MOVE: Component Groupware Foundations for Collaborative Virtual Environments

Pedro García, Oriol Montalà, Carles Pairot and Robert Rallo
Department of Computer Science,
University Rovira i Virgili
Email: pgarcia@etse.urv.es, {omp.ei, cpg.ei}@estudiants.urv.es

Antonio Gómez Skarmeta
Department of Computer Engineering,
University of Murcia
Email: skarmeta@fcu.um.es

ABSTRACT

The design of a Virtual Environment (VE) is a distributed problem of multi-user access to shared resources. Such problem requires careful design decisions in order to provide a seamless system infrastructure capable of supporting flexible interactions in the shared scenarios.

The complexity of this domain has led to intricate software systems that provide ad-hoc solutions to specific problems. Furthermore, many of them have gone to a dead end, due to their non-extensible design and their lack of code and module reuse.

This paper presents a VE that is constructed on top of a component groupware framework. Our major aim is to provide an extensible infrastructure offering a set of collaborative services in a seamless way. At the conceptual level, it provides essential collaborative services: shared sessions, support for synchronous and asynchronous components, security, coordination, and a server-side awareness infrastructure. At the architectural level, the framework is constructed on top of a middleware integration platform and uses high performance publish/subscribe notification services. Finally, we present the advantages and limitations of this approach.

Categories and Subject Descriptors
D.2.1.1 [**Software Engineering**]: Software Architectures – domain specific architectures.

General Terms
Design, Experimentation.

Keywords
Virtual Environments, Frameworks, Component Groupware, Distributed Systems.

1. INTRODUCTION

In the last years we have been experiencing advances in networking technologies and protocols as well as database, computer graphics and display technology. As a consequence, a lot of Collaborative Virtual Environments have emerged thanks to the increasing computation capabilities of desktop computers as well as the enormous growth in network bandwidth and the ubiquity of the Internet.

In this line, many research efforts have tried to solve the distributed problems inherent to the design of VEs. Relevant examples of these VEs are DIVE [3], MASSIVE [5], NPSNET [10] or SPLINE [9]. Many of them have successfully addressed problems like world structuring and zone partitioning, state propagation, coordination and consistency, and interest management and awareness.

Nevertheless, the complexity of the domain has led to huge intricate software systems that are difficult to extend and reuse, and thus precluding system interoperability and flexibility. To cope with these problems, the trend is to augment modularization in the design of such infrastructures.

JADE [11] contributes to augment design modularization proposing a generic component framework that outlines system interoperability. However, this proposal specifies a minimal kernel focused on inter-communication capabilities that do not offer collaborative services like session and zone management, state propagation, coordination and consistency, and awareness.

Another interesting approach is the NSPNET-V move towards component architectures. NSPNET Bamboo [16] is a component system where code modules operate in a cross-platform and cross-language manner. In NSPNET-V, Java components can be dynamically loaded at runtime and XML is used as a message interchange format.

We also outline as a key problem in the design of VEs the lack of awareness services enabling the extraction of information from the running infrastructure in order to better understand the group interactions by means of indicators or data-analysis tools. These awareness services can improve the shared experience and enable situational reactions to specific events in the multi-user environment.

In this line, the TOWER [15] project is developing an infrastructure to provide awareness by analysing events from groupware systems and visualizing them using Blaxxun technologies. On the other hand, MASSIVE-3 [5] includes Record and Replay mechanisms that permit to recover and reconstruct collaboration flows in the shared scenarios. Our approach goes further; we consider it is very important to design VEs considering those awareness services that enable situational reactions to events in the environment.

To finish with, we also outline two declarative component approaches like CONTIGRA [2] and Jamal [14]. Both approaches define XML-based mark-up languages for 3D component description and assembly. With well-defined component interfaces and descriptors it is possible to automate code-generation and benefit from dynamic introspection mechanisms.

In order to solve the problems mentioned above, we construct our MOVE environment on top of a component groupware framework called ANTS. This framework offers key collaborative services to components like session management, state propagation, security, coordination, and a server-side awareness infrastructure. Furthermore, the framework is constructed on top of a middleware integration platform and uses high performance publish/subscribe notification services. Software facades permit us to change the underlying middleware services without affecting the upper layers.

The paper is structured as follows: Section 2 overviews the MOVE environment and design objectives stated for this system. Section 3 describes the ANTS framework and presents the collaborative services offered to components. Section 4 explains in detail the MOVE architecture specifying how it benefits from the underlying framework. To finish with, in section 5 we draw conclusions of our current work and present the advantages and limitations of this approach.

2. MOVE OVERVIEW

MOVE is a 3D collaborative environment where users (Avatars) can interact with other users or with shared artefacts like voting tools, presenters, 3D simulations, documents or multimedia contents.

MOVE is one of the key applications being tested in the Catalonian Internet2 project and it is mainly deemed for educational purposes in different scenarios like virtual classroom simulations, archaeological or medical applications.

Its architecture benefits from ANTS Computer Supported Cooperative Work (CSCW) Framework. In addition, MOVE has been developed using open technologies such as VRML or the H-Anim v1.1 [6] avatar standard specification. The development language chosen has been Java and the bridge used to communicate this language with VRML has been the External Authoring Interface [17]. The use of open specifications permits us to work with any VRML2 compliant plugin like Blaxxun or Cosmo and to select any H-ANIM avatar for user representation in the shared scene.

2.1 Problems in Existing 3D CVEs

Existing 3D Collaborative Virtual Environments (CVE) presents certain problems that from our point of view have not yet been completely resolved:

- The first problem is the high complexity in developing such a system of this kind. This type of environments must be easily scalable and should support as many concurrent users as possible degrading as least as possible system's performance. Existing 3D CVEs have dedicated many efforts in developing the environment itself rather than making it more scalable or making it component oriented so as new components may be added in the future. This is what we call component reusability.
- Another problem is data extraction. These CVEs were not thought for retrieving data in an easy way. Sometimes it can be interesting to grab data about what a specific user has been doing in our environment for data-mining, logging or capture and replaying purposes.

The solutions that MOVE adopts to solve the problems mentioned above are:

- Define a component model based on the standard JavaBeans specification that hides complexity to users providing transparently remote persistence, distributed events and component descriptors in XML and packaging. By adopting this component model we can create new tools that can be easily inserted into our system by means of "importing" them.
- Define a dedicated awareness and event monitoring service together with an agent system which reacts to the events triggered in the bus. By adopting this method, any kind of information can be obtained, such as logs, application of data-mining algorithms or usage of capture and replay techniques.

To achieve all these objectives, MOVE uses ANTS CSCW component framework. This infrastructure facilitates development of collaborative components, and thus achieving a smooth transition from local to distributed applications.

3. ANTS FRAMEWORK

The ANTS system aims to provide a generic multi-user collaborative framework. Being the overall problem a complex distributed challenge, it is necessary to solid distributed technologies as a basis as well as to provide generic collaborative services to the user and developer communities. As we can see in Figure 1, the collaboration bus is a key piece of the overall architecture.

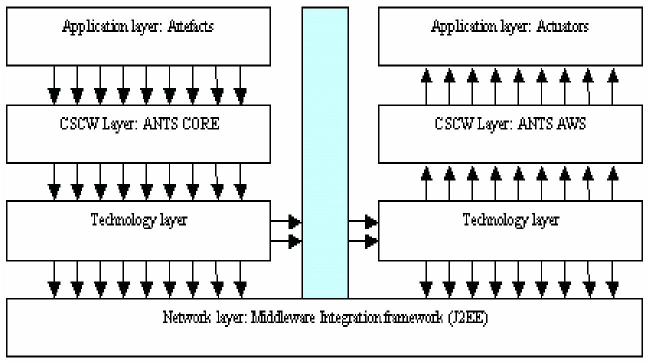

Application layer: Artefacts
Application layer: Actuators
CSCW Layer: ANTS CORE
CSCW Layer: ANTS AWS
Technology layer
Technology layer
Network layer: Middleware Integration framework (J2EE)

Figure 1. ANTS Architecture.

On the one hand, all components trigger events to this distributed information bus, and on the other hand, awareness components listen to the bus for information about what is happening in the system.

While other systems like Lotus PlaceHolder or CORONA used publish/subscribe middleware to propagate state changes, and NESSIE or Khronika used it to create awareness systems, none of them created an integrated framework where the state changes propagated to the bus can be easily captured by an integrated awareness system.

In this line, we can consider the collaboration bus not only an event dispatcher, but a higher level abstraction constructed on top of publish/subscribe middleware that transparently to developers creates a state propagation system for shared components and a mediator system in which third-party actuators can react to concrete events in the environment.

We can distinguish three main layers in the overall architecture: a technology layer, a CSCW service layer, and an application layer. Upper layers represent high level abstractions that hide complexity to users from the layers below.

In this line, the application layer is aimed to end users that extend the framework hotspots to create collaborative components and applications or monitoring agents. The CSCW layer comprises the set of collaborative services that the framework offers to application level components. Furthermore, these CSCW services are built on top of distributed services offered by the underlying technology layer. Let us show their main functionalities:

- At the application layer, we have created a component model which abstracts access to a remote property-based persistent component, and to distributed events produced in the current session context.
- The CSCW layer comprises the set of collaborative services that the ANTS framework offers to the application layer in a coherent way. It thus provides session support, component's life cycle, communication abstraction, coordination support, and security. It also includes a seamless server-side platform for awareness actuators.
- At the technology layer, we have created software components that transparently facade advanced middleware services and network components. Our approach permits us to select from a variety of middleware vendors in database systems, application servers and notification services.

3.1 Application Layer

At the application layer, the framework provides two extension hooks: development of new collaborative components and awareness actuators reacting to information events produced in the framework. We, however, study awareness actuators in the CSCW layer as part of the overall awareness model.

Our major aim is to facilitate development of collaborative components, and thus achieving a smooth transition from local to distributed applications. Obviously, our approach is to be based on existing specifications already adopted in the developer community.

Like many other groupware toolkits [7], we have decided to use the Java language and the Java Bean component specification as our preferred development technologies. In this line, our approach aims to sustain Java Bean components in a container that hides complexity in distributed programming and networking skills. The general idea is to provide a property-based persistence system, and to construct a Java distributed event service. Components are also packaged in JAR files, and they use XML to provide component descriptor.

3.2 Component Model

The component model comprises persistence, events, introspection, packaging, and component deployment in the ANTS component repository:

Properties are shared data structures stored in the ANTS server. Each Java Bean component can use three types of <u>properties</u>: string properties, string indexed properties, and Serializable object properties. These properties are persistently stored in a database by the remote property-based component and thus assure component <u>persistence</u>. We believe that string properties and indexed properties are a strong base for storing textual or XML strings containing persistent state. Yet, we supply serializable object properties enabling storing of more complex java objects.

Following the Java Bean bound properties approach, each property change will trigger an event (in this case, to the <u>distributed event service</u>) allowing a later subscription to PropertyChangeEvents. Completely coherent with the Java Bean approach, we also provide addPropertyChangeListener methods to receive property change events in bound properties. In our case, and transparently to developers, property change events are received from the distributed notification service and restricted to events in the current session and component.

Another key issue of the ANTS framework is to support <u>coordination</u> in a generic way allowing coordination rules for access control, concurrency control, and floor control. Our approach is also inspired in the JavaBean model, and particularly in the constrained properties.

Each method can trigger a PropertyVetoException like in the constrained property convention. We, however, differ in the source of the VetoException; in the JavaBean model, vetoes are thrown by registered VetoableChangeListeners; in our model, vetoes are thrown by registered coordination managers. Coordination managers interpose in calls to a component to assure validity of the requested action

Concerning <u>customization and introspection</u> of components in the ANTS framework, we provide an XML descriptor file. In general, a descriptor includes properties and event information. Event information will be the key for the awareness infrastructure in order to know which events are triggered by each resource in the shared environment.

Finally, component <u>packaging</u> uses standard JAR files like in the Java Bean component model. The package must contain the XML descriptor, component classes, and any local resource –images, files– used in the component.

Packaged components can be deployed in the ANTS server following an automated process. In this process, components are registered in the component repository using the XML descriptor, and unpacked resources are copied to the Web container.

The repository is an important element of the overall framework that permits to get information about registered components, and allows insertion and removal of artefacts in shared session contexts.

3.2.1 CSCW Layer

The CSCW layer comprises the set of collaborative services that the ANTS container offers to the application layer. We will, however, distinguish two main modules, namely container runtime (ANTS.CORE) and awareness services (ANTS.AWS), seamlessly interconnected by the collaboration bus.

ANTS.CORE comprises a generic and extensible application layer for collaborative tools, providing services like shared sessions, coordination control, and integration for both synchronous and asynchronous applications. ANTS.AWS is a generic awareness service providing appropriate actuators for events received from the collaboration bus.

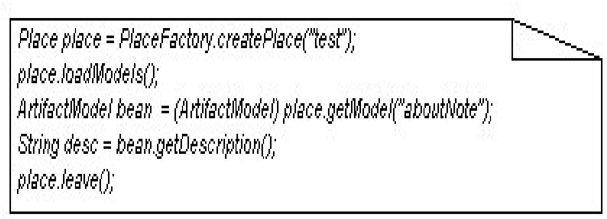
3.2.1.1 ANTS.CORE

ANTS.CORE is an essential module in the ANTS framework that includes explicit support for Sessions, shared artefacts, coordination control, security, and a seamless security model.

We have been strongly influenced by Multi-user Object Oriented (MOO) architectures because of their extensible and generic architecture. In a MOO system, "place" objects, which represent discrete locations interconnected by portal objects, define the topology of the space. These portal objects can establish any graph-directed relationship between "places". In a MOO architecture every static or dynamic entity is an object (Thing), every object has a set of properties and a set of verbs, and properties and verbs can be added or removed in runtime.

Like a MOO system, our shared session is represented by the Place object, interconnected by portal objects (Links). The shared artefacts are represented by Java Bean components dynamically loaded by the ANTS container. Object properties are represented in our model by the remote persistent property-based component, and object verbs are Java Bean classes.

The Place represents the shared session containing users, components, and links to other places. It provides methods in order to send or subscribe to events in the current context. It also supplies methods to get connected users and available links to other places. Furthermore, it permits dynamic load of Java Bean components to the shared context. Let us study the component's life cycle (Figure 2):



```
Place place = PlaceFactory.createPlace("test");
place.loadModels();
ArtifactModel bean  = (ArtifactModel) place.getModel("aboutNote");
String desc = bean.getDescription();
place.leave();
```

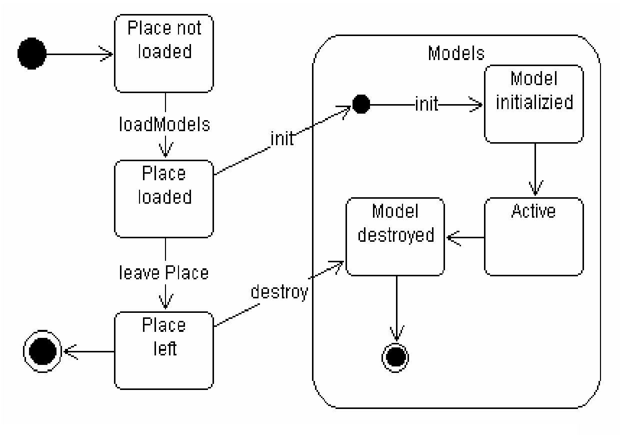


Figure 2. Component's life cycle.

When the *loadModels* method is invoked in a Place, all Java Bean components are activated (calling the *init* method) and classes are dynamically loaded by the ANTSClassLoader. The ANTSClassLoader retrieves classes from the public web directory using the HTTP protocol. Finally, when the user leaves the shared session, the *destroy* method is invoked in all JavaBean components currently active in the container.

3.2.1.2 ANTS.AWS

ANTS.AWS is a server-side awareness infrastructure enabling the triggering of a set of actuators in response to events produced in the collaboration stream.

ANTS.AWS is consistently integrated with the component model through XML descriptors included in packaged components. As such, every descriptor contains the events that this component triggers to the bus. Because this information is stored in the repository during the deployment phase, external monitoring application or AWS actuators can be aware of the information produced by every component in the platform.

This service creates the base for sophisticated monitoring applications, intelligent agents, and persistent actuators activated by events received from the Notification system. Based on existing CSCW Awareness systems [12], AWS includes an event repository, a notification service, and an infrastructure to filter or get information from events in the distributed bus.

AWS follows a Mediator design pattern with three important entities: Sensors, the Mediator, and actuators. Sensors represent any physical or software component producing events to the Notification system. Actuators perform special tasks on response to sensors; and the Mediator binds a sensor with one or more actuators, and it is responsible for launching actuators in response to events.

In relation to sensors, we consider two distinct types: event sensors and time sensors. Event sensors are activated in response to application events produced in the information bus. An event sensor is created using a subscription following the constraint filter language available in the Notification system.

On the other hand, time sensors are activated by the ANTS time scheduler; we can create a time sensor for a specified date or even set repetitive dates. The time scheduler notifies the awareness service when a time sensor has been activated.

In a possible scenario, we could create a subscription telling that we are interested in events produced by the user called "pedro" (event sensor), and that we want to make them persistent (actuator). This would log each event coming from user "pedro" occurred in the environment. We could also create our own bots, which would react to events specified by our subscription.

3.2.2 Technology Layer

First of all, such a problem requires a solid infrastructure providing security, scalability, persistence, transactions and performance. As we explained before, many existing systems rely on non-distributed, single-threaded environments that limit their applicability to more complex problems.

For this reason, we have chosen the Java 2 Enterprise Edition (J2EE) standard as our preferred technology infrastructure. It saves us from implementing infrastructure and system-specific code and permits us to base on open specifications and components. This technology makes our system vendor-independent: we can choose any J2EE-compliant application server or relational database.

Concerning message oriented middleware, we needed a publish/subscribe notification service for our framework. Being JMS (Java Message Service) a standard alternative, we were also strongly interested in the new Elvin notification service. This service has a very good performance and has been used for CSCW purposes. As a result, we implemented a facade API that permits us to choose between any JMS-compliant messaging solution and Elvin. Event filtering is then obviously achieved in our environment, directly accessing constraint filter languages available in these messaging systems.

In order to connect to a specific messaging middleware, developers only care about registered bus aliases in the framework. We provide several sample bus aliases to different middleware services. Depending on the selected middleware, we can choose different transport layers, such as, raw multicast, reliable multicast, encrypted Secure Socket Layer (SSL) streams, ordered streams, according to our specific requirements.

Non-expert programmers can simply select *SSL* as the communication channel (bus alias) and thus use an encrypted communication channel. Applications requiring scalability for a high number of users could then select the *RMCAST* alias.

4. MOVE ARCHITECTURE

In this section we will deal with MOVE's collaborative architecture. We will focus on the essential collaborative services provided, like session and zone management, component model and state propagation, coordination and consistency, and awareness.

4.1 Session and Zone Management

In order to determine the environment structure, the place concept is used. A place is a piece of virtual environment where users can interact with each other and use the different shared artefacts available. ANTS.CORE defines a session as a group of objects associated with some common communications pattern and supporting full-duplex multipoint communication among an arbitrary number of connected application entities. This is essentially what a place means in MOVE and any other MOO environments. By using this philosophy, we identify session with place.

In our case, the concept of "Places" is similar to the concept of "Locales" in SPLINE [5]: places are the fundamental unit of world composition, where each place represents a clearly separated region of the virtual world, such as a room or an open space, and can contain many virtual shared artefacts as well as avatars and portal objects.

As far as portal objects is concerned, we can say they are used for place linking purposes, thus providing a mechanism of transport between different places. By using this zone partitioning approach, we can define a session hierarchy where places can contain subplaces.

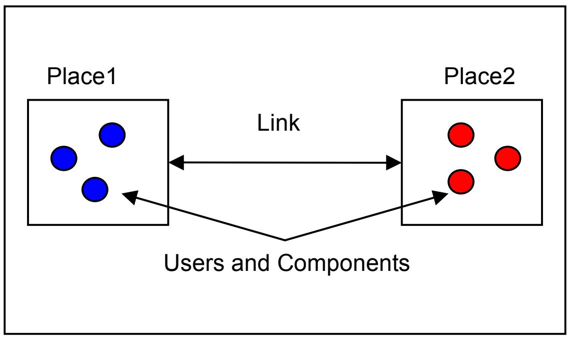


Figure 3. Zone Partitioning in MOVE.

4.2 Shared Artefacts and State Propagation

Another essential concept is the explicit support for **shared artefacts**. Normally, users engage in a collaborative interaction with other users in the shared context, or with accessible collaborative components. These collaborative components – usually called artefacts – range from synchronous to asynchronous, depending on the interaction time frame. Explicit support for creation and insertion of new artefacts represent a differentiating factor between extensible and non-extensible CSCW toolkits.

MOVE is based on a modular architecture for every component. There are different types of components available, such as voting, presenter, video, and document or multimedia tools. All of these components follow the Model-View-Controller (MVC) paradigm. Every client owns a local copy of the controller, which is the tool used to modify the state of the component, and the view, which is the representation of the component that is shown on the screen.

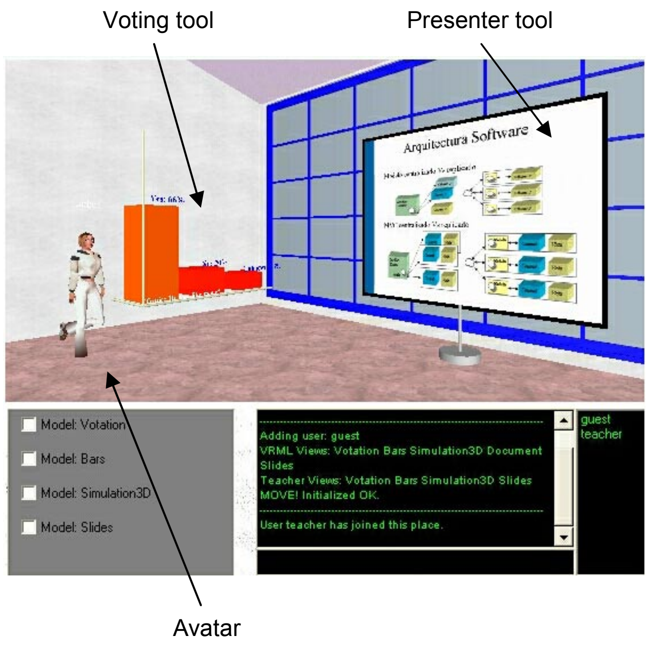
Our main aim is to design the system so that it is easily extensible and reusable. To do so, every component has an XML descriptor that contains all the necessary properties and information related to events that can be thrown, besides the component's classes and any local resources. This new component is packaged using the standard JAR format.

Packaged components are deployed in the ANTS server following an automated process where these components are registered in the component repository, using the XML descriptor, and unpacked resources are copied to the Web container. Using this mechanism makes integration of new components particularly easy because it is like "importing" the new component (Tool) into the system before it is ready to use.

One of the most difficult problems to solve when programming distributed systems is maintaining the state of the components among all other participants. In our case, this problem is minimized and transparent to us by using ANTS CSCW framework because it guarantees that the state will be propagated to every each user in the same place that we are in. This way, when developing new components for our system, this problem is solved and we can concentrate our development efforts on other aspects.

Another fact concerning state propagation is that persistence in our system is maintained, as the natural behaviour for every change in any property of the system is stored immediately to the database by means of propagating the event to the server. Other distributed virtual environments are not persistent in the sense that they are transient due to the fact that information is stored among all the clients. This may be an advantage sometimes but if all the clients go down at the same time the information may be lost.

We have developed a set of shared artefacts which include a voting tool, a presenter tool (both can be seen on Figure 4), a bars tool, a document tool, a jukebox tool and a video tool, as well as two tools for grabbing user's attention (the hook and camera tools). In these components, state propagation mechanisms provided by ANTS CSCW framework have been used.



```
------------------------------------------------------------
Adding user: guest
VRML Views: Votation Bars Simulation3D Document
Slides
Teacher Views: Votation Bars Simulation3D Slides
MOVE! Initialized OK.
------------------------------------------------------------
User teacher has joined this place.
```

Figure 4. MOVE snapshot showing the presenter and voter artefacts and an avatar.

However, there are certain components that cannot use the state propagation mechanisms that the framework provides us with. One of the examples we have found is the avatar movement component. It is clear that it is impossible to transmit every positional event generated by the movement of a certain user to other users in the same place. This would generate a huge amount of event overhead over the network that even clients would be unable to deal with. In this case, we have had to optimize this component by implementing proximity event filters. As it is known, one of the weak points of real-time 3D environments is the enormous processing time required to render the 3D scene. This is why when there are lots of users in the same place the cost of rendering them is unacceptable for the client. To solve this problem we have implemented a proximity event filtering algorithm so as clients only receive the events of the avatars that are in a certain threshold distance near them, as seen in Figure 5.

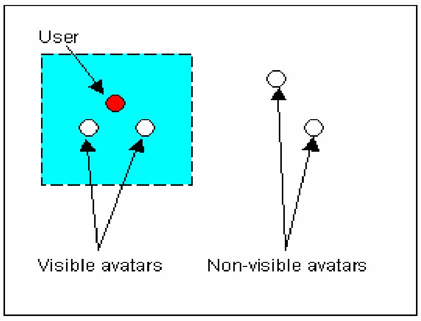


Figure 5. Only the avatars in user's threshold distance will be drawn.

Finally, note that a personal profile management configuration option has been added so that MOVE users can choose their own virtual representation in our environment (avatar), as well as many other properties that could be used in the future by an agent system.

4.3 Coordination and Consistency

Another key issue in any CSCW framework is the explicit software support for **coordination policies**. These policies can be set programmatically or declaratively, and can be categorized in roles for access control and concurrency control. The platform should provide extension hooks to insert custom coordination components that would then transparently interpose in normal calls to a collaborative component.

In our case, and considering that MOVE was initially designed as an educational collaborative virtual environment our access control policies have been established by means of defining basically two main roles: *teacher* and *student*. By using this approach, the teacher role members have total access to manipulating the state of the shared artefacts, whereas student role members can only view the result of these state changes made by teachers.

We are working in the development of a concurrency control mechanism by using a token-based lock component, to avoid more than one user manipulating any shared artefact. This mechanism will also allow users to transport artefacts from one place to another, thus assuring data consistency among sessions.

4.4 Awareness

Awareness is an important feature of collaborative environments. Awareness can be defined as an understanding of the activities of others, which provides a context of your own activity. From our perspective, an awareness platform should provide data acquisition from the running environment. In this line we emphasize the importance of a well-defined awareness model.

MOVE benefits from ANTS CSCW awareness and monitoring services. In fact, ANTS.AWS is the base for the MOVE agent infrastructure. In MOVE it is possible to program bots that perform special tasks and that react to events produced in the environment. As an example, we provide a guide bot that shows the whole place to new-comer users and interacts with the shared artefacts when a certain user moves besides it. More advanced agents can be developed and in fact we are using them pervasively to test our environment to simulate high user loads.

As seen in Figure 6, the bus is used for state propagation and for awareness purposes. On the one hand, each time the user moves it transmits its movements to the bus and they are received by the rest of the clients in the shared context (1 and 2). On the other hand, the monitoring service (*Mediator*) listens to the bus (3) and launches specified actuators in response to filtered events. In this case, the Mediator activates a guide bot that starts a sightseeing tour over the place. As we can see, this network view of the system complements Figure 1 in order to understand the overall framework architecture.

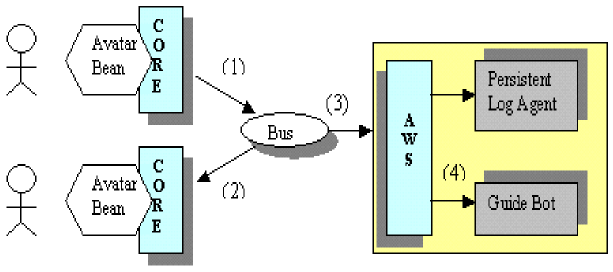


Figure 6. MOVE components and bots.

Right now we have simple bots or agents running, yet in the future these could be enhanced so as we could have advanced guiding bots help bots or even bots that could collaborate and interact with each other by using agent communication protocols.

Concerning logging and capture of events, the AWS mediator system enables us to activate custom actuators in response to specified events in the environment. With ANTS AWS we include a persistent actuator that logs events to a relational database. We also provide simple capture & replay actuators that permit re-using data captured in the environment. We have yet used these captured data in a data analysis system that employs machine learning algorithms like clustering.

4.5 Performance issues

As a result of applying massive tests to the system, we have found that the critical point resides in the client side and not in the ANTS platform or server -side middleware.

In the server-side there is very small memory consumption and processing power dedicated for every user connected to the system. Furthermore, the notification system can handle massive event flows without degradation and thus achieves enough scalability under high loads.

In the client-side, the 3D visualization engine requires big amounts of memory when it handles complex 3D simulations or avatars. Moreover, when a very high number of events arrive to the client, this has problems to transmit them to the visualization engine and thus can lead to system degradation.

In order to solve that problem, we have implemented a distance-based algorithm that discards events that are not located in a specified radius around each user. This strongly limits unnecessary processing in the client side and permits us to handle a large number of concurrent users.

Obviously, the main performance problem in MOVE resides in user movements that produce a high number of events. To further refine the platform we are also considering the implementation of dead-reckoning algorithms.

Nevertheless, although user simulations showed that the system scales up smoothly with 200 users in a shared session, the Internet 2 project usage scenario provides better hardware and bandwidth resources capable of sustaining a higher number of users.

5. CONCLUSIONS

MOVE is a Collaborative Virtual Environment constructed on top of a component groupware framework. The VE fully benefits from the underlying collaborative services provided by the ANTS infrastructure.

While the tendency for modularity is rippling through VE systems, we foresee interesting research in the development of component frameworks offering collaborative services in a seamless and extensible fashion to VE developers.

Our work aims to provide consistent groupware foundations to the development of VEs and thus enabling further reuse and interoperability in this complex domain.

One of the key aspects of this work is to provide extensibility at all levels of the framework. It is thus possible to create new components or artefacts, to develop new coordination mechanisms, to interface with different middleware services and propagation channels, and to develop new actuators for the awareness service. This creates an interesting Playfield for research in VEs by opening the system to third-party extensions willing to solve specific problems in this wide domain. Furthermore ANTS and MOVE are open source projects that welcome contributions and extensions by commercial, educational or research groups.

We are now developing new artefacts required by the educational communities in archaeological, medical and pedagogical applications involved in the Internet2 Catalonian project. The use of MOVE in a real setting will give us interesting feedback to create new awareness actuators and bots and to further improve interaction capabilities in the MOVE environment

As future works, we plan to further improve the coordination and consistency control mechanisms, to test different propagation channels and to apply data-analysis techniques to better understand collaboration flows in the running environment.

We also foresee interesting research in the awareness extensions to the MOVE Virtual environment. The provision of advanced awareness indicators and the extension of the agent system can lead to new interesting scenarios in VEs.

6. ACKNOWLEDGEMENTS

This project has been partially funded by the Generalitat de Catalunya' Internet2 Advanced Telecommunications project and European Union ITCOLE project.

7. REFERENCES

[1] Broll, W. Populating the Internet: Supporting Multiple Users and Shared Applications with VRML. ACM SIGGRAPH: Proceedings of the VRML '97 Symposium. New York, 1997, 87-94.

[2] Dachselt, R.; Hinz, M.; Meißner, K.. CONTIGRA: An XML-Based Architecture for Component-Oriented 3D Applications. Proceedings of the ACM Web3D 2002 Symposium, Tempe (USA), 24.-28.

[3] DIVE Homepage. http://www.sics.se/dive.

[4] Garcia P, Skarmeta A., Rallo R., ANTS: A new Collaborative Learning Framework, European Conference on Computer Supported Collaborative Learning. 2001.

[5] Greenhalgh, C., Purbrick, J and Snowdon, D. Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. Proceedings of the CVE 2000 Symposium, ACM, 2000, 119-127

[6] H-Anim Humanoid Working Group. http://www.h-anim.org

[7] Hofte, G. Henri ter. Working Apart Together: Foundations for Component Groupware. Telematica Instituut, Enschede, The Netherlands, 1997, 288.

[8] Marsic, I. DISCIPLE: A Framework for Multimodal Collaboration in Heterogeneous Environments. ACM Comp. Surveys, vol 31, no. 2es, 1999.

[9] Mitsubishi Electric Research Laboratories. Scalable Platform for Large Interactive Networked Environments (SPLINE). http://www.merl.com/projects/spline/.

[10] NPSNET Research Group. http://www.movesinstitute.org/~npsnet/index.html.

[11] Oliveira, M., Crowcroft, J. and Slater, M. Component Framework Infrastructure for Virtual Environments. Proceedings of the CVE 2000 Symposium. ACM, 2000, 139-146.

[12] Prinz, W. NESSIE: An Awareness Environment for Cooperative Settings. Proceedings of 6th European Conference on Computer Supported Cooperative Work. Kluwer Academic Publishers, 1999, 391-410.

[13] Roseman, M. and Greenberg, S. Building Real-time Groupware with Groupkit, a Groupware Toolkit. ACM Transactions on Computer-Human Interaction, 3(1):66—106, March 1996.

[14] Rudolph, M. Jamal:Components Frameworks and Extensibility. http://www.web3d.org/TaskGroups/x3d/lucidActual/jamal/JamalImpl.html

[15] Tower Project. http://tower.gmd.de.

[16] U.S. Department of Defense (DOD). Distributed Interactive Simulation, IEEE Standard 1278. 1993.

[17] VRML 97 External Authoring Interface. http://www.vrml.org/WorkingGroups/vrml-eai/Specification/

[18] Watsen, K.; Zyda, M. Bamboo, A Portable system for Dynamically Extensible, Real-Time, Networked Virtual Environments. Proceedings of the IEEE Virtual Reality Annual International Symposium (VRAIS'98), Atlanta, Georgia, 1998, pp 252-259.

[19] ANTS MOVE. http://ants.etse.urv.es/move/