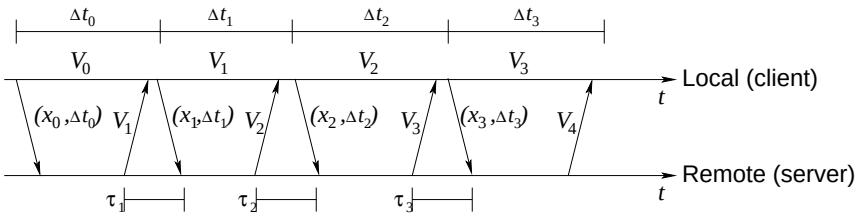

Figure 4. Timeline of data unit exchanges between client and server.

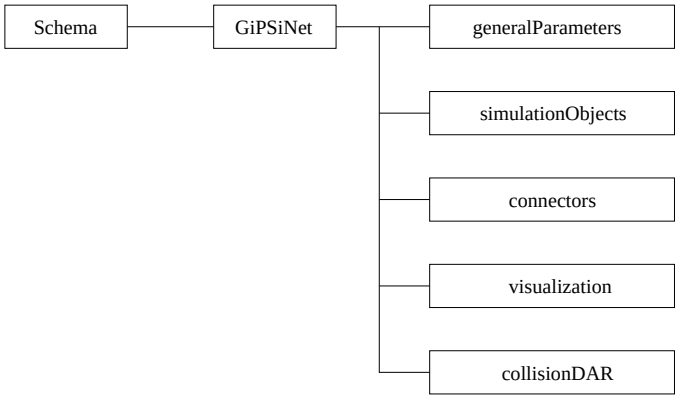

Figure 5. An overview of the XML schema representing the data exchange between a simulation server and a user interface client.

2.2. GiPSiNet Modules

Data representation module. This module implements a format to represent the data exchange implied by the GiPSi kernel-I/O API. For example, we adopt XML, a platform independent and well supported technique, to exploit the data exchange between a simulation server and a user interface client. Figure 5 shows an overview of the XML schema used in GiPSiNet to set up or change the parameters of the simulation environment.

Protocol module. A simulation server should sense clients' input through haptic devices, apply corresponding operations, and respond to clients with haptic feedback and the updated simulation object geometry in real time. Therefore, in the protocol module, we implement a lightweight protocol for efficient communication between the client and server. This protocol is based on User Datagram Protocol (UDP), which incurs much less overhead than Transmission Control Protocol (TCP) and is consequently better suited for real-time traffic.

Segmentation module. The sizes of client and server data unit (i.e., $(x_i, \Delta t_i)$ and V_{i+1}) are currently of the order of 14 and 11,664 bytes respectively. However, the size of packets in the Internet traffic cannot exceed certain limits, typically 1500 bytes (the Ethernet Maximum Transmission Unit). Therefore, a data unit is divided into multiple packets (*segmentation*) if it is longer than the limit. Moreover, in a best-effort network such as the Internet, there is no guarantee of data delivery, i.e., packets can be lost, duplicated, delayed, or delivered out of order. Meanwhile, the light-weight protocol in GiPSiNet is

UDP-based and UDP provides no mechanisms to ensure reliable data delivery. Therefore, segmentation handling is necessary.

Compression module. Data compression can reduce bandwidth requirements in communication network and the loss probability. Moreover, compressed data has a shorter transmission time given the same end-to-end available bandwidth. However, compression also imposes additional processing at the data source and destination. There is intensive research on devising efficient compression algorithm for various types of data. In GiPSiNet, we implement a data compression algorithm based on the standard Burrows-Wheeler algorithm [4], a simple and light-weight algorithm leading to better compression rates than other widespread techniques. Moreover, this algorithm can compress each data unit independently of others and consequently make the compression resilient to data losses.

Section 3 introduces the rest of the GiPSiNet modules incorporating with the techniques to ensure timely data delivery.

3. Timely Data Delivery

Most of today's networks are unreliable and are based on the best-effort model, which provides low level of QoS. Data delay and losses can greatly impair the realism and fidelity of a networked simulation. An approach to address these issues is to apply techniques to ensure timely data delivery in the GiPSiNet middleware.

Retransmission. There are three categories of data losses: the client data unit loss, the server data unit loss, and both data unit loss. If a server data unit is lost, the client will fail to receive the updated system state and the simulation will appear irresponsive to the user. On the other hand, if both data units are lost, both the server and the client will wait for data from each other and the simulation will be deadlocked. A solution is to let the client resend its data unit to the server at a regular interval length δ regardless of the client data unit loss. This regular retransmission by the client can support timely data delivery to some extent.

Forward Error Correction (FEC) Module. FEC is an error control method which adds redundancy to the data transmitted using some algorithm. The receiver only needs a certain portion of the data that contains no errors. FEC does not eliminate packet losses, but it can significantly reduce the data loss probability. Hence, a FEC module is included in the GiPSiNet middleware.

Playout Module. The round-trip time (RTT) changes over time due to varying network conditions. This variance of network delay (jitter) disturbs the temporal relationships in the visualization and hapic media stream. An approach to address this problem is to buffer the received packets and delay their playout, and the buffering time inteval should be long enough such that most packets can be received before their scheduled playout times. However, this buffering time cannot be too long, since it would effect the realism of a networked simulation. Hence, the appropriate way is to set the buffering time dynamically according to the network conditions (RTT) and the server computation time.

In GiPSiNet, a playout module is included to implement the adaptive playout. We will apply Algorithm 4 from [5], which appears to be effective, simple, and robust for

audio applications over the Internet. Accordingly, this dynamically changing playout time is included as Δt_i in the client's request to the server, and is the basis to compute a linearized approximation of the system state V_{i+1}.

4. Conclusion and future work

In this paper, we presented the architectural design of the GiPSiNet middleware, which extends our GiPSi framework to network environments. We also described the techniques applied in GiPSiNet to ensure timely data delivery and enhance the quality of networked surgical simulations. In the future, we will implement GiPSiNet, run experiments to obtain a better perception of the interaction between network and the virtual environments, and evaluate the technical and educational impacts of GiPSiNet.

Acknowledgements

The authors acknowledge Technology Opportunities Program (TOP) of Department of Commerce, NASA NNC05CB20C, NSF CCR-039910, NSF IIS-0222743, NSF EIA-0329811, NSF CNS-0423253, and the Virtual Worlds Laboratory at Case Western Reserve University.

References

[1] T. Goktekin, M.C. Cavusoglu: GiPSi: A Draft Open Source/Open Architecture Software Development Framework for Surgical Simulation, *the International Symposium on Medical Simulation* 2004, 240–248.

[2] M. C. Cavusoglu, T. G. Goktekin, F. Tendick: GiPSi: A Framework for Open Source/Open Architecture Software Development for Organ Level Surgical Simulation, *IEEE Transactions on Information Technology in Biomedicine*, 2005 (In Press).

[3] J. Smed, T. Kaukoranta, H. Hakonen: Aspects of networking in multiplayer computer games, *Proceedings of the International Conference on Applications and Development of Computer Games in the 21st Century* 2001, 74–81.

[4] M. Burrows,D. J. Wheeler: A block-sorting lossless data compression algorithm, Digital Systems Research Center Research Report 124, 1994.

[5] R. Ramjee, J. F. Kurose, D. F. Towsley, H. Schulzrinne: Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks, *INFOCOM* 1994, 680–688.