ATLAS – A Scalable Network Framework for Distributed Virtual Environments

Dongman Lee, Mingyu Lim, Seunghyun Han
Information & Communications University
58-4 Hwaam-dong, Yusung-ku
Daejeon 305-732, Korea
+82-42-866-6163

{dlee, cats, dennis}@icu.ac.kr

ABSTRACT
A distributed virtual environment (DVE) is a software system that allows users on a network to interact with each other by sharing a common view of their states. As users are geographically distributed over large networks like the Internet and the number of users increases, scalability is a key aspect to consider for real-time interaction. Various solutions have been proposed to improve the scalability in DVE systems but they are either focused on only specific aspects or customized to a target application. In this paper, we classify the approaches for improving scalability of DVE into four categories: communication architecture, interest management, concurrency control, and data replication. We then propose a scalable network framework for DVEs, ATLAS. Incorporated with our various scalable schemes, ATLAS meets the scalability of a system as a whole. By providing system developers with a set of APIs as a network infrastructure, ATLAS intends to support various applications The integration experiences of ATLAS with several virtual reality systems ensures the versatility of the proposed solution.

Categories and Subject Descriptors
C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *client/server, distributed applications.*

General Terms
Design, Management.

Keywords
DVE, scalability, network framework.

1. INTRODUCTION
A distributed virtual environment (DVE) is a software system that allows real-time interaction among geographically distributed users by providing various levels of sharing in terms of space, presence, time, etc [23]. A shared context among users is achieved by sharing each user's activities to the rest of users and often enhanced by replicating the information at each user's site. However, as users are geographically distributed over large networks like the Internet and the number of users increases, scalability is a key aspect to consider for supporting real-time interaction [17].

Various attempts have been made to improve scalability in DVEs. A key design issue in them is to reduce message exchange as much as possible in terms of number and size without harming the shared context and interactive performance. A most typical approach is filtering unnecessary messages by dividing a virtual world into several regions [1,18] or localizing the area of interest of users [2,7,9]. However, if the filtering is solely done by a server, it burdens the server with a flood of messages and thus increases network delay. To avoid this, peer/peer or peer/server models are adopted with multicast support [1,2,9,18]. Another way to reduce message exchange is to replicate virtual world data at the client from the server. A key concern here is how to efficiently reduce transmission delay of the virtual world data [3,5]. However, replication requires synchronization among replicas in the presence of multiple concurrent updates, which eventually lead to inconsistent views among users. This results in the need of concurrency control. It should provide acceptable interactive performance and consistency [21]. In summary, key considerations for scalability improvement in DVEs are interest management, communication architecture, data replication, and concurrency control.

Existing systems have proposed various solutions to the scalability problem but their approaches are limited to specific applications and/or do not cover all four aspects of scalability considerations. NPSNET [18], which is primarily aimed to military simulations, attempts to increase scalability not only by using spatial filtering mechanism but also by adopting the peer-to-peer communication architecture. However, it pays little consideration on data replication or concurrency control schemes. MASSIVE systems [2], which are designed for teleconferencing, reduce communication overheads by leveraging spatial relationship among participants or by providing aggregated group of view. They also do not consider world data replication or concurrency control mechanisms. Both approaches assume that a whole world data is replicated at each client and a pessimistic concurrency control is used. DIVE [9], which is designed to support 3D teleconference, introduces the aura concept to support natural user interaction and to reduce the communication cost, and supports heterogeneous network requirements based on DiveBone [6]. However, it overlooks scalable concurrency control and data replication. PaRADE [21] is an exemplary system which attemps to improve interactive performance in concurrency control by using a prediction-based scheme. It, however, does not provide scalable message filtering and data replication. QUICK [3] framework takes into account dynamic objects distribution to participants without considering concurrency control or interest management. Bamboo [24], provides interfaces for device management, networking, graphical user interface, and extensibility using callbacks to application designer. It, however, delegates to application developers the mechanisms for interest management, data replication, and concurrency control.

Advances in network and 3D graphics technologies have led to growing demands for various DVE applications such as virtual community, collaborative engineering, network game, and so on. In this paper, we propose a scalable network framework for DVEs, ATLAS. It provides various scalable schemes in all four scalability aspects described above, not restricted to specific ones. For meeting various DVE application requirements, these schemes are provided to system developers as APIs. Intending to support various applications, ATLAS introduces an intermediate layer playing a role of routing and transforming messages between ATLAS and applications. We have succeeded to integrate ATLAS with several applications, which ensures the versatility of the proposed solution.

The rest of the paper is organized as follows. In section 2, we discuss four scalability considerations in large DVE systems. Section 3 introduces the proposed network framework for DVEs, ATLAS, and its internal modules in detail. Section 4 illustrates our integration experiences of ATLAS with various applications. Conclusion with future work follows in Section 5.

2. SCALABILITY CONSIDERATIONS

In this section, we discuss the four key design issues that should be considered for scalability of DVEs.

2.1 Communication Architecture

Since a DVE provides users with a shared context by exchanging their states with each other, how to reduce communication overhead is a key design consideration. Depending on how the communication is coordinated, the communication architecture can be characterized as follows: client/server model, peer/peer model, and peer/server model.

In the client/server model, all the messages are sent to a server and then the server distributes them to all or some of users according to synchronization requirements. Apparently it is not scalable as the number of participants in a virtual world increases since the server becomes a bottleneck. Even if additional servers are used [7], the delay due to additional communication overhead in servers is inevitable. To avoid this, the peer/peer model is introduced. It allows users to directly exchange messages. However, each user has to assume all the responsibility of message filtering and synchronization. The peer/server model exploits the benefits of two models described above such that consistency management is done by a server and communication among users is performed directly by themselves using multicast.

2.2 Interest Management

Though the computing powers and rendering speed are rapidly increasing, network resources still remain very expensive compared with computational resources. To overcome the limitations of the network resources, various relevance-filtering mechanisms should be considered. Since users need not receive all update messages related to the whole world, they instead receive only messages which they are interested in [8,23]. The interest management is divided into two methods according to the fidelity of capability of message filtering: dividing a virtual world into several regions and localizing the area of interest of the participants [16].

2.3 Concurrency Control

Shared information in DVEs is often replicated at each user's site to provide acceptable interactive performance, especially where users are geographically distributed over large networks like the Internet. Replication enables users to locally access and update the data. On the other hand, the cost of replication is to maintain synchronization among replicas in the presence of multiple concurrent updates, which eventually lead to inconsistent views among users. As communication delay increases, the probability of conflicts between operations does as much. Therefore, concurrency control is required to maintain synchronization among replicas. Approaches to concurrency control have been broadly categorized into pessimistic, optimistic and prediction scheme [25].

The pessimistic scheme blocks a user until a lock request for an object is granted and then allows him to manipulate the object. It is simple and guarantees absolute consistency. However, when communication delay becomes high due to increase of the number of users, they suffer from the long response time, which in turn deteriorates the interactive performance.

The optimistic scheme allows users to update objects without conflict checks with others and thus to interact with objects more naturally. However, when conflicts occur, a repair must be done. This not only makes systems complex but also perplexes users with undoing and redoing of previous actions on user interfaces.

The prediction based concurrency control is to allow users to lie in the optimistic scheme as well as to eliminate the need for repairs like the pessimistic scheme to support real time interaction. The owner of an object predicts the next owner before users request ownership of the object based on the position, orientation, and navigation speed of the users. The key to the prediction based concurrency control is to accurately predict a correct one among several users requesting an ownership and grant the ownership to it in time.

2.4 Data Replication

For supporting real time interaction in DVEs, it is common to replicate virtual world data from the server at the client. Each client then updates its replicated data by local changes or notification of remote changes. As the size of the virtual world increases, it becomes significant transmission overhead increasing initial download delay of the virtual world data, especially when to replicate a whole virtual world to the client. To reduce the overhead, several on-demand transmission (partial-replication) techniques are devised [3,5]. In these techniques, instead of downloading the whole virtual world objects into the clients' machines, copied are only objects that the user needs [17]. A key aspect in partial replication is how to efficiently replicate the required data lest that user's immersion in virtual world be disturbed for the loss of data. For efficient replication, two schemes are used together in general – prioritized transfer of objects, and a caching and prefetching techniques [19].

3. ATLAS

In this section, we describe how each scalability issue discussed in the previous section has been dealt in ATLAS and its detailed architecture.

3.1 Scalability Support

3.1.1 Communication Architecture

ATLAS supports the peer/server model as a primary communication architecture. A server joins several multicast addresses assigned to regions and maintains the membership of users and the states of a virtual world. Each participant multicasts its update messages directly to other participants in its region and its neighboring regions as well. ATLAS also supports the client/server model for versatility. This model is useful when web-based DVE applications and multicast is not available.

3.1.2 Interest Management

We assume that a whole virtual world is divided into several logical regions and that each participant can communicate with other participants in a region to which he belongs and those in neighboring regions. We support the interest management scheme based on user interests and spatial distance [10]. We leverage human heuristic such that, for instance, in a virtual shopping mall, users often tend to move to and crowd specific places with their own interests and to interact with those who have similar interests. In our interest group based filtering scheme, the users of the same interests dynamically form a group when they get close. Each user in the group multicasts update messages to the rest of the group whenever he moves or interact with the world. On the other hand, when the group is included or collided with the interest area of a user who is not a member of the group, that is, not sharing the same interests, the representative of the group sends the aggregated update information of the group with low frequency to the user. It can enhance the interactive performance as the number of users in DVE increases and crowds in a specific place.

We also support inter-region interactions with a more scalable manner [16]. While a few systems [1,2,18] support inter-region interactions, users have to pay the price: they must be always informed of the status of all the users in neighboring regions some of whom they are not interested in. This imposes communications overhead on the users who wish to pursue interactions with other users across regions and thus makes the system less scalable. The region manager selects only a subset of users from the neighboring region whose members have high possibility of interaction with users in the current region. This subset of users forms another multicast group. This enables users in the region not to receive all the update messages from the neighboring region. They receive the update messages regarding only the users in whom they are interested in the neighboring region.

3.1.3 Concurrency Control

The prediction based concurrency control is to allow users to lie in the optimistic scheme as well as to eliminate the need for repairs like the pessimistic scheme to support real time interaction. It is suitable to allow real time interactions for users. In the existing approach [21], users wishing to obtain an ownership multicast the ownership requests with their predicted collision time to all the potential owners for distributed control. The owner of an object collects only the requests sent to it and performs prediction for determining the next owner based on the predicted collision times. However, as the number of users and objects in a virtual world increases, the owners and the network may become overwhelmed. This causes the owner to fail to transfer the ownership to the next owner in time because of the delayed request processing time.

We developed the entity-centric prediction based concurrency control scheme [25] that satisfies the needs for scalability in terms of interactive performance as the number of users increases. Only the users surrounding a target entity multicast the ownership requests by using the multicast group address assigned to the entity. The number of messages per owner decreases drastically since the owner receives only from users joining the entity multicast group instead of from all users in the same region. This reduces network bandwidth consumption. Therefore, an owner makes prediction with the reduced number of messages, which, in turn, results in a short request processing time. It allows the owner to determine the next owner and pass the ownership in time.

Our scheme improves performance by grouping closely gathered entities into one entity group and sharing a multicast address among group member entities [13]. This reduces the number of frequent join and leave operations, and maintains enough interactive performance. Another enhancement supports users with various navigation speeds [14]. It allows as many Entity Radii as the number of different speed and allocates a separate queue for users of each speed. Each queue is examined in parallel to predict the next owner candidate and among the selected candidates is chosen the final candidate, which contributes to the timely advanced transfer of ownership by using appropriate Entity Radius based on a user's speed.

3.1.4 Data Replication

Several data replication schemes [3,5] using partial replication do not handle a scalability problem efficiently. The spatial relationship used in the existing approaches just guesses the user's behavior from proximity between the user and objects. This makes it difficult to determine which types of objects are more important to an individual user, not reflecting the user's interests – a significant factor affecting the user's behavior. Since the diverse types of objects become existent as the number of objects in a virtual world increases, it is more difficult to correctly predict the user's behavioral pattern according to the types of objects.

For efficient data management of a virtual world, we have proposed a scheme using user-based caching and prefetching exploiting the object's access priority generated from spatial distance and individual user's interest in objects in DVEs [19]. The scheme leverages the locality obtained from interactions between a user and objects during user's navigation in a virtual world, so called "user-based" data management. We assume that the behavioral pattern of a user is explained using the user's interest in objects in the world. To incorporate the user's interest into the access priority of objects, we leverage the fact that a user tends to repeatedly visit objects interesting to it or highly popular objects likely attracting it. To enumerate the level of interest and popularity of an object, we introduce two values, interest score and popularity score of an object, respectively. The interest score of an object is set per user and represents how much the user expresses its interest to the object. The popularity score of an object is set per world and represents how many people in the world express their interest to the object. By combining these two values with the spatial relationship, we improve the performance of caching and prefetching since the interaction locality between the user and objects are reflected in addition to spatial locality. Interest and popularity scores are determined by the number of access times for a given object. For further improvement of cache hit rate, we incorporate user's navigation behavior into the spatial relationship between a user and the objects in the cache. We observe that a user usually alternates a navigation mode between wandering and moving. An object residing at user's moving direction should have the same priority as an object residing at the opposite side of the user's moving direction in case of a wandering mode since it implies the possibility of rapid rotation. Apparently the former object should have higher priority than the latter object in case of a moving mode. This provides more

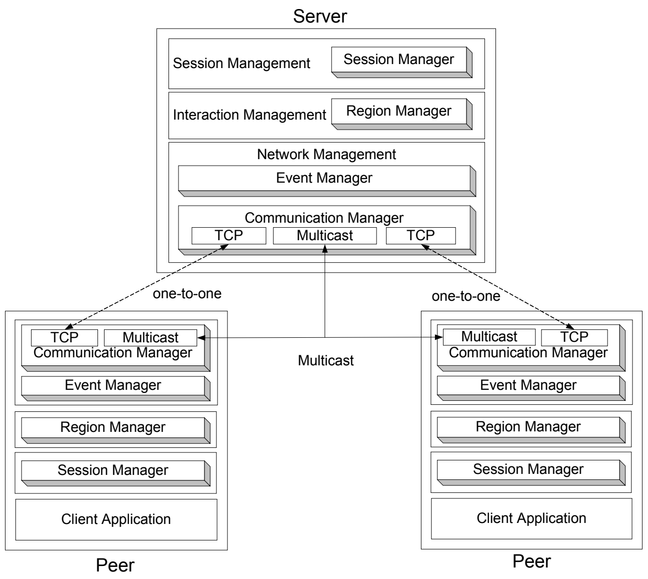

Figure 1. ATLAS architecture.

accurate information for caching and prefetching.

3.2 System Architecture

ATLAS provides modules for server and peer parts as illustrated in Figure 1. Each part consists of several major modules such as *communication manager*, *event manager*, *session manager*, and *region manager*. The *communication manager* is used for communication channel management. With selective communication types, ATLAS can be organized as the client/server model or the peer/server model. The *event manager* mediates messages between local events sent to remote peer and remote events processed by proper manager modules. The *session manager* manages membership in a session and the *region managers* in it. The *region manager* has a role of keeping a consistent state in a logical region.

3.2.1 ATLAS Events

Peers or servers communicate with each other via ATLAS events. There are six types of an event: *session event*, *region event*, *data event*, *interest event*, *concurrency event*, and *dummy event*. Each event type has its own purpose. For instance, a *session event* is defined for session-related events such as user-login and logout. The type of an event also implies which modules will handle the event. For instance, a *session event* is processed in the *session manager*. Each event type has several different event IDs depending on the purpose of the event. For instance, *interest event* has more than ten event IDs such as *USER_MOVED*, *USER_LEAVED*, etc. Each event class provides the interfaces for marshalling or unmarshalling the message. The *event manager* dispatches events to a proper handler. The detailed information on each event type is described in [15].

3.2.2 Communication Manager

The *communication manager* is responsible for creation and deletion of communication channels, and message transmission. For channel management, the manager owns a channel list which holds channels currently being open. It supports *polling* and *threaded socket group* to receive incoming messages. To send outgoing messages, the *communication manager* provides several communication methods such as unicast, multicast, and broadcast. Since we abstract a raw socket interface to facilitate easy communication channel management, it is possible to extend ATLAS basic socket classes for providing various communication protocols such as reliable multicast. In addition, the *communication manager* strictly handles communication exceptions to provide robust and stable communication channels.

3.2.3 Event Manager

The *event manager* mediates events between the *communication manager* and high-level components, such as the *session manager*, the *region manager*, the *interest manager*, the *concurrency manager* and the *data distribution manager*. It extracts the event information from the message delivered by the *communication manager*. It then checks the type and event ID field of the event to find an appropriate handler. The event is delivered to the chosen handler which has been registered to the event manager to handle the event. The structure of the *event manager* is illustrated in

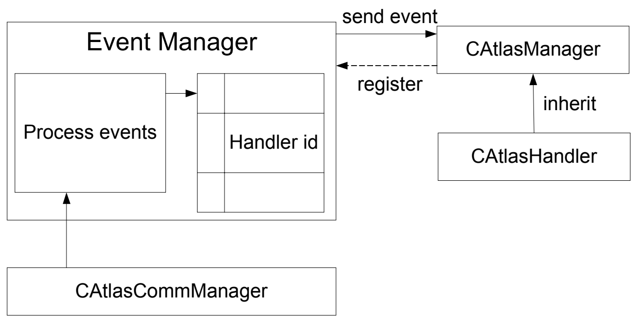

Figure 2. Structure of the event manager.

Figure 2.

All the ATLAS managers inherit *CAtlasHandler* class to receive and process events from the *event manager* or other components. A manager that wishes to receive specific type of events must register its reference to the components from which the manager is to receive events. The *event manager* has another role of meditating outgoing event to other remote peers or server. High-level managers such as the *session* and *region managers* send their events via the *event manager*. Then it selects channels corresponding to the destinations of the events and passes them to the *communication manager*.

3.2.4 Session Manager

A session in ATLAS implies a basic unit which is an independent virtual world. The *session manager* provides users with the interfaces for entering or leaving its virtual world and membership management, and defines specific rules applied to the session. Since we assume that a virtual world is divided into several logical regions, the *session manager* holds the references to the *region managers* that manage logical regions. It inherits *CAtlasHandler* class in order to handle incoming events and to send session specific events to remote peers or server.

ATLAS also supports multiple sessions. For dynamic management of sessions, the *multi-session manager* holding reference list of *session managers* provides users with the interfaces for initiation, termination, selection, join, leave, creation and deletion of sessions. To enter a virtual world, users query the *multi-session manager* about information on available sessions, and select a session among several sessions maintained by the ATLAS server. Receiving a login request from a user, the *session manager* verifies the user with his/her name and password. It then informs the user of regions in it, among which the user enters a default region. After receiving information on other users and objects in the region, the user can interact with them.

3.2.5 Region Manager

The *region manager* plays a major role of keeping a consistent state or view among users who participate in a virtual world. For this, it keeps track of all states information including dynamic

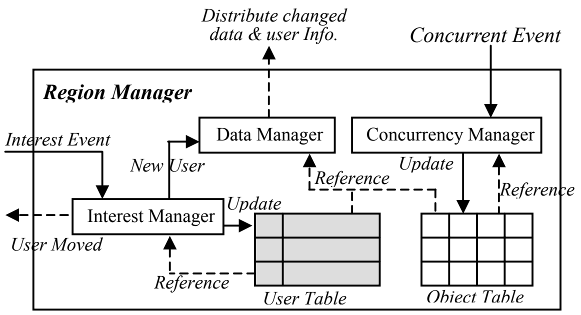

Figure 3. Structure of the region manager.

objects and users in a region or partial information of neighboring regions. Figure 3 illustrates a basic structure and event flows in the *region manager*.

The *region manager* has modules each of which contributes to the scalability described in Section 3.1: *concurrency manager*, *interest manager*, and *data manager*.

The *concurrency manager* resolves conflicts of concurrent access to an object in a region. It provides users with the entity-centric prediction based scheme [25] and the optimistic scheme as well as the simple locking based scheme. Whenever an object moves or a user wishes to manipulate an object, the *concurrency manager* can adopt the registered scheme as its concurrency control mechanism.

The *interest manager* manages users' interest in other users and objects and keeps track of information on the users who participate in a region and its neighboring regions. Whenever a user's state changes, the *interest manager* updates the user table. It provides several filtering mechanisms including not only the basic region-based filtering but also the inter-region interaction scheme [16] and the user interest group-based filtering scheme [10]. Whenever a user moves in a region, the *interest manager* selects an appropriate destination according to the specific interest management scheme, which prevents unnecessary event from being propagated to other users who are not interested in the event.

When a user joins a region, the *data manager* in the server sends him the region state information including user and dynamic object states to make the system consistent. It then notifies other users in the region of the new user. The *data manager* incorporates the mechanism described in 3.1.4.

4. INTEGRATION EXPERIENCES

In this section, we describe our experiences that we have obtained from the integration of ATLAS with various applications. ATLAS is designed to provide a network framework for DVE applications. While ATLAS provides no graphics and UI support, it does a veneer layer which intermediates between ATLAS and applications based on various graphics and UI models. The veneer layer mainly performs transformation between application events and ATLAS events. It also provides applications with a set of interfaces for interaction with ATLAS.

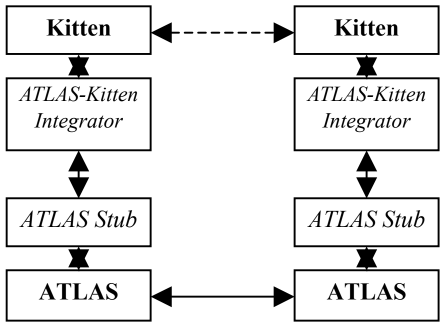

Figure 4. Integration architecture of ATLAS with Kitten.

ATLAS is implemented in C++ and Java. It has a set of classes, and the detailed implementation of the classes is described in [15].

4.1 ATLAS + Kitten

Kitten [12], developed by VR lab at KAIST, is a graphic library to implement a single user virtual reality system. It consists of a rendering module, a simulator module, and a user interface module each of which runs as an independent thread. We integrated ATLAS with Kitten for a collaborative engineering application. Integration is done with help of the intermediate veneer layer composed of *ATLAS-Kitten Integrator* and *ATLAS Stub*. Figure 4 shows the integration architecture.

The application is designed to enable users to visualize large CAD data and manipulate it for discussion on design decisions. It is built on the peer/server model and implemented in C++. A server performs user membership management and conflict resolution. For conflict resolution, the system adopts a simple lock based concurrency control scheme that is suitable for detailed and correct manipulation of CAD visualization data. Figure 5 is a capture image of the application.

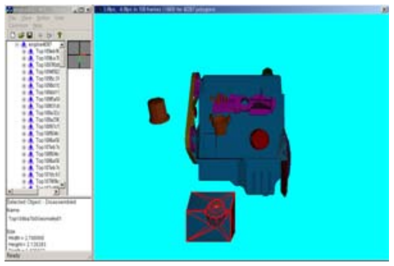

Figure 5. Collaborative engineering application.

4.2 ATLAS + Virtual Playground

Virtual Playground (VP) [22], developed by Human Interface Technology (HIT) laboratory at University of Washington, provides a shared virtual world that demonstrates how people learn, perform cooperative work, and engage in entertaining activity within 3D distributed virtual environments. It originally uses its own network modules which just provide primitive network functionality without any specific network management or filtering schemes.

We integrated ATLAS with VP without modifying it. Instead, we just replace the network modules with ATLAS and place a veneer layer between the two. In the veneer layer, VP events and data types are converted to ATLAS events and vice versa. Figure 6 shows the overall integration architecture.

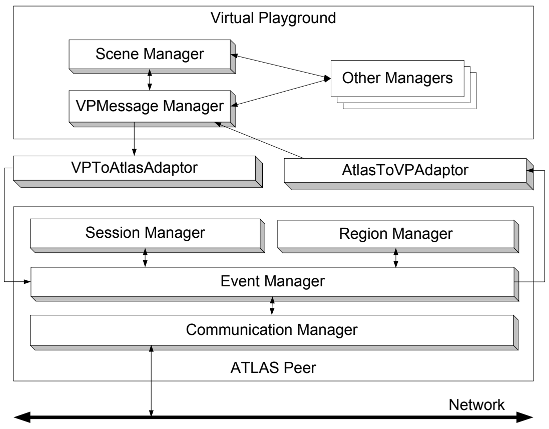

Figure 6. Integration architecture of ATLAS with Virtual Playground.

The veneer layer is composed of two lightweight modules. *VPToAtlasAdapter* plays a role of converting or mapping VP events, which need to share among participants, to ATLAS events. The mapped events are processed and filtered in ATLAS modules based on the event types. The filtered ones are then sent to the destination. *AtlasToVPAdapter* performs the opposite operations of *VPToAtlasAdapter*.

VP is written in Java. We use a Java version of ATLAS to integrate the two systems and the client/server model is used as a communication architecture. The integrated system performs region-based filtering to reduce communication overhead. VP uses a portal concept to allow a user to move to other place. We assign a logical region to a specific place which is separated by the portal. Figure 7 shows a user navigating in the integrated system.

Figure 7. Virtual Playground on ATLAS.

4.3 ATLAS + X3D Browser

ETRI/ICU Virtual shopping mall [11] (web site: http://cds.icu.ac.kr/VS/index.htm) is a web-based application which uses ATLAS as a communication infrastructure and RTV X3D viewer [20] as a browser.

Based on the client-server model, the application consists of an ATLAS server, a Web server, and clients. The ATLAS server manages not only a consistent state of a virtual world, but also concurrency control among users. The web server performs distribution of static virtual world data, avatar information, product information, and world rule information which describes dynamic objects in a virtual world.

A client, written in Java, is composed of an ATLAS client module and a RTV X3D browser. Communication and event passing between an ATLAS client and an X3D browser is done using EAI [4] and the extended interfaces of the browser. Figure 8 shows the architecture of the web-based virtual shopping mall system.

To enter a virtual shopping mall, users first connect to the web server, and download an applet and a X3D browser to set up a user interface. As a user selects a session and an avatar type which he/she is interested in, static world data and world rule information are downloaded from the web server in order to initialize the virtual world. Using the world rule information, an ATLAS client initializes dynamic objects and their properties obtained through EAI. To support late comers, the client downloads the current state information of dynamic objects from the ATLAS server and updates the virtual world.

To support scalability, the application applies the user interest group based filtering scheme [10] and the inter-region interaction scheme [16] as interest management described in Section 3.

For real time interactive concurrency control, we use the optimistic scheme rather than the pessimistic one. Figure 9 shows a client view of the virtual shopping mall.

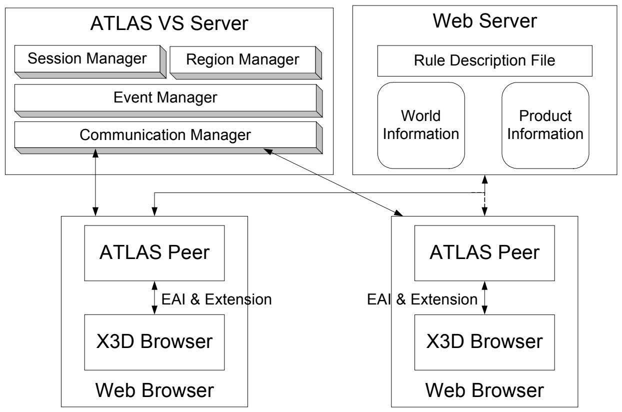

Figure 8. System architecture of ETRI/ICU virtual shopping mall.

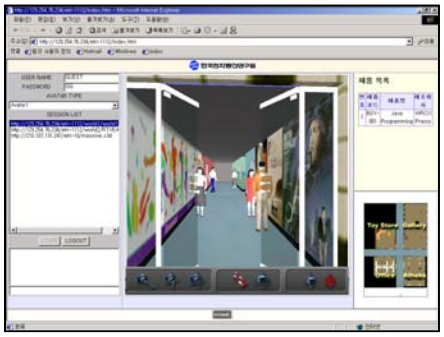

Figure 9. ETRI/ICU virtual shopping mall.

5. CONCLUSION

Scalability is one of important design issues in DVEs as users are geographically distributed and their number increases. In this paper, we have analyzed the scalability in terms of communication architecture, interest management, concurrent control, and data replication. We have proposed a network framework, ATLAS that supports scalable solutions based on our previous work in these four aspects. ATLAS provides a set of APIs in Java and C++ which suits various requirements of applications. To meet various application requirements, we support a peer/server model as well as a client/server model. To improve scalability, ATLAS allows a user to receive update messages from only others in whom he is interested instead of all users in the same and neighboring regions, which enhance the interactive performance. To resolve conflicts of concurrent updates of objects and grant an ownership to a right user in time, ATLAS provides a prediction-based concurrency control scheme in which the current owner of an object receives ownership requests only from users adjacent to the object not from all users in the same region. When replicating objects from server at local hosts, ATLAS reduces downloading time by caching and prefetching only objects in which users are interested in terms of proximity between users and objects and access priority of objects based on user's behavior pattern. Successful integrations of ATLAS with several applications ensure its versatility.

As the requirements of the users are diverse, it may require a DVE system to be dynamically extended or adapted to new services during runtime. We currently work on extending ATLAS for this, including resource discovery, resource monitoring, and dynamic world partitioning.

6. ACKNOWLEDGMENTS

We would like to thank members of CDSN laboratory for their work and comments during development of the system. We would also like to thank K Wohn and HS Kim at KAIST, Tom Furness and Bruce Campbell at HIT Lab, and BT Choi and HD Kim at ETRI for their help with this work. This work was supported by Virtual Reality Research Center of Korea Science and Engineering Foundation and the National Research Laboratory Program funded by Ministry of Science and Technology, the Republic of Korea.

7. REFERENCES

[1] Barrus, J., Waters, R. and Anderson, D. Locales: Supporting Large Multiuser Virtual Environments. IEEE Computer Graphics and Applications, November 1996, 16(6):50-57.

[2] Benford, S. and Greenhalgh, C. Introducing Third Party Objects into the Spatial Model of Interaction. European Conference on Computer Supported Cooperative Work, Lancaster, September 1997.

[3] Capps, M. The QUICK Framework for Task-Specific Asset Prioritization in Distributed Virtual Environments. IEEE Virtual Reality, March 2000, 143-150.

[4] Carey, R., Bell, G. and Marrin, C. ISO/IEC 14772-2:2001, The Virtual Reality Modeling Language (VRML) - Part 2: External authoring interface (EAI) see at http://www.web3d.org/fs_specifications.htm.

[5] Chim, J., Lau, R., Si, A., Leong, H., To, D., Green, M. and Lam, M. Multi-Resolution Model Transmission in Distributed Virtual Environment. ACM Symposium on Virtual Reality Software and Technology, November 1998, 25-34.

[6] Frécon, E., Greenhalgh, C. and Stenius, M. The DiveBone-An Application-Level Network Architecture for Internet-Based CVEs. ACM Symposium on Virtual Reality Software and Technology, University College London UK, December 1999.

[7] Funkhouser, T. RING: A Client-Server System for Multi-User Virtual Environments. ACM SIGGRAPH Symposium on Interactive 3D Graphics, Monterey California USA, April 1995, 85-92.

[8] Greenhalgh, C. and Benford, S. Boundaries, Awareness and Interaction in Collaborative Virtual Environments. IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Cambridge Massachusetts USA, June 1997, 193-198.

[9] Hagsand, O. Interactive Multiuser VEs in the DIVE system. IEEE Multimedia, Spring 1996, 30-39.

[10] Han, S., Lim, M. and Lee, D. Scalable Interest Management Using Interest Group based Filtering for Large Networked Virtual Environments. ACM Symposium on Virtual Reality Software and Technology, Korea, October 2000, 103-108.

[11] Han, S., Lim, M., Lee, E. and Lee, D. A Scalable Network Support for Internet-based 3D Virtual Shopping Mall. HCI'02, Phoenix Park, Korea, February 2002.

[12] Kitten for PC. http://vr.kaist.ac.kr/project1.htm.

[13] Lee, D., Yang, J. and Hyun, S.J. Scalable Predictive Concurrency Control for Large Distributed Virtual Environments with Densely Populated Objects. ACM Symposium on Virtual Reality Software and Technology, Korea, October 2000, 109-114.

[14] Lee, E., Lee. D., Han, S. and Hyun, S.J. Prediction-based Concurrency Control for A Large Scale Networked Virtual Environment Supporting Various Navigation Speeds. ACM Symposium on Virtual Reality Software and Technology, Canada, November 2001, 227-232.

[15] Lim, M., Han, S., and Lee, D., ATLAS – Internal Specification. Project Report, 2001. http://cds.icu.ac.kr/research/area/dve.

[16] Lim, M. and Lee, D. Improving Scalability Using Sub-Regions in Distributed Virtual Environments. International Conference on Artificial Reality and Telexistence, Tokyo Japan, December 1999, 179-184.

[17] Macedonia. M. and Zyda, M. A Taxonomy for Networked Virtual Environments. IEEE Multimedia, January-March 1997, 4(1):48-56.

[18] Macedonia, M., Zyda, M., Pratt, D., Brutzman, D. and Barham, P. Exploiting Reality with Multicast Groups. IEEE Computer Graphics and Applications, September 1995, 38-45.

[19] Park, S., Lee, D., Lim, M. and Yu, C. Scalable Data Management Using User-Based Caching and Prefetching in Distributed Virtual Environments. ACM Symposium on Virtual Reality Software and Technology, Canada, November 2001, 221-226.

[20] Real Time Visual Company. http://www.realtimevisual.com.

[21] Roberts, D. and Sharkey, P. Maximising Concurrency and Scalability in a Consistent, Causal, Distributed Virtual Reality System, Whilst Minimising the Effect of Network Delays. IEEE Workshops on Enabling Technology: Infrastructure for Collaborative Enterprise, 1997, 161-166.

[22] Schwartz, P., Bricker, L., Campbell, B., Furness, T., Inkpen, K., Matheson, L., Nakamura, N., Shen, L., Tanney, S. and Yeh, S. Virtual Playground: Architectures for a Shared Virtual World. ACM Symposium on Virtual Reality Software and Technology, Taipei Taiwan, November 1998, 43-50.

[23] Singhal, S. and Zyda, M. Networked Virtual Environments Design and Implementation. Addison Wesley, July 1999.

[24] Watsen, K. and Zyda, M. Bamboo-A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments. Virtual Reality Annual International Symposium, Atlanta Georgia, 1998, 252-259.

[25] Yang, J. and Lee, D. Scalable Prediction Based Concurrency Control for Distributed Virtual Environments. IEEE Virtual Reality 2000, New Brunswick NJ USA, March 2000, 151-158.