Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

Service-Oriented-Architecture based Framework for Multi-User Virtual Environments

Xiaoyu Zhang and Denis Gračanin
Department of Computer Science
Virginia Tech
Blacksburg, VA 24060, USA

ABSTRACT

Service-Oriented Architecture (SOA) is an application framework used for creating complex enterprise systems by integrating distributed services. The SOA standards are primarily focused on the service composability and data interoperability. Because of the featured capabilities of SOA, it is also used in distributed simulations. However, SOA has its limitations in terms of the performance of real-time message exchanging. In order to address the disadvantages and improve the application performance, we propose a framework that combines the streaming technology and SOA. The proposed framework is used for constructing multi-user Virtual Environment (VE) applications by integrating the application content from distributed services. The additional streaming channels applied to SOA enable the services to actively propagate the real-time messages. The VE applications constructed using the framework have better performance. However, due to the distributed architecture of SOA and the heavy payload of message exchange in the framework, the application performance needs to be evaluated. We describe the metrics used to evaluate the performance and present the evaluation results. Based on the interaction latency we collected from the experiments, we discuss the categories of applications that can fit well in our framework.

1 INTRODUCTION

Internet based virtual world applications, such as Second Life™, have gained more and more popularity recently. The massive multi-user virtual world applications are typical of Distributed Virtual Environments (DVEs). The DVE applications provide computer simulated virtual worlds where users gather together in a shared space and interact with each other. The growing population of the application users brings great challenges for the developers to expand the content of the virtual world. Outsourcing the implementation via open application programming interfaces (APIs) and integrating third party components into the application have proven to be successful in popular web applications, such as Facebook (Facebook 2008) and Second Life (Linden Lab 2008).

Outsourcing the implementation of the application content encourages the participants' creativity which leads to more "attractive" applications. With provided development toolkits and open APIs, the developers can implement and upload their scripts to the application platform where the application loads the scripts as components and runs them on demand. Therefore, the seamless integration of third party components is feasible. However, such approach lacks standardization which limits the reusability of the developed components. Meanwhile, the cost of deploying the components on the target platform limits the ability of the third party developers to fully utilize and maintain their own resources.

Applying Web Service technology to incorporating services as components in the applications has been explored in distributed simulations. The application components are distributed services that maintain their own states and generate messages corresponding to the invocations. Integrating services as application components is the feature of Service-Oriented-Architecture (SOA).

SOA treats services as macro level components for the constructed application (Papazoglou, Traverso, Dustdar, and Leymann 2007). It expands the Web as the open platform for application development and deployment. The service integration and interoperation have been the major focuses for distributed simulations and other SOA applications. SOA has its advantage in terms of cross-platform integration and open standards for interoperation. However, the processing efficiency sets a big performance barrier for SOA because of its distributed architecture and the XML-based messaging. In the meantime, SOA treats most services as stateless services. It is difficult for stateful services to push the updates when their internal states have changed.

Our research is focused on creating a framework using SOA for dynamically constructing DVE applications from

978-1-4244-2708-6/08/$25.00 ©2008 IEEE

services. With the assistance of the framework, the third party developers can contribute the DVE application by providing "active" services. Meanwhile, the framework provides a shared platform where users can gather in a virtual world and seamlessly interact with the content sourced from various services. The framework we propose adapts both the SOA and EDA (Event-Driven-Architecture) and applies the streaming technology to address the real-time interaction challenge. The framework applies standards such as Web Service, X3D, MPEG-4 so that the applications can be developed and accessed with reduced cost.

As the performance of SOA remains a big issue, applications that require real-time response might face performance challenges on our platform. In order to verify the real-time performance of our SOA, we designed an experiment to evaluate the interaction latency of the developed application. Based on the experiment results, we discuss the applications that can fit well on the our framework.

2 Related Work

Using framework in VE application development can greatly reduce the cost by integrating reusable code units as application components. Application centric frameworks (e.g.DIVERSE, Ygdrasil) provide a component-based development and deployment platform (Kelso, Satterfield, Arsenault, Ketchan, and Kriz 2003, Pape, Anstey, Dolinsky, and Dambik 2003). In these frameworks, components developed as dynamic objects are dynamically integrated in the application at the deployment time. Application developers have less effort in the complex application development as reusable components have implemented most fundamental functionalities.

Services are coarse-grained components on the Web versus the dynamic objects which are more fine-grained components on the framework. In real-time 3D visualization applications, services are mainly used as the data sources. Zhang et al designed a DVGE system to integrate data services (e.g. data imaging service, 3D object data service and geo-model service) for creating collaborative geo-information applications (Zhang, Gong, Lin, Wang, Huang, Zhu, Xu, and Teng 2007). Likewise, the framework we proposed treats services as coarse-grained components in our framework (Zhang and Gračanin 2007).

A *Web Service* is defined as a functionality that can be programmatically accessible via the Web (Tsur, Abiteboul, Agrawal, Dayal, Klein, and Weikum 2001). A fundamental objective of Web Services is to enable the interoperability among different software applications that run on a variety of platforms (Medjahed, Benatallah, Bouguettaya, Ngu, and Elmagarmid 2003, Vinoski 2002). The interoperation has been enabled by the tremendous standardization effort to describe, advertise, discover, and invoke Web Services (Curbera, Duftler, Khalaf, Nagy, Mukhi, and Weerawarana 2002). Web Services are increasingly being adopted as a framework to access Web-based applications.

SOA treats services as more coarse-grained and heterogenous components for constructed applications (Papazoglou, Traverso, Dustdar, and Leymann 2007). Incorporating SOA in the real-time simulations has been explored in XMSF (Blais, Brutzman, Drake, Moen, Morse, Pullen, and Tolk 2005) and other projects (Chen, Cai, Turner, and Wang 2006, Tsai, Fan, Chen, and Paul 2006). These projects are mainly focused on exploring the interoperability between simulation components by using Web Service technology.

Even though most simulations require critical real-time communication, the performance has been rarely evaluated in their infrastructures. Revised SOA such as RTSOA (Real-Time Service Oriented Architecture) focused on the efficiency of service composition instead of the application performance running on the architecture (Tsai, Lee, Cao, Chen, and Xiao 2006).

One of the major factors preventing widespread use of SOA is performance. The performance of Web Services is affected by the process of exchanging information via service invocation(Zilora and Ketha 2008). That process includes the following steps: request construction, SOAP message construction, transmission, server listening, SOAP message deconstruction, and request processing.

Some argue that the SOAP protocol is too heavyweight (Suzumura, Takase, and Tatsubori 2005). Other approaches include using more tightly-couple protocols (RMI, CORBA) (Juric, Kezmah, Hericko, Rozman, and Vezocnik 2004) or using streaming message exchanges (Oh and Fox 2007). Another alternative approach is using active streaming servers to push the real-time data (Comet 2008). Extending the Web Service infrastructure by adding streaming channels to deliver time sensitive data (e.g.multimedia data) has been developed in IBM (IBM alphaWorks 2006).

Based on the reviewed literature, we learned that incorporating streaming to compensate the service data interaction could take the advantage of both the SOA and real-time streaming technology. Therefore, we extend our framework on SOA for constructing DVE applications from services. As there is little effort in investigating the impact of the latency in SOA for real-time 3D applications, we introduce our SOA incorporated framework and evaluate the performance of the real-time interactions for the VE applications.

3 Framework

The proposed framework is designed to support the following functions: 1) dynamic application content integration, 2) a unified 3D presentation for application content accessing, and 3) a shared multi-user virtual space. The 3D application content is provided by the third party services. The services are developed independently and distributed over

the network. The content of the application is dynamically assembled by composing the services. The users can access the application content with 3D browsers.

In our framework, the services from third party providers are called the *component services*. Each component service provides two sets of interfaces. The 3D application content from the services is delivered through a streaming interface. The controls that manipulate the content are exposed via Web Service interface. The 3D content from the services are collected and integrated into an application by a centralized server known as the *container*. A descriptive scene graph (described in X3D) is used among the container and component services. The users access the constructed application through standalone X3D/MPEG-4 browsers.

The architecture of the framework is shown in Figure 1. The dashed lines indicate the streaming data.

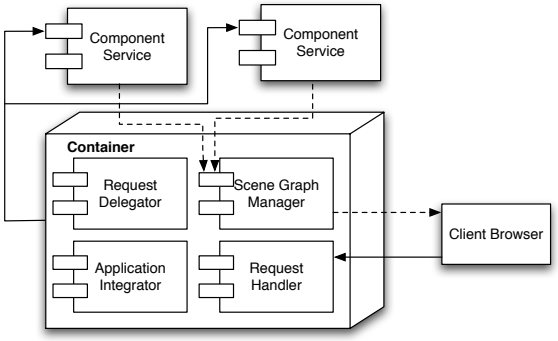

Figure 1: The framework architecture

The design of our framework is based on the following principles. We introduced a distributed Model-View-Controller (MVC) model to indicate the function design of the services and the container. By following this model, the control commands are separated from the application presentation updates in the data stream exchanged between the container and the services. The framework adapts both the SOA and EDA (Event-Driven-Architecture) to drive the service integration and interoperation. The seamless integration is addressed by introducing an ontology to abstract the application domain knowledge. The streaming technology is applied to reduce the interaction latency and enables the services to actively push the updates.

3.1 Distributed MVC

Model-View-Controller (MVC) is the classical paradigm used in user interface (UI) design (Gamma, Helm, Johnson, and Vlissides 1995). The MVC paradigm can be adapted in a distributed format for the applications that are composed from distributed components, as shown in Figure 2.

In the distributed MVC model, the application view is assembled in a centralized unit as the *container*. It is a

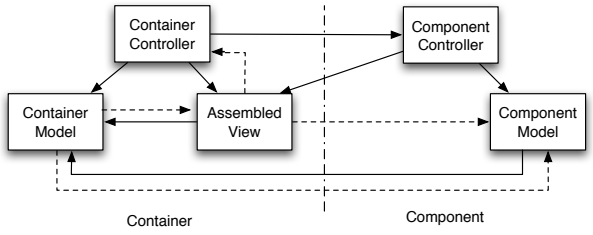

Figure 2: Distributed MVC pattern

composition of the views from the components. In order to hide the data details and hold the data privacy, the data model for each individual component is not allowed to be accessed directly. The data model for the composed application is built from the component data model though their controllers. The container holds no direct controls of updating the component views. When a component view needs to update, the controller of the container will delegate the requests to the component controller. The update messages will be propagated to the container and changes will then apply on the assembled view.

Based on the distributed MVC model, the controller of each component is the interface to access both the UI data and the data model data. The controller of the container interacts with the local data model as well as the controllers from other components. The controller for each component should be well designed so that others can retrieve the data model data, UI data, or request for the state updates on the components.

3.2 Event Driven Architecture

The container is a composer that integrates the distributed services. It includes the application logics to coordinate the interoperation between the services and the container. As most UI applications, the composed application is driven by the events defined in the application logics. The application logic contains the definition of the events and the actions corresponding to the events. In this way, the framework combines the event-driven architecture in the SOA for the component coordinations.

3.3 Application Content Streaming

The application content for VE applications is generally presented in scene graph. In the framework, eXtensible 3D (X3D) is used to describe the scene graph and the UI interactions. The application content in X3D can be delivered over the network. X3D is an ISO standard for XML-enabled 3D file format aimed to facilitate the interactively manipulating, communicating and displaying scenes (Web3D Consortium

2008). The advantage of using X3D is that users can use a standard X3D browser to access any X3D applications.

The UI on the client side is mirroring a part of the shared scene graph on the container side. The updates of the application are rendered on the clients by applying the scene graph updates from the container. In order to improve the performance and applying standards, we introduce the additional streaming channel using MPEG-4 for the scene updates propagating. MPEG-4 includes BIFS (BInary Format for Scene) as a part of its standard (Walsh and Bourges-Sevenier 2002). BIFS is designed incorporating with X3D standard. Using MPEG-4 streaming, X3D UI data and updates can be well synchronized and streamed from a peer to another. Therefore, X3D is used as the media for the application content in our framework.

3.4 Ontology for Integration

An ontology can be used in application modeling as it provides the common vocabularies, nomenclatures and taxonomies linked to detailed information resources. We use ontology to abstract the knowledge that describes the relationship and the organization of the application elements. The application elements are associated with the features of how the application is modeled. We use an ontology an a top-down approach to coordinate the service integration.

The component services are defined as the instances of the application elements in the ontology. The integration and intercommunication between the elements is done by the event-driven architecture in the framework. The ontology abstracts the concepts of events, rules, actions, and their relationships that are used in the EDA. The knowledge of multi-user controls (e.g. access control and view consistency control) are also included in the ontology.

Based on the ontology, the semantic information for an application instance of a certain domain is modeled into what we call a *profile*. The profile defines the agreement among the components providers. Derived from the ontology, we developed a terminology used for component behavior descriptions and application task descriptions. Incorporating ontology in our framework, a multi-user 3D application can be seamlessly constructed from various component services.

4 Framework Workflow

The major focus of this paper is to give a performance measurement of our proposed framework. In order to present a clear picture of where the performance thresholds are, we will first illustrate the workflow in the framework. Figure 3 shows how the framework supports composing and running a 3D application. There are four main activities in the framework.

The first activity is application composition. As shown in Figure 3, the application is composed before users access

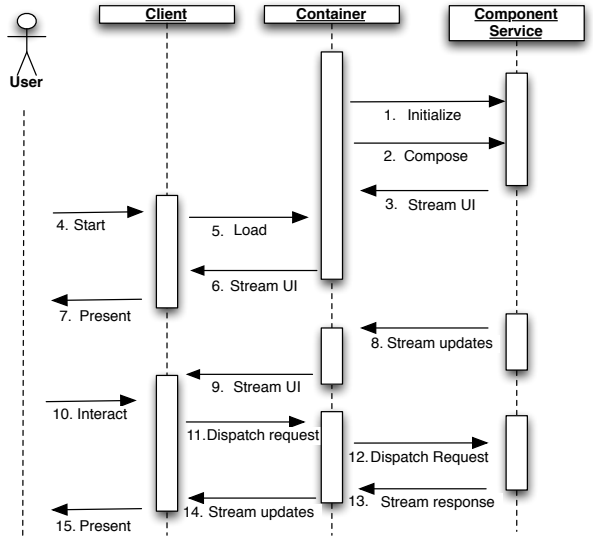

Figure 3: The sequence diagram of the framework workflow

the application. The container initializes the composition by sending messages to all the available services (message 1) and calling the corresponding methods for application content (message 2). The UI data is streamed in MPEG-4 format and delivered to the container (message 3).

The second activity is application accessing. The user starts the client application to load the scene from the container (message 4 and 5). The container generates a subject view for the users based on user's context, and it streams the UI data to the client. As the client is an interactive X3D browser or MPEG-4 player, it renders the UI and presents the application to the user (message 6 and 7).

The third activity is active service-status updating. Whenever the status of a service is updated, it pushes the updates to the container without the probe messages from the container (message 8). The container will multicast the updates to the clients that are interested in the UI changes (message 9).

The forth activity is the most important activity. It is the user interaction activity. When the user interacts with the client and triggers a UI request, the message will be sent to the container (message 10 and 11). The container decides which service holds the responsibility of handling the request and forwards the message to it (message 12). Once the service completes handling the request, it streams the UI updates to the container (message 13). The container evaluates the change, creates the corresponding updates on user's subjective view, and streams the UI updates to the client (message 14). The client shows the updates when it receives the messages (message 15).

As we can see in Figure 3, the request messages are rerouted twice before arriving to the component service. Meanwhile, the responses arrives to the client after being forwarded by the container. Extra latency is introduced because of the distributed characteristic of SOA. As any of the latency happened in the user interaction activity will impact the application usability, we will only focus on evaluating the performance of this activity. In our measurement, we will take the response latency of the user interaction as the benchmark of the application performance.

5 Performance Evaluation

The distributed architecture of the framework provides flexibility and scalability but it sacrifices the runtime performance. The extra network latency between the container and the service providers might have impacts on the real-time performance of the composed application. Fast response to user's interaction is an important factor in the human information processing loop. The delayed response can break the information processing loop and cause the presence distortion. Therefore, it is critical to evaluate the response latency to user's interaction for the real-time VE applications.

We use several techniques in the framework to improve the performances. In order to present an object view of the interaction performance for the applications constructed using the framework, we have defined a set of metrics for the real-time performance evaluation.

5.1 Metrics

We introduce the concept of *interaction latency* to measure the system performance. The interaction latency is the time interval from when the user issues the control to the time when the user perceives the updates from the system. That is the time interval between message 10 and message 15 in Figure 3.

The interaction latency (T_i) includes three parts: the *network latency*, *marshaling cost* and the *process latency*. The network latency (T_n) is the time consumed for network transport. The marshaling cost (T_m) is the time spent in buffering or parsing the messages. The process latency (T_p) is the time used (either on the container or service providers) from when a stimulus is received to the time when the response is issued.

The interaction latency is the objective metric to measure whether the system performance meets the human perception criteria. The latency is expected to be within 1.0s which is an acceptable usability criteria for most 3D applications. The network latency and process latency are used for the resource impact analysis. The marshaling cost is used for analyzing the efficiency of network communication protocol.

We use the following methods to get T_i, T_n, T_m, and T_p. We log the time on the client, container, and component service. For example, on the client side, when message 11 is issued, we log the time as $tclient_{11}$. When the message is parsed and received on container, we record the time as $tcontainer_{11}$. After the container sends message 12 to the component service, $tcontainer_{12}$ is recorded on the container. Time stamp $tservice_{12}$ is recorded on the service when the message 12 is received and parsed. Therefore, we can calculate the interaction latency (T_i) from network latency between client and container (T_{nCC}), network latency between container and service (T_{nCS}), marshaling cost in interpreting messages between client and container (T_{mCC}), marshaling cost in interpreting messages between container and service (T_{mCS}), process latency in container (T_{pC}), and process latency in service (T_{pS}). The data gathered for each of the parameters can be used to analyze the network and the computing power impact, providing insights for the future performance enhancements.

5.2 Preliminary Study Result

We have carried out several preliminary studies on our available network condition, the performance of service invocations, and the performance of various service implementations. The network used in the study using has very good performance. The network latency between the hosts is mostly within 10ms.

The performance of a Web Service invocation depends on the processing time on the service side. The payload of marshaling/unmarshaling for SOAP messages is very low with simple method calls. Most of the latency for the light weight service invocations are equal to the network latency. The real performance criteria comes from the software implementation and levels of service invocations. In the service implementation, if inter-process communication is used for forwarding the Web Service invocations to a 3D streaming application, the processing time on service can be two time expensive than using light-weight threads communication. Using streaming over RTP requires additional cost on buffering and marshaling. If a client request requires additional interoperation between services, the cost can be very expensive.

Table 1 compares different cost based on our observation. In the preliminary study, we implemented two level invocations (service to service invocation) using the inter-processing communication to see the worst possible performance (4223ms).

5.3 Evaluation Design

The experiment is implemented using a simple application created using the framework. The application provides a VE which hosts various virtual objects sourced from different service providers.

Table 1: Preliminary Study Results

Cost	Time (ms)
Network Latency	8
Service Processing Time (Threads)	358
Service Processing Time (Inter-Process)	999
One-Level Service Invocation	2849
Two-Level Service Invocation (Service)	4223

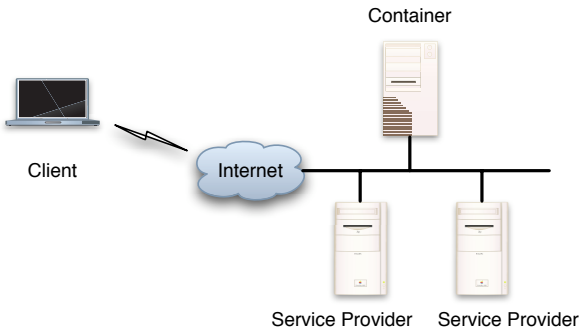

Figure 4: Experiment Environment

The interaction task is to rendering an information board for a 3D object. When the user moves the mouse over a 3D cube in the virtual world, a board with the object information will be displayed. The board data is loaded based on the scene updates from a service. This interaction task mirrors the activity of dynamic loading and replacing/adding virtual object in the virtual environments. The interaction requires one-level component service invocation to accomplish the task.

The application is created using the GPAC (Feuvre, Concolato, and Moissinac 2007) toolkits and AXIS Web Services. The service is using thread communication. The scene update (BIFS update in MPEG) is streamed using RTP protocol. The experiment environment is shown in Figure 4. The container server is a Dell Pentium D 2.0G Windows machine with 2G RAM. The component service that hosts the object scene data is Dell Pentium 4 3.2G Windows machine with 2G RAM. The client is a 1.8G Windows laptop with 512M RAM. The client laptop connects the Internet using a wireless connection. The container and service providers are connected to the Internet with gigabyte intranet.

5.4 Evaluation Result

The results are shown in Table 2. As the network latency is very small, we ignore it from the experiment results. The values in Table 2 are the average value of 40 trials. The total

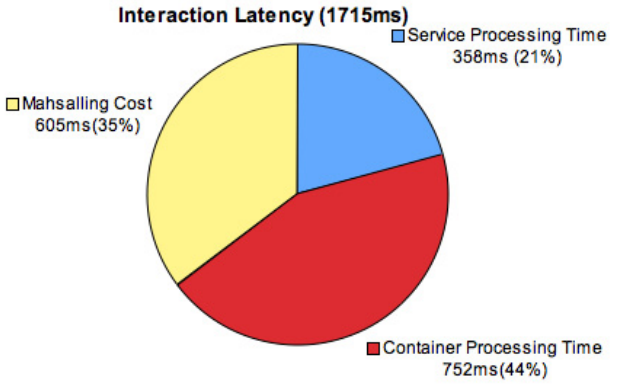

Figure 5: The Ingredient of Interaction Latency

interaction latency is about 1.7 seconds. The ingredient of the interaction latency is shown in Figure 5.

Table 2: Experiment Results

Response Latency	Time (ms)
Interaction Latency	1715
Marshaling Cost	605
Process Latency (Container)	358
Process Latency (Service)	752

From Figure 5 we can tell that the processing latency has been the major part of the delay. Both the component service and the container took around 1s (65% of the interaction latency) which greatly impact the performance. The message marshaling also pretty expensive. It takes 35% of the time in the interaction latency. By taking a deeper look into the marshaling cost, we discover that 59% of the latency (around 0.3s) is from the service to the container(shown in Figure 6). The difference between container to service invocation and client to container invocation is that our client does not use Web Service to send the requests.

6 Discussion

The SOA architecture has its advantages in terms of composibility. However, the distributed architecture might have an impact on the real-time performance. The experiment results show that the network latency and message marshalling/unmarshalling can affect the interaction response. Therefore, the applications that developed using SOA require special tolerance of the delayed responses.

Even though the framework is facing challenges on the real-time performance because of its SOA, there are several categories of applications that fit smoothly in the framework. The streaming technology has its advantages

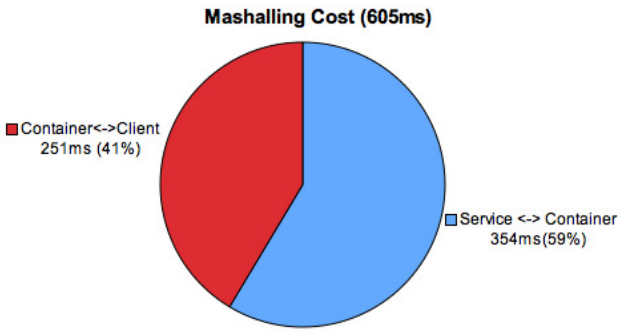

Figure 6: The Ingredient of Network Latency

in broadcasting time-sensitive data to a broad audience. Therefore, the applications developed on our framework can also benefit from the feature. Applications with the following characters can be created using our framework.

3D Live Broadcasting

3D live applications can be used to broadcast a concert, game, or lecture. Applications of such 3D live shows normally have a short life cycle. The application ends when the live session is over. Another feature of such applications is the goal to attract the largest audience possible. Instead of being interested in identifying who the users are, they care more about how many the users are. On the other hand, users still have the capability of accessing general information about the show via simple interactions. Such interaction allows a little more latency which has little impact the presence of the virtual reality.

The applications have short life cycle and need to attract big numbers of audience. Our framework provides a suitable platform for these applications because the composed application is generally a 3D portal. Many users have been gathering in the portal application and exploring the applications in the virtual world. The live broadcasting application can dynamically plug into the portal application and deliver the live content to the users.

Dynamic Animation

Playing dynamic animations on demand is another type of applications that are capable for our framework. The animation of the application is unpredictable, and it changes based on the internal status of the service or the request parameters from the users. That means the animation cannot be cached either on the container or on the client side application. Once users start interacting such applications, few consecutive demands will be issued.

Such applications have continuous responses to users' single requests. Therefore, users are expecting the latency as the online VOD (video on demand) applications which they are already familiar with. Another advantage of using the framework is that it supports different rendering types. The animation can be rendered either just for a user or for all the users accessing the application. The service provider only needs to declare which rendering type it requires the container to support. The container can build the scene graph data as either the part of the global scene graph or the scene graph for an individual view.

Virtual Touring

For massive multi-user DVE applications, most activities in the virtual world is the virtual world exploring and peer to peer communications. Apart from the peer to peer interaction, the activity users spent most time in the virtual world is navigation. Navigation is a type of 3D user interaction which requests few frequent interactions with the environment. Applications designed with navigation intensive tasks to explore complex 3D models can be integrated using our framework.

An example of such application is a virtual museum which offers an automatic tour to the users inside the museum. When a user starts the tour, her viewpoint is automatically changed by following the predefined the path. The activity inside the museum is to explore the virtual objects by following the consecutive commands streamed from the service. Whenever the service developers need to replace the virtual objects or change the paths, they can apply the new service immediately without bothering the users to update their applications.

7 Conclusion

Implementation outsourcing brings more opportunities for the VE applications to integrate resources and creative implementations. Our framework applies SOA to integrate services as the coarse-grained components for the massive VE applications. The infrastructure integrates several methods such as ontology and streaming to address the seamless integration requirement and the performance challenges.

The pilot study reveals the real-time performance for the constructed applications on the distributed architecture. It also brings up the discussion on what the type of applications fit in the distributed architecture with the trivial interaction latency. More studies of the framework performance improvement and the user study on various application context will be carried out.

REFERENCES

Blais, C., D. Brutzman, D. Drake, D. Moen, K. Morse, M. Pullen, and A. Tolk. 2005. Extensible modeling and simulation framework (XMSF) 2004 project summary report. Technical report, Naval Postgraduate School, Monterey, California.

Chen, X., W. Cai, S. J. Turner, and Y. Wang. 2006. SOAr-DSGrid: Service-Oriented Architecture for Distributed Simulation on the Grid. In *PADS '06: Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 65–73. Washington, DC, USA: IEEE Computer Society.

Comet, W. 2008. Comet (programming). http://en.wikipedia.org/wiki/Comet_(programming) [Last accessed: Apr. 2008].

Curbera, F., M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. 2002. Unraveling the web services web: An introduction to soap, wsdl, and uddi. *IEEE Internet Computing* 6 (2): 86–93.

Facebook 2008. Facebook developers. http://developers.facebook.com [Last accessed: Apr. 2008].

Feuvre, J. L., C. Concolato, and J.-C. Moissinac. 2007. GPAC: open source multimedia framework. In *MULTIMEDIA '07: Proceedings of the 15th international conference on Multimedia*, 1009–1012. New York, NY, USA: ACM.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1995. *Design patterns: elements of reusable object-oriented software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

IBM alphaWorks 2006. IBM web service streaming engine. http://www.alphaworks.ibm.com/tech/streamingengine [Last accessed: Feb. 2008].

Juric, M. B., B. Kezmah, M. Hericko, I. Rozman, and I. Vezocnik. 2004. Java rmi, rmi tunneling and web services comparison and performance analysis. *SIGPLAN Not.* 39 (5): 58–65.

Kelso, J., S. G. Satterfield, L. E. Arsenault, P. M. Ketchan, and R. D. Kriz. 2003. DIVERSE: a framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence: Teleoper. Virtual Environ.* 12 (1): 19–36.

Linden Lab 2008. Second Life: Official site of 3D online virtual world. http://secondlife.com/[Last accessed: Apr. 2008].

Medjahed, B., B. Benatallah, A. Bouguettaya, A. H. H. Ngu, and A. Elmagarmid. 2003. Business-to-Business Interactions: Issues and Enabling Technologies. *The VLDB Journal (to appear)*.

Oh, S., and G. C. Fox. 2007. Optimizing web service messaging performance in mobile computing. *Future Gener. Comput. Syst.* 23 (4): 623–632.

Papazoglou, M. P., P. Traverso, S. Dustdar, and F. Leymann. 2007. Service-oriented computing: State of the art and research challenges. *Computer* 40 (11): 38–45.

Pape, D., J. Anstey, M. Dolinsky, and E. J. Dambik. 2003. Ygdrasil: a framework for composing shared virtual worlds. *Future Gener. Comput. Syst.* 19 (6): 1041–1049.

Suzumura, T., T. Takase, and M. Tatsubori. 2005. Optimizing web services performance by differential deserialization. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services*, 185–192. Washington, DC, USA: IEEE Computer Society.

Tsai, W.-T., C. Fan, Y. Chen, and R. Paul. 2006. DDSOS: A dynamic distributed service-oriented simulation framework1. In *ANSS '06: Proceedings of the 39th annual Symposium on Simulation*, 160–167. Washington, DC, USA: IEEE Computer Society.

Tsai, W. T., Y.-H. Lee, Z. Cao, Y. Chen, and B. Xiao. 2006. Rtsoa: Real-time service-oriented architecture. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, 49–56. Washington, DC, USA: IEEE Computer Society.

Tsur, S., S. Abiteboul, R. Agrawal, U. Dayal, J. Klein, and G. Weikum. 2001, September. Are Web Services the Next Revolution in e-Commerce? (Panel). In *VLDB Conference*.

Vinoski, S. 2002, February. Web Services Interaction Models, Part 1: Current Practice. *IEEE Internet Computing* 6 (3): 89–91.

Walsh, A. E., and M. Bourges-Sevenier. 2002. *The MPEG-4 jump-start*. Prentice Hall Professional Technical Reference.

Web3D Consortium 2008. Communicating with real-time 3D across applications, networks, and XML web services. http://www.web3d.org/[Last accessed: April 2008].

Zhang, J., J. Gong, H. Lin, G. Wang, J. Huang, J. Zhu, B. Xu, and J. Teng. 2007. Design and development of distributed virtual geographic environment system based on web services. *Inf. Sci.* 177 (19): 3968–3980.

Zhang, X., and D. Gračanin. 2007. From coarse-grained components to dve applications: a service- and component-based framework. In *Web3D '07: Proceedings of the twelfth international conference on 3D web technology*, 113–121. New York, NY, USA: ACM.

Zilora, S. J., and S. S. Ketha. 2008, March. Think inside the box! Optimizing web services performance today [web services in telecommunications, part ii]. *IEEE Communications Magazine* 46 (3): 112–117.

AUTHOR BIOGRAPHIES

Xiaoyu Zhang is a PhD student of Computer Science Department at Virginia Polytechnic Institute and State University. His research focuses on the distributed virtual environment, Web 3D, computer supported collaborative work. His email address for the proceedings is <zhangxy@vt.edu>.

Denis Gracanin is an Associate Professor of Computer Science at Virginia Polytechnic Institute and State University. His research interests include virtual reality and distributed simulation. He is a member of AAAI, ACM, IEEE, SCS, and SIAM. Heis email address for the proceedings is <gracanin@vt.edu>.