ICAT 2004

Dynamic Shared State Management For Distributed Interactive Virtual Environment

Youndong Park*, Jinwook Kim*, Heedong Ko*, Yoonchul Choy**

Korea Institute of Science & Technology*
{*youndong, jinook, ko*}*@imrc.kist.re.kr*

Yonsei University**
ycchoy@rainbow.yonsei.ac.kr

Abstract

Building an interactive virtual environment may involve diverse modules for performing various tasks like handling a group of interaction devices, multi-modal/ multi-cluster rendering displays as well multi-purpose simulations. This requires a fine-grained sharing of inter-module states. Previously, this fine-grained control can only be achieved through hardware subsystems within a single computer. With advancement of network technology, it is becoming possible to achieve fine-grained control over a cluster of computers that are closely connected, hence, an interactive virtual environment with multiple modules distributed across a local-area network.

Distributed Interactive Virtual Environment (DIVE) may involve distributed modules as well as distributed virtual environments that are shared across a group of user participants. In either case, the goal of DIVE is user's perception of interacting with a single virtual environment system; this is made possible through sharing dynamically changing state information across distributed objects and events. Depending on update rate and synchronization requirements of the dynamically changing shared states, the system requires from fine-grain to coarse-grain control over the shared states. In this paper, we introduce effective approaches to dynamically shared state management (DSSM) across the spectrum of control requirements by DIVE.

Key words: *Dynamic Shared States, Responsive Cyber Space, Distributed Interactive Virtual Environment*

1. Introduction

In order to make the cyberspace responsive to its surrounding environment, the virtual environment must take input from various sensors distributed in the physical environment where the user resides and provide a course of action in response to sensed input interactively. In order to handle a wide variety of input sources and to generate more realistic responses, a distributed interactive virtual environment (DIVE) framework was developed to handle network attached interaction devices as well as various sensors in real time. DIVE framework can support multi-channel display environment for immersive virtual environment. A node is defined as a collection of networked computers and interaction devices as a unit of interaction space that is controlled by a single user. The node may consist of distributed modules that are hosted by difference computers as a cluster.

The same framework was extended to handle networked virtual environments where participants may come from different nodes in the wide-area network. Each node should keep the virtual environment consistent with each other so that each participant may create an illusion of sharing the common virtual environment together in time and space [2].

DIVE framework involves a fine-grained control of dynamic shared states among multiple modules in a single node as well as a coarse-grained control of dynamic shared states among participating nodes. In this paper, we propose an architecture and software platform for sharing dynamic states in DIVE for a responsive cyberspace (RCS).

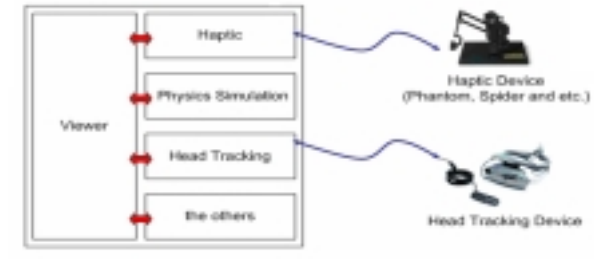
2. Requirements in Distributed Virtual Environment

Since we build a DIVE system based on network, the key consideration of our approach is how to make sure that the participant hosts or the participant of distributed interactive systems keep consistent with the view of the dynamic shared state such as a position and a velocity of object in virtual environment.

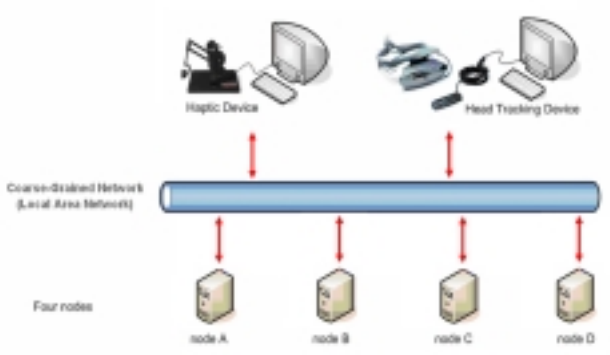
In this paper, we classify dynamic shared state management as follows.

1. Between components in the same node: Fig 1 (a)
2. Between hosts in coarse-grain networked virtual environment (net-VE) such as a clustered virtual environment (clustered-VE) system which consist of multiple hosts on LAN: Fig 1 (b)
3. Between net-VE systems in fine-grain network such as World Wide Web: Fig 1 (c)

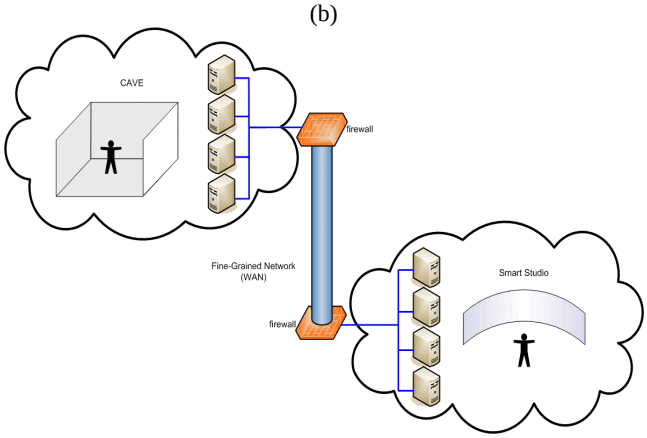
DIVE system should share user interaction state to responsive cyber space and also render the 3D virtual space to multiple display environments. We will briefly discuss the other issues such as multiple displays, scenario scripting, and so on, for DIVE.



(a)



(b)



(c)

Figure 1 Sharing dynamic shared state; (a) Sharing between components in a stand-alone VE system (b) Sharing between nodes of coarse-grained networked-VE system such as CAVE-Like system [3] using PC-cluster (c) Sharing between coarse-grained networked-VE systems on fine-grained network such as WAN.

3. Related Works

OpenGL Performer is probable the most performance-oriented high-level 3D graphics rendering toolkit for the SGI IRIX, Linux, and MS Windows platforms. It is commercially available from SGI. It has been designed to take all advantage of the system resources to achieve the best possible frame rate and real-time performance. OpenGL Performer is at its best in visual simulations and virtual reality applications. But it does not include the facilities to DIVE and for user interactions. And it provides a minority device such keyboard, mouse, joystick and trackball.

VR Juggler[4] is one of the first attempts to create a comprehensive software platform for the development and the usage of VR applications. It is a very promising the open source research and development project founded and lead by the VR Juggler group at Iowa State University. It provides both a development with C++ API and a reconfigurable run-time environment as like NAVERLib[1] library at KIST.

Avango is a VR development environment created at FhG-IMK (Fraunhofer Institute for Media Communication). Avango greatly extends OpenGL Performer's scene graph objects to allow for multi-sensory VR application development. It has a scripting language (Scheme) such as Scenario Scripting Component in NAVERLib, which allows for rapid prototyping of applications. The main goal of that is to integrate the wide variety of VR devices user at FhG-IMK and to be highly extensible.

4. Dynamic Shared State Management for Distributed Interactive Virtual Environment

4.1 Introduce to NAVERLib

NAVERLib is a microkernel architecture framework in DIVE and an object-oriented modular system. It provides component modules for a variety of interactions, interfaces, and virtual contents that can be composed in the VR environment. The entire component modules can be written by using XML based on NAVERLIB scripting language.

4.2 Sharing Dynamic State among Components such as Devices, Physics Simulation, and Visualization

A VR environment system consists of multiple components, such as: device management components, display components for visualization, sound components, and etc. Hence, it needs to share dynamic state between each component.

EventManager, which is implemented in NAVERLib, is a component for dispatching dynamic state values between components. Using EventManager, a user can describe XML scripts to dispatch dynamic state values

or execute handler functions. The user can specify the bounding conditions and simple transformation rules.

XML script for EventManager describes routers which consist of source, destination or handler. A router is a path for dispatching values in every rendering frame. The router should have a source of the dynamic state value. A router may have a destination of the dynamic state value dispatching path or a handler function which will be called if the source value satisfies its condition.

4.3 Sharing Dynamic Shared States in Coarse-Grained Networked Virtual Environment

KAVE (Kist cAVE-like system) is common system which is based on coarse-grained networked virtual environment as shown in Figure 2.

In each node of KAVE system, some components work as a server in a common server-client model. A server component updates and propagates dynamic shared state. If the component, which works as a server, is in an active mode, it change the values of dynamic shared state and will propagate them to the client components which are located on the other nodes as shown in Figure 3 (Blue line).

If the component is a passive server, it will receive dynamic state from the other system in networks, which may use different structure, and propagate them to the clients as shown in Figure 3 (Red lines).

The other component is a client. It receives dynamic shared state from the server and reflects the data using EventManager to the other components in the same node (4.2).

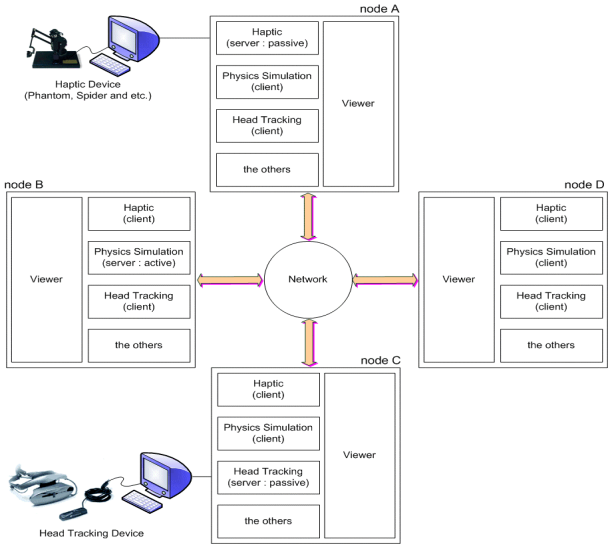


Figure 2 Example of Constitution of Components in KAVE

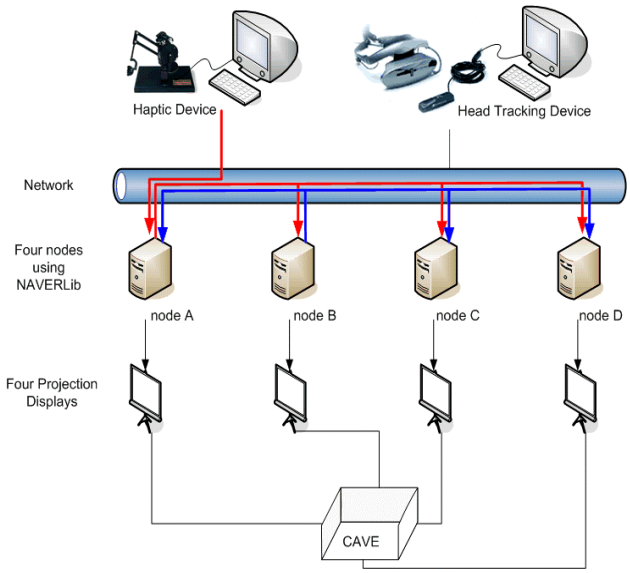


Fig. 3 Example of KAVE using NAVERLib; Red-Line: dataflow of haptic device state (passive mode), Blue-Line: dataflow of physics simulation state (active mode)

Every component in DIVE may have its own update rate. For example, the update rate for input and output data of the peripheral device is 60Hz, it means this device will receive and send the data 60 times in a second, physics simulation component may have different update rate to make the sense more real, for example 1KHz. In order to support various update rates, dynamic shared state is implemented using multi-threading and multi-processing. Clustered-VE software should provide various facilities such as a capability of choosing synchronous or asynchronous responses and minimizing locking overhead. It also contains shared resources that are manipulated by multiple threads concurrently in general.

To overcome those problems, the components in this system are implemented using design patterns such as wrapped facade pattern (It encapsulates the functions and data which are provided by existing non-object-oriented APIs, for example thread functions, within more concise, robust, maintainable, and cohesive object-oriented class interfaces.), component configuration pattern (It allows an application to link and unlink its component implementations at run-time without modifying, recompiling, or statically re-linking the application), and so on [5].

4.4 Sharing Dynamic Shared States in Fine-Grained Networked Virtual Environment

Typically, DIVE systems should be considered a multiple user interactions between fine-grained networked virtual environment systems. For example, a user may interact in a virtual world using a force/feedback device, which vibrates when collision occurs. The other user, who is located in the other place, can share context of the virtual environment. That context must be illusion on the other virtual

environment system. We design software architecture for sharing between heterogeneous systems. The server component which works in an active mode receives an input data from force-feedback device and propagates them to the client component or the other server component which works as passive mode in the other systems.

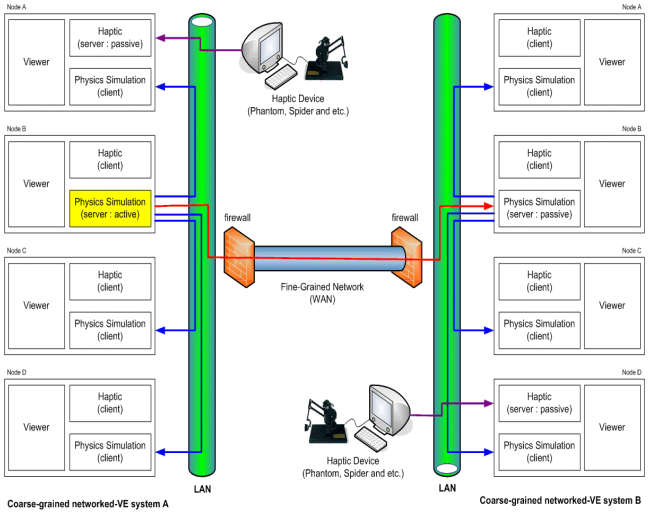


Figure 4 Sharing Dynamic Shared States in Fine-Grained Networked Virtual Environment

In Figure 6, the system A handles the transmission of dynamic shared state to and from the System B, which is based on asynchronous message passing. The physics simulation component is an active server for modification and propagation in system A. The haptic component in system A, allows the physics simulation to send a dynamic shared state including an action in order to control scene-graph. Then, the physics simulation in system B will receive a dynamic shared state and manipulate to virtual object in scene-graph of system B.

This architecture will enhance to overall performance of communication in fine-grained networked VE system and it allows each system to minimize the structure modification in each system. This framework makes the system extensible, reconfigurable and scalable. Therefore, the systems in fine-grained network can extend new functions or interfaces by simply defining dynamic share state without modifying the Kernel. The specification of dynamic shared state is specified in the script file and it can be easily modified to adapt with the scenario of VR contents.

4.5 Other Issues in Distributed Virtual Environment

Visualization is a basic facility in VE. Typically, Loader component with visual rendering in VR system constructs scene-graphs with nodes by reading an XML script. The objects which are created by modeling tools are loaded to compose a "World". Loader component loads the objects and compose the world hierarchically. This is a different way of constructing a world with various objects. The user does not need to modify/recompile the source code to construct a different world. In order to create a new scene-graph, the user only needs to change the XML script, not the source code of the program, and execute it immediately to see the result. In that way, the programmer can save a lot of time in developing VR applications.

Scenario scripting provides a way to control the stories in virtual environment which consist of multiple worlds and avatars. By employing various scenarios, the virtual world environment may be able to provide various contents to users.

Multiple displays can connected to multiple PCs to make clustered-VE systems such as KAVE system or a large surrounded projection screen with stereo for a 3D rendering. The displays in clustered-VE system consisted of one master and the other slaves. Each display host has its own hostname, IP address, and port number. All the computers which are connected to the cluster have the same port number. Display Manager, which is implemented as a component in NAVERLib, identifies the PCs in the cluster by their hostnames in the XML script. View offsets should be set up to be suitable for the visual clustering environment on the PCs. For the above example, although the master computer, node A, has no offset, the other two slave computers' heading offsets are 30 and -30 degrees respectively.

Device manager component is a component for communication with peripheral devices. It may support input and output devices in DIVE. There are five types of device that can be support by Device manager component such as: analog, button, dial, tracker, and force device. Using this component, the user can define the device that will be used.

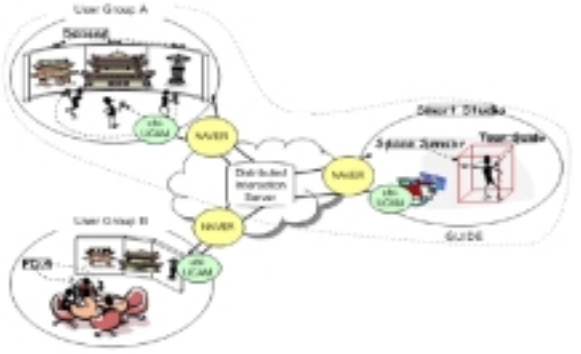
Physics simulation component is a component to simulate the Newton's physics law. Using this component, users can give natural dynamic behaviors to virtual objects in the world. The objects will be governed by the gravity, move under external forces and collide each other. Physics simulation component is implemented using Virtual Physics library. It automatically constructs the physics world from scene graphs loaded by Loader component.

5. Implemented System based on NAVERLib

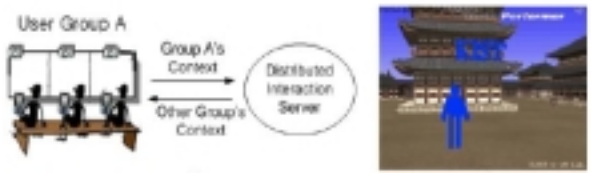
5.1 Context-based Interactive System with Distributed Interactions between Heterogeneous Platforms: GIST

We have implemented context-based virtual heritage system based on fine-grained networked VE [6][7] as shown in Figure 5(a). The virtual system enables users to remotely explore cultural heritage instead of taking a long trip to Kyongju, the real site for cultural heritage of Shilla dynasty. This system allows a guide to steer

user groups in virtual heritage according to scenario describing tour schedule. A guide is able to change the scene by his gestures such as movements of arms or legs, and space sensor senses guide's activities through 3D camera and generates guide's context. The guide service generates commands for movement in virtual heritage by analyzing the guide's context. Users can explore virtual heritage with their own PDA which provides user's profile containing identity, age, sex, and vernacular to context-based services such as language-adapted user interface and heritage-information service. User groups in distributed virtual systems are synchronized by sharing contexts that represent changes at remote nodes. Although the virtual heritage of each user group runs on heterogeneous system, it can be efficiently synchronized with others as shown in Figure 5(b).



(a)



(b)

Figure 5 Context-based Interactive System with Distributed Interactions between Heterogeneous Platforms; (a) Context-based virtual heritage system (b) synchronization among distributed interactive virtual system

5.2 Physics Simulation in KAVE

We implement the physics simulation system in a KAVE system by using NAVERLib with the sharing dynamic state in coarse-grained network.

The KAVE consists of three wall screens, a floor screen (each screen is a square with size 2.2m), four CRT Projectors which display a stereoscopic image to the screens, a haptic device (SPIDAR), a tracker device (ISense900), and a 5.1 channel sound system. The system can provide a sense of reality by displaying auditory and physics simulation. User can observe, touch and manipulate the virtual objects as shown in Figure 6.

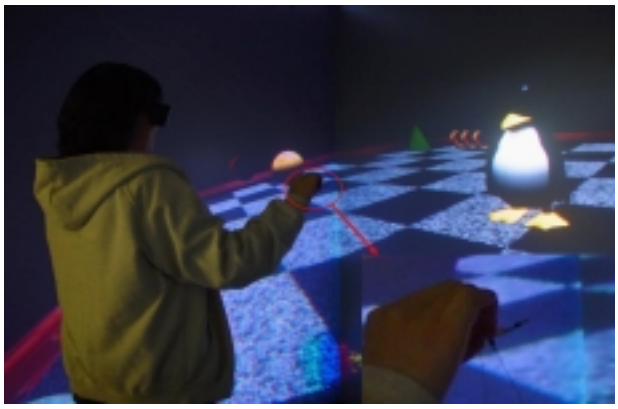


Figure 6 Example of sharing dynamic shared state in KAVE using NAVERLib

6. Conclusion and Discussion

Each component in DIVE system makes it possible to encapsulate complicated procedures of functions into a simpler mechanism, to introduce modularity, to improve managing dynamic shared state between components, between hosts in coarse-grained networked, and between VE systems in fine-grained networked virtual environment. It eventually enhances the overall performance also.

Using this mechanism, the responses from asynchronous and synchronous operations can be processed efficiently. The distributed interactive virtual environment system developer can easily implement state sharing mechanism, not only for a rendering component but also for various components, such as sound management, physics simulation and input/output of peripheral devices.

Finally, the design and the implementation of this dynamic shared state management provide several advantages. First, the system developers can easily and effectively constitute a DIVE. Second, the participants can experience in RCS with enlarged display using multiple projections, a various devices for user interactions, a physic simulation, 3D surround sound, and so on.

In the future, we should improve the way for sharing dynamic state between RCS and consider another type of dynamic shared state such as audio/video streaming data. We also will consider about sharing dynamic state issues with each virtual environment system.

Acknowledgements

This reported work was sponsored by Korea Institute of Science and Technology (KIST) fund under Tangible Space Initiative (TSI) Project.

References

[1] A Software Architecture for Extensible, Flexible and Scaleable Networked Virtual Environments *by C.H Park, Ph.D. Thesis Korea University, 2002*

[2] Networked Virtual Environments *by Sandeep Singhal, Michael Zyda, Addison Wesly, 1999*
[3] Projection-based Virtual Reality : The CAVE and its Applications to Computational Science *by Carolina Cruz-Neira, Ph.D. Thesis, University of Illinois at Chicago, 1995*
[4] A Virtual Platform for Virtual Reality Application Development *by Allen Bierbaum, VR Juggler, MS Thesis, Iowa State University, 2000*
[5] Pattern-Oriented Software Architecture Volume 2. Pattern for Concurrent and Networked Objects *by Douglas Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, Wiley, 1999*
[6] CIVE: Context-based Interactive System for Distributed Virtual Environment *by Seiie Jang, Youngho Lee and Woontack Woo, GIST U-VR Lab, 2004*
[7] NAVER: Networked and Augmented Virtual Environment aRchitecture; design and implementation of VR framework for Gyeongju VR Theater *by C.H Park, H.D Ko, T. Kim, Computers & Graphics 27 (2003) 223–230*