Towards an Interaction Model in Collaborative Virtual Environments

Diego Martínez, José P. Molina, Arturo S. García, and Pascual González

Lab. of User Interfaces and Software Engineering (LoUISE),
Instituto de Investigación en Informática de Albacete,
University of Castilla-La Mancha, Albacete, Spain
{diegomp1982,jpmolina,arturo,pgonzalez}@dsi.uclm.es

Abstract. This paper highlights some of the current problems in CVE development. These problems are mainly due to a lack of a good interaction model guiding the behaviour of the environments. This paper introduces the definition of a model based on the idea of the *interaction views*. The key concepts for understanding *interaction views* are also given during the explanation of the model. The paper also describes a reference architecture based on this model which can be found useful for the design of modelling tools, and a prototype application that helps understanding both the architecture and the model.

Keywords: Virtual Reality, Interaction, Collaboration.

1 Introduction

Collaborative Virtual Environments (CVEs) have been a topic under intensive study in the last few years. The current status of the technology, which allows computers with a good graphical performance and VR devices at lower prices, has boosted this research. Many techniques and ideas have arisen, and researches have shown many of the best properties of CVEs, such as presence and object manipulation. Furthermore, many technologies and tools have emerged around these kinds of systems: techniques for managing groups, frameworks for building distributed environments and interconnecting them. All these tools together with the tools typical for VEs (OpenGL, VRML/X3D, etc.) should provide enough support to implement useful environments as this is the current interest of developers. In the last years, developers are no longer looking for environments which look and feel like real, but environments that offer the user an efficient way to achieve a goal.

However, when looking at the systems described in the literature [1,2,3,4,5,10], most of them can be categorized in two main families: commercial and experimental environments. The first family of these systems allows a high scalability but its members doesn't benefit from the best properties of CVEs. The second family of systems refers to the experimental systems used by researchers. They benefit quite well from the features found in CVEs, but focusing on just one feature that is,

R. Shumaker (Ed.): Virtual Reality, HCII 2007, LNCS 4563, pp. 310–319, 2007.

somehow, evaluated. In both cases, collaboration is faced exclusively from the point of view of information sharing. The more common techniques used are:

- **Collaboration Buses:** Many systems, like MOVE [7] or like MPEG´s back channels [1,6], use an event dispatcher connecting all the clients in the CVE and that, in addition, can be used to build a higher level of abstraction through the definition of communication protocols between objects.
- **Shared Scene Graph:** This solution is more appropriate for VR applications, and it is used in other platforms such as DIVE[3,4] or MASSIVE[1]. All the users share the same definition of the position and status of each object in the environment. The shared information is all contained in the scene and clients do not communicate with each other directly, but trough modifications in the Scene Graph. This technique encapsulates all the details which allow users to share a common world.

These techniques allow different users to communicate, but they do not deliver messages according to other factors like the identity of the user or their interests. Even in those projects that actually try to build generic environments –as the ones cited before- little effort has been done in defining ways to process and map interactions. This fact makes the development of CVEs a difficult task [7,11]. In this paper, we explain an interaction model for CVEs, which is based on one of the implicit properties of VEs: the physical representation and the definition of interactive areas.

In order to illustrate this model, a reference architecture is also given, and a prototype application is described. The prototype is called CVRPrismaker, which defines a virtual room where several users can collaborate playing with a block-based construction game named Prismaker, similar to the more popular LEGO but this time any face of the box-shaped blocks can be connected to any other block. The design of CVRPrismaker includes the key aspects of the approach presented here. Besides, to carry out its development in a systematic way, the TRES-D methodology has been used [9]. The relevant aspects of CVRPrismaker will be detailed during the explanation of the model: both highlighting the points that established the basis for this model, and also showing the deficiencies in its design. Possible improvements are also shown during that explanation. However, event though CVRPrismaker perfectly serves as a testbed for many of the key elements in CVEs, it is just a prototype implementation, and a new system based on a mature implementation of this model is currently under development. Flexibility is the main objective for this implementation. The system will allow the easy definition of a high set of object's behaviours and will be easy to integrate with many other business models.

2 The Proposed Model

As pointed out in the previous section, the collaborative interaction model for CVEs described in this paper is based on the physical representation. Given the nature of interactions in VEs, a physical representation must be given for every object that allows interaction (even data and other "logical" elements). The user will have to deal with these physical representations to produce its interactions to the system.

Usually, these features are translated in CVEs with the usage of a scene-graph, which holds the information about each object's position and status, enriched which some other "semantic" features (material, weight…). To guide interaction, the system applies some "general" rules to every object.

Instead of using this approach, this model is based on the definition of several geometric views of the same object: one or more *appearance views*, and one or more *interaction views*. The *appearance views* can be exchanged based on the object state or the LOD required. The *interaction views* establish the conditions to determine whether a user action may change the state of objects of the world.

To help understand the behaviour of the model, some key concepts are explained:

- **Action:** Any user input, a contact in a data glove, a key press… is considered as an *Action*. *Actions* are no more than messages, and thus, they need something to process that message. The *interaction views* in the objects will be those filters, and they will tell if this *Action* ends up changing the state of the world or not.
- **Interaction:** If the appropriate *Action* is performed over the appropriate object, the target object will produce an *Interaction*. *Interactions* are responses of the system to a user action through a given object that may change the state of the world. Given that every change in the world will be due to an *Interaction*, it is important that all the clients who have the target object in their scene graphs perform the *Interactions*. Also, given the distributed nature of the virtual environment, the execution of the *Interactions* must be synchronized with the other clients granting exclusive access to the necessary objects. A tight control over *Interactions* becomes necessary to achieve consistency.

2.1 The Geometrical Actions

The interaction management done in the system is based on the concepts above and on one main idea: given that everything in a Virtual Reality application will have a physical representation, it is considered easier to give user *Actions* a representation too, and so it is done in this model. As it has been said when *Actions* were defined, they represent users inputs in the form of a message. That message is of a symbolic type, that is, a tagged value. But, in addition to that, the model associates a geometrical representation to each message. This representation is the part of the space where this message is propagated.

This geometrical representation is usually not as complex as the rendered geometry and, in fact, it is currently implemented by using simpler geometries, more precisely sets of spheres. The system will process each of the user *Actions* propagating the resulting messages to the different objects accessible in his scene graph.

2.2 The Interaction Tree: A Layered Scene Graph

The proposed model, as most VE applications, uses a kind of Scene Graph, but of a completely different kind. As every Scene Graph, it contains the definition of the position and status of every object in the VE. But the information available for each object is different. Each object is defined as a set of different views. Some of the views define the appearance of the object, while others define the kind of action that will trigger an interaction over the object.

The Appearance view. Every object in the scene will have one or more appearance views. This view or set of views defines the "visual" aspect of the object and contains information usually complex and useless for its use in interaction decisions. The union of all the appearance views of the objects of one environment would correspond to the classical definition of Scene Graph given in the literature. To manage this information, standardized Scene Graphs such as OpenSceneGraph or OGRE could be used.

The model defines a restrictive interface to the actual "appearance Scene Graph" used, which only allows it to define the positions of the objects, its appearance and the definition of one objects position relative to another object. This restricted interface is one of the key concepts that allow the model to keep its independency from the actual Scene Graph used in the implementation. Only the appearance views will be Scene Graph-dependant.

The Interaction views. Apart from the "visual" view available for every object, if the object allows any interaction to the user, it will contain (al least) one interaction view.

An *interaction view* determines the geometrical place where a user can perform his or her *Actions* to trigger an *Interaction* over the target object. Also, every *interaction view* contains the conditions the user *Action* has to fulfil to trigger the *Interaction*. These conditions may vary from one object to another and they are based on the geometries of both the *interaction view* and the user *Action* itself, and in the status of the objects involved in the *Interaction*.

For this purpose, the *interaction views* are defined basically as set of very simple geometries, for instance spheres as in the implementation of CVRPrismaker. This technique was inspired in our prototype implementation. The approach followed for this implementation distinguished two views, a graphical view and a 'logical' view. The logical definition described the volumes in every object where interactions could be performed (see figure 1).

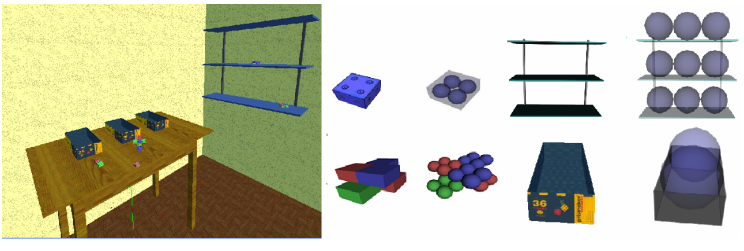

Fig. 1. Logical Views used for CVRPrismaker

The idea underneath is that users are not interested on the whole object, but only on those regions where he or she can perform any interactions; i.e. the selves only could contain nine objects, so nine interactive areas were defined. Boxes did the same, they defined the spaces were users would interact to pick/release objects.

Even though these areas are used in these two objects to model receivers to the user actions, the definition given for pieces and figures gave them a different meaning. These reduced geometries could also be used to define the constraints that

determined whether two figures could be joined or not. This use is explained more in depth in point 2.3.

After some study on these ideas, some other concepts raised. When studying the interactive regions of an object, it is often easy to identify relations between these regions. These relations would allow their division into different views according to some predefined features:

- **Object's functionality:** Each *interaction view* would contain the regions with an associated functionality.
- **User identity:** The *interaction view* will contain the interactive regions accessible for each kind of user in the system.
- **User message:** The object will contain a view for each kind of message the object can process, i.e. if we implement a system where a user can produce 'pick' and 'drop' messages using his gloves, each object would define two *interaction views*, the first for the 'pick' and the second one for the 'drop' messages.

The *interaction views* available for a user define his capabilities over each object. These objects are complete objects, that is, full functionality can be accessed. Furthermore, they process all the *Interactions* produced in the system, even if they come from other users with different *interaction views*. The limitation is for the local user: he can only trigger the *Interactions* available though his *Interaction views*.

Overall, *interaction views* might be considered like *filters*. Those filters are placed in the Scene Graph associated to an object and they process the messages propagated to the object. A*ctions* will be detected by those filters according to the message propagated into the *Action* and to the geometries of both the *Action* (message) and the *interaction view* (receiver).

2.3 The Handheld Objects as *Action* Modifiers

All along this paper it has been said that each of the user's *Actions* would have an associated geometry, but the origin and shape of those *Actions* has not been specified. This avoidance had the purpose of keeping a broad conception about the available geometries for user *Actions*.

Initially, user *Actions* will be modelled with a sphere in a virtual position, and the message within will be the gesture performed by the user. This definition is enough to allow the user to interact with objects directly, but some other behaviours would need more complex *Actions*. For interactions in which the user must use a tool to operate a third party object, the schema used is a bit different. It is not the user's hand but the handheld object position the one which is interacting. In CVRPrismaker, the logical geometry of the object was used to test whether the interaction could be performed. Under this focus, CVRPrismaker's handheld figures transformed the received single-sphere *Actions* into *Actions* with a different message and the geometry of the handheld figure. This way, every object held by the user would transform the *Actions* received, modifying both its geometry and its content according to its own rules.

This behaviour can be used in general modelling tools to manage de assemble/disassemble operations. The object to assemble, i.e. a girder, would transform the spherical 'drop' user message as follows: Its content would be

transformed into a 'join' message, and the geometry used would be the interactive areas placed in the points where the girder can be assembled to a structure.

Also, a bigger concept of transitivity can be used, allowing the successive concatenation of filters/modification of the user *Actions*.

2.4 Interaction Views as *Action* Receivers

This usage of the *interaction views* is one of its simpler uses, but defines an efficient and flexible way to manage interaction. As seen in most VR systems, it is possible to identify a reduced set of messages: due to user inputs (drop, pick, point, etc), or to the used tools (cut, lift, etc). If an interaction view for each of these possible *Actions* is defined, the definition of the world is faced from a very flexible way: if all the *interaction views* available for a given message (i.e. pick) are put together the result will be the Scene Graph of the virtual regions where that action may have an effect (the regions where the user can perform the 'pick' operation).

For each message, a different geometry with the relevant elements to process that message will be used. This geometry may not be a part of the own object, as a result of the own meaning of the message studied. Also, this separation allows us to differentiate which data/geometry will be used to represent the object when processing a given message.

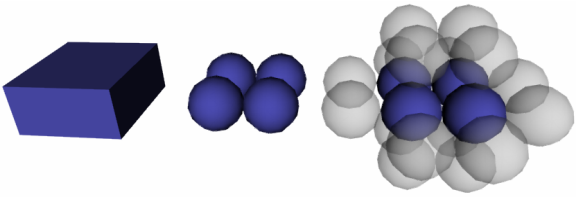

Fig. 2. Views needed for a correct design of CVRPrismaker

In CVRPrismaker, only one *interaction view* was defined for every object. When facing the definition of the pieces, the *interaction view* was well suited to manage 'pick' messages, but not so well suited to the 'join' user's messages. CVRPrismaker should have used the *interaction views* shown in the figure 2: one to process 'pick' messages and a different one for 'drop' messages.

For modelling tools, the interaction data needed to 'join' pieces is about the regions where new pieces can be added (16 for each of the Prismaker pieces). These areas will be spaces around the places where elements may be connected. This can be faced as defining the correct places where pieces or connectors may fit, building the *interaction view* joining these places.

3 A Reference Architecture

At this point, an example implementation of this model is given. The design is simple but faces the challenges of CVEs. Like the model, this architecture uses a distributed

Scene Graph that keeps the same status in every client. The key to achieve consistence is that ***Committed Interaction***s are shared by all the clients. All the clients execute the same ***Interaction***s in the same order and with the same operands (objects and discrete data). This feature will be transparently managed by the system. Even though this feature would be enough to assure system's consistency, the periodical retransmission of the object's status would avoid completely any possible loss of consistency.

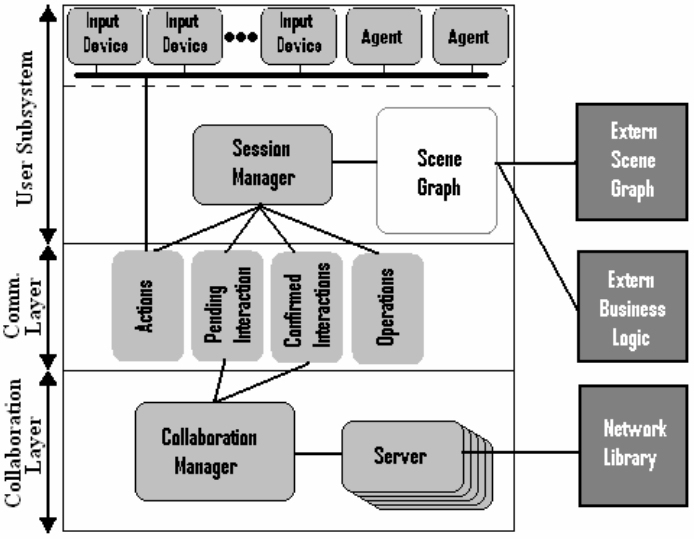

Fig. 3. Purposed architecture overview

This design of the system uses a generic Scene Graph and a Network library. These elements could be taken from the commercial products available. The current implementation of the model is considering OGRE and Raknet as a suitable solution for this purpose, but other tools could be used.

3.1 User Subsystem

It controls one user's view of the world. It contains a copy of the Scene Graph where the appropriate *interaction views* will be loaded. The *Session Manager* is responsible for feeding the Scene Graph with the *Action*s produced. To achieve this purpose, it is responsible of two main tasks:

- **Controlling user actions:** This system will send the *Action* messages produced by the user to the appropriate objects, and will ask for the necessary permission to perform an *Interaction* when this situation occurred.
- **Receiving confirmed interactions:** It will receive all the change messages from the connected clients. These messages will be delivered to the appropriate *interaction view* in the Scene Graph, that executes the *Interaction* appropriately.

3.2 Communication Layer

This layer represents the communication between the user subsystem and the rest of the virtual environment. Through this layer user actions will be declared, checked and, in the appropriate cases, shared to all the other clients. It is implemented as a set of four mailboxes:

- **Actions Mailbox:** All the *Actions* messages produced in the subsystem are sent to this mailbox. The actions are produced by any of the elements in the first sublayer of the User subsystem. These are the *Active Objects* in the Scene Graph and may represent anything able to produce an *Action*. This would include both users and other intelligent behaviours like Agents.
- **Pending Interactions Mailbox:** When an *Action* has fulfilled the conditions of an *interaction view*, it is inserted in this mailbox. The *Collaboration Layer* will determine if the *Interaction* will be executed or not. This decision is made according to the mutual exclusion conditions available over the involved objects.
- **Confirmed Interactions Mailbox:** When the *Collaboration Layer* finds a *Pending Interaction* that can be executed, a message is sent through the *Network library* to all the clients in the system (including the producer's client). Once the messages are received, they are inserted in each on the Confirmed Interactions Mailbox. This is a 'synchronized' mailbox.
- **Operations Mailbox:** Once a client is notified to perform a given *Interaction* over one object, the appropriate actions are taken. Many of these actions will depend on the logic under our interface layer but many others will have to modify this interface itself, modifying the interaction Scene Graph also. This is not a problem itself but, as many systems do, it is common that the drawing of the world is done together with the calculation of the state for the next frame (interaction processing). As this model does not use Scene Graph replication for drawing, those actions that may change the position, appearance and status of the objects in the scene graph are stored here until the drawing is completed.

3.3 Collaboration Layer

This layer is responsible for making all the clients in the system share the same status of the Scene Graph. To control this, the C*ollaboration Layer* will have to send to all the connected users every operation changing the state of the Scene Graph. As the only element which can change the state of the Scene Graph are interactions, this layer will be responsible for making sure that all the *Interaction*s on the system are sent to the rest of the clients. Also, the C*ollaboration Layer* is responsible for telling whether a user *Interaction* can be performed over a given object, or whether the object is being modified by other client and the *Interaction* must be ignored.

4 More Complex Behaviours

The usage of the *interaction views* shown in this paper is very simple. It just uses the *interaction views* to distinguish between the available messages propagated over the system. This view is very restricted, and more complex definitions might be given if

the differentiation of the *interaction views* of the objects in a Scene Graph is done according to other concepts.

If we identify the different views according to the identity of the user, we would be working on a hierarchical environment, where every user would only access the allowed *interaction views*. The *interaction views* could be aware of the internal structures of the organization and provide *interaction views* only for members of a given group or even only for a given user (personal objects).

Also, patterns could be identified so that the users in the system are classified under a given set of roles. This way, it would be the user's role, and not its own identity the one which would determine the *interaction views* to load in the local copies of the Scene Graph.

If the differentiation of the views is done according to the user's knowledge of the object, we come up with a different philosophy. If the interactive regions of the object are differentiated according to the experience needed to understand and use the objects correctly, then an adaptive environment could be defined, with simplified views for novel users and other more complicated for experts.

Also, this model differentiates object's appearance from their functionality. This way it would be possible for two users to have different *appearance views* of the same object. This could be permitted to allow a higher adaptation of the environment. These users would be able to collaborate using their common *interaction views*.

Also, the *appearance views* available could be dependant of the *interaction views* available, allowing one user to see one representation of one object (i.e. a mailbox) while another user, i.e. the owner, can have a completely different perception of that same object (i.e. a 'new staff' folder in his desk).

All in all, *interaction views* are a very new concept and more study over its applicability is still necessary.

5 Conclusions and Future Work

This paper has given an introduction to the *interaction views*. These views allow a new way of defining the objects in the environments; these objects will have a 'layered' definition where they will define both its external appearance, and the relevant geometry for each kind of the available interactions over the object.

Using this concept, an interaction model has been proposed, and a simple execution workflow, assuring the more important properties of the environment, explained. Following this model, and with the purpose of benefiting from our experience in modelling tools, a reference architecture for a CVE has been proposed.

This definition, however, makes a very limited usage of the potentialities of the *interaction views* concept. As a future work, we will keep studying this concept, and a more elaborated proposal will rise.

The final goal is the definition of an application that could work as a skeleton for CVEs. This skeleton would define a clear set of rules that would dictate the way objects are managed in the VE and the logical way through which user actions are filtered and how those actions would be mapped into interactions. Thus, users of this system would only need to define the objects in its VE and, if any kind of new object is needed, define the behaviour of these objects.

Acknowledgements. This work has been supported by the Ministerio de Educación y Ciencia of Spain (CICYT TIN2004-08000-C03-01), and the Junta de Comunidades de Castilla-La Mancha (PAI06-0093).

References

1. Joslin, C., Di Giacomo, T., Magnenat-Thalmann, N.: Collaborative Virtual Environments –From Birth to Standardization. In: IEEE Communications Magazine 42, 28–33 (2004)
2. Oliveira, J.C., Shirmohammadi, S., Georganas, N.D.: Collaborative Virtual Environments Standards: A Performance Evaluation. In: IEEE DiS-RT99 Workshop Proceedings, pp. 14–21
3. Frecon, E.: DIVE: A generic tool for the deployment of shared, virtual environments. In: Proc. of ConTEL 2003, vol. 1, pp. 345–352 (2003)
4. Frécon, E., Stenius, M.: DIVE: a scaleable network architecture for distributed virtual environments. Distrib. Syst. Engng. 5, 91–100 (1998)
5. Steed, A., Tromp, J.G.: Experiences with the Evaluation of CVE Applications. In: Proc. of CVE'98, University of Manchester, U.K., pp. 123–130 (1998)
6. Walsh, A., Bourges-Sevenier.: The MPEG-4 Jump-start Prentice Hall PTR
7. García, P., Montalá, O., Rallo, R., Gómez, A.: MOVE: Component Groupware Foundations for Collaborative Virtual Environments. In: Proc. of 4th International Conference on Collaborative Virtual Environments, 2002 Bonn, Germany, pp. 55–62 (2002)
8. Rousseau, M., Slater, M.: A Virtual Playground for the Study of the Role of Interactivity in Virtual Learning Environments. In: Proc. of PRESENCE 2005: The 8th Annual International Workshop on Presence, London, UK, International Society for Presence Research, pp. 245–253 (2005)
9. Molina, J.P., García, A.S., Martinez, D., Manjavacas, F.J., Blasco, V., López, V., González, P.: The development of glove-based interfaces with the TRES-D methodology. In: Proc. of ACM VRST 06, Limassol, Cyprus, pp. 216–219 (2006)
10. Steed, A., Frecon, E.: Construction of Collaborative Virtual Environments. In: Sánchez, M.I. (ed.): Developing Future Interactive Systems, Idea Group 2005, pp. 235–267 (2005)
11. Goebbels, G., Laliot, V., Göbel, M.: Design and evaluation of team work in distributed collaborative virtual environments. In: Proc. of ACM VRST'03, Osaka, Japan, pp. 231–238 (2003)