

A Generic Communications Module for Cooperative 3D Visualization and Modelling over the Internet: the Collaborative API

C. Pedrinaci¹, J. Aguado¹, J. Azpiazu¹, A. Bernaras¹, A. García-Alonso², H. Sánchez³
¹Parque Tecnológico de San Sebastián, ²Universidad del País Vasco, ³Universidad de Extremadura
{carlos, jessica, jon, amaia}@miramon.net, agalonso@si.ehu.es, sasah@unex.es

Abstract

Cooperative three-dimensional visualization and modeling applications allow a distributed group of users to work together with a model they share. To implement this kind of applications the underlying communications system must provide reliable and ordered multicast of users interactions. Due to the high complexity that characterizes the models, network bandwidth requirements have limited their use to intranets or in a few cases to very high-speed Internet connections.

In this paper we present a communications module that solves this problem. The library exposed, which is called Collaborative API, supports the creation of very efficient cooperative 3D visualization and modeling applications by optimizing the use of the network resources.

The Collaborative API, implements a new communications architecture: the dynamic client/server. The communications module presented in this paper is illustrated by two examples of applications that use it to provide cooperative 3D visualization over the Internet.

1. Introduction

Cooperative visualization over the Internet of three-dimensional models allows a geographically distributed group of persons to share the view of one or several three-dimensional objects. Cooperative modelling can be seen as an extension of Cooperative visualization where it is also possible to dynamically modify the shared model.

Every cooperative visualization or modelling application for Internet must satisfy a set of requirements. First of all, they must provide reliable multicast groups communication. Moreover, these applications have to work in a globally ordered way among the interactions performed by the members of the collaborative session. Also, it is required to have a failure recovery mechanism, that is to say, it must have tolerance to errors. Next, as a consequence of the

previous requirements but also as an added value for the application, it is necessary to have a group membership control system. Last, the use of Internet for establishing communications adds networks bandwidth difficulties. Therefore applications must optimize the communications management to adapt to low bandwidth situations.

There are already several applications that offer cooperative visualization or modeling. However, none is prepared for its use across the Internet, and require high or even very high-speed connections. As a result only the main companies in the world benefit from the use of such kind of applications.

The same way, many efficient group communication libraries are available nowadays. This is the case of JGroups [1], Spread [2] or Ensemble [3], to name a few. Unfortunately, these libraries don't satisfy the requirements collaborative applications have. In particular, establishing a global order between user's interactions is not achieved.

To solve all these difficulties we have developed a communications library that allows developers to implement cooperative visualization or modeling applications over the Internet: the Collaborative API.

We first present an overview of its main characteristics. Next, we explain the underlying architecture that supports it. Afterwards, we expose the different steps required for creating a cooperative tool and we finally illustrate its use with two applications that are currently using the library.

2. The Collaborative API

The Collaborative API is an efficient, reliable and versatile groups communication library suitable for creating cooperative 3D visualization or modelling applications over the Internet.

The whole library has been written in Java. Thus, it is platform independent and avoids the compatibility issues associated to the Internet's heterogeneity. It is encapsulated in a 63KB jar, which makes it suitable for its use in applets and even in PDA's. Moreover, it has been optimized in a way that allows home users to

profit from collaborative work benefits, as it does not require any additional hardware or high-speed network connections.

2.1 The dynamic client/server architecture

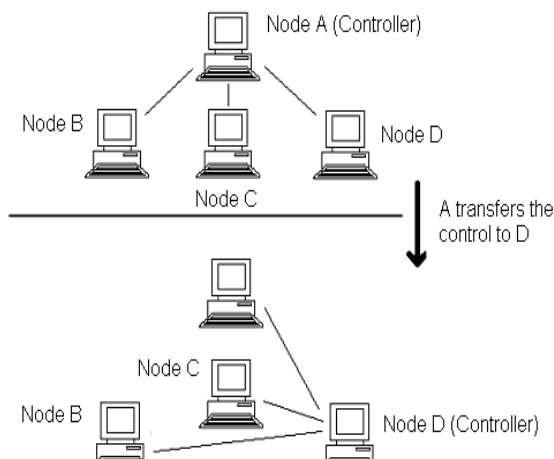
The Collaborative API implements a new communication architecture we have called the dynamic client/server. It uses the most appropriate features of the client/server and distributed architectures, applying a peer-to-peer approach. The result is a new architecture suitable for supporting cooperative visualization or modeling over the Internet.

In the client/server architecture, the server centralizes the information and serves it to the clients. Thus, the server is the main component of the architecture and usually does all the processing work. Using such architecture for collaborative visualization or modeling applications facilitates establishing a global order between users interactions. However, it has some drawbacks associated. First of all, the fact that a server failure provokes the whole system to fall down does not provide errors tolerance. Next, this approach requires having a very powerful computer acting as the server. Last, sending models to client machines establishes very high restrictions over the network bandwidth to be used. As an example of such an approach SGI has created the OpenGL Vizserver [4] where the hardware and network requirements are excessive even for medium-size companies.

In the distributed architecture, the information is distributed among all the hosts. This solution is harder to manage but provides tolerance to errors. Nevertheless, in that case it is very difficult to establish a global order among users' interactions.

To benefit from both architectures advantages we have created a new architecture that allows the server role to be switched in run-time (see Figure 1). Like in peer-to-peer [5] approaches, the Collaborative API implements both the client and the server and acts accordingly depending on the situation. Again, the server (also referred as *controller* in this paper) is a critic node, thus, the library implements a replacement mechanism that automatically transfers the server role to an active host.

The dynamic client/server architecture offers the possibility of a new paradigm for collaborative applications management. The applications can follow



a distributed architecture by replicating the information among the hosts. That way, the whole system is error tolerant but what is more important, communications can be optimized. Applications can use a short messages protocol. As a consequence the messages interchanged during a cooperative session are very small and their transfer is fast even with low bandwidth connections.

2.2 The communications support

To be able to apply this new paradigm for managing cooperative visualization and modelling applications, the underlying communications technology must satisfy some requirements.

First, it has to provide reliable multicast capabilities.

Figure 1. Dynamic client/server architecture

This includes loss less transmission of messages to the participants of a cooperative session. Moreover, the order between the messages must be kept. Finally, for ensuring coherence between all the member's workspaces, the system must satisfy the principle of atomicity. In other words, a message has to be received by all the participants or none.

Second, as a consequence of the principle of atomicity, the communications technology has to provide a robust group membership control. It is necessary to keep track of the systems that are taking part in the cooperative session.

The communications architecture relies on TCP/IP for establishing the connections. This way, thanks to the protocol features, the Collaborative API provides reliable communication. In particular, TCP/IP grants loss less and ordered transmission [6]. Moreover, it deals with packet fragmentation and reassembly whenever the message's size exceeds the maximum supported. It is true that TCP requires a three-way handshake connection, which is not required in UDP. However, given that control switch will only happen whenever a new user requires or wants to interact with the system, the network bandwidth and time spent in establishing connections is negligible compared to the benefits associated to the use of TCP. Moreover, the use of UDP for multicasting, that is, the use of IP multicast, requires multicast routers and class D IPs. Such a requirement is unacceptable if we want let home users take part on a collaborative session. Thus, the Collaborative API extends the reliable unicast transmission TCP offers, to reliable multicast by replicating messages. This represents an overhead on the controller side, which is minimized by handling every connection by a thread. This solution has proven to be efficient for the applications we want to support as the number of members in a session rarely exceeds ten users.

The collaborative module we have developed provides a robust group membership management control. Thanks to its internal data structures, every Collaborative API instance keeps track of all the members that are taking part in a session as well as their associated IP and listening port, improving the time spent when switching the controller node. Moreover, every new connection, even those that are refused, are notified to the participants of the session. The same way disconnections are also notified to the members of the collaborative session. Finally, the internal data structures do also provide the means for reacting upon controller failures, by automatically selecting a substitute for the server node. This is performed by selecting the next active node from the members list.

Another important characteristic concerning the communications module is the fact it is based on well-known design patterns [7]. These confer to the API a modularity that results in an easier maintenance and development. In particular the patterns are used in the messaging mechanism but are also used for providing a very powerful and versatile feedback mechanism for cooperative application developers.

Regarding the messaging mechanism, the collaborative module has been developed in a way that allows modifying or replacing a particular message independently from the rest of the API. The same way, it allows cooperative application developers to easily implement their own application protocol. Programmers do not have to worry about the underlying communications mechanisms as the Collaborative API transparently handles it. The messages are divided in two sets. The first set is formed by all the internal messages of the communications library, like for example *DisconnectionMessage*, *ConnectionMessage* or *TakeControlMessage*. The second set concerns the application's messages and does only provide the means for developers to implement their own messaging policy based on the application requirements.

The Collaborative API provides the mechanisms needed to inform applications about every situation that may take place during a collaborative session. Thus, the API users, that is collaborative application developers, are informed any time a message arrives and have available a simple interface to treat them. This allows providing procedures that are triggered every time a particular message arrives. In addition, the API provides methods for obtaining all the information an application may require concerning the collaborative session, like the users connected, the current controller, etc.

To better illustrate the characteristics presented but also to present the steps required for integrating the

Collaborative API, in the next section we briefly explain how to use the API.

3. Using the Collaborative API

As we have previously explained, thanks to the dynamic client/server architecture, the Collaborative API allows optimizing the use of network resources. For that, collaborative applications that make use of this communications library must follow a short messages policy. Using an approach similar to the use of client-side scripts in the web, the server node only needs to send a particular command when the user interacts. Client nodes, receive the message, interpret it and apply it to their own working memory. Given that the communications module we have developed provides reliable multicast communication, every participant's workspace is consistent with the rest of the session members.

A direct consequence associated to the use of a short messages policy, is to remove the need for a central server with high processing power. Instead, the workstations that are usually used for visualization or modeling tasks can be employed for cooperative visualization or modeling work.

It is also important to say that the short messages policy encourages the modularization of applications functionality. Software Engineering has proven the benefits of modularization in software applications, concerning maintenance or improvement tasks. Thus, the use of the Collaborative API indirectly improves the applications that integrate it.

The API we have developed is characterized by its ease of integration into collaborative visualization applications. Integrating the communications module into an application is as simple as extending the *CollaborativeInterface* class and creating a set of messages.

The *CollaborativeInterface* encapsulates all the functionality of the communications module. It abstracts the developer away from collaborative session management. At the same time, it provides all the utilities the collaborative application developer may require, like *JoinSession*, *BroadcastMessage*, etc. Only the feedback mechanism has to be implemented if needed. Implementing application behavior based on the messages (internals to the API or belonging to the application that uses it) received during a collaborative session, just requires programming the associated abstract method. The internal mechanisms of the Collaborative API directly take care of triggering the appropriate method in run-time. Therefore, no active waiting is required and the developer is given the possibility to treat every message reception.

The creation of application specific protocols is also very simple. Every message needs to extend the *App_Message* class. This class ensures the message transferred contains all the information about the sender. The developer of a collaborative application is thus, abstracted away from the communication internal details but still has the data required for implementing an application protocol. It is worth noting that applications do not require implementing any message for providing chat capabilities. Given that this is a typical feature in collaborative applications, it has already been integrated in the API.

To illustrate the use of the Collaborative API, we present in the next section two applications that use it.

4. Use cases

The Collaborative API has been used in two applications to benefit from its cooperative communication capabilities. The first one is a cooperative 3D visualization for CAD models. The second one implements a new paradigm for visiting virtual 3D worlds modeled in VRML.

4.1. 3DShared

3DShared [8][9] is a freeware cooperative visualization application platform independent. It uses the Collaborative API to provide real time visual exploration and multiple-user interactions over three-dimensional models among geographically remote users. It allows engineers, modelers and customers to view the virtual design of products through its three-dimensional replica. Its user interface is presented in Figure 2.

3DShared is composed of two modules: 3D API and the previously described Collaborative API. The 3D API takes care of the aspects related with the visualization of three-dimensional models. The Collaborative API ensures the correct communication among the members of the cooperative session. As the design and implementation of both modules is independent, 3DShared offers the functionalities of both modules in the same interface. Consequently, the cooperative visualization of three-dimensional models is allowed.

3DShared benefits from the capabilities offered by the Collaborative API and the dynamic client/server it implements. To do so, it uses a short messages policy. That way, the use of network resources is optimized. During a collaborative session, the controller host communicates the interactions its user is performing

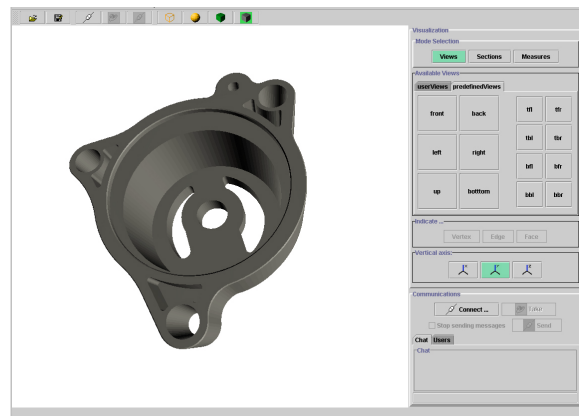


Figure 2. 3DShared user interface

over the model. The Collaborative API of every

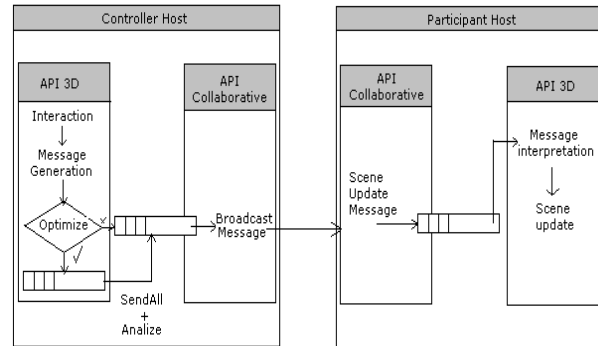


Figure 3. 3DShared communication process

participant host receives the messages. These messages are then handled by 3DShared, which ensures they are correctly interpreted by the 3D visualization module. This communication process can be seen in the Figure 3.

To better support collaboration 3DShared has three communication tools. The first one is a chat tool. Its functionality being directly offered by the Collaborative API. The second one is a file transfer tool. This is used on session establishment for transferring models to the session members. The same way, every time a new user joins the ongoing session, the current state of the model is transferred. Finally, the 3D-window where the virtual models are rendered represents the main communication utility between the users.

Several experiments have been made to measure the speed in the transfer of the graphics file and the updating messages of the scene among several hosts. These experiments have shown that the Collaborative API optimizes network bandwidth use as 3DShared has been used with 56Kbps modems [9].

4.2. Guided visit through a 3D virtual world

Traditional virtual 3D tours consist on a fixed path, established when designing the virtual world. When entering the world, or activating a sensor within it, the user's avatar is automatically taken through a sequence of fixed points, allowing the creator of the world to make greater emphasis in the areas that he or she decides. But this paradigm shows two clear drawbacks: on the one hand, the lack of interactivity of the user, who is limited to watch what is being showed to him, with no possibility of changing the course of the visit. On the other hand, the path will have been defined at the moment of designing the world, reason why it will be the same one for all the users who visit the world, that is to say, is not customizable or adaptable to the needs of the visitor.

The guided visit is a new navigation paradigm in virtual 3D worlds over Internet [10]. It allows making real-time adaptable visits through a virtual 3D world. Moreover, it improves usability by allowing users to control the system.

In the guided visit paradigm, users are connected to a master user that guides them through the virtual world. The guide knows the world, and can move freely through it. All the movements performed by the guide are automatically reproduced in the guided systems.

In order to improve the usability of virtual 3D worlds visits, the guided visit must allow to dynamically interchange the role of guiding between the users connected. That is, every user that is being guided through the world should have the possibility to become the new guide, and start guiding the rest of the users involved.

The Collaborative API provides the appropriate communications system by means of the dynamic client/server architecture it implements. Moreover, the system does not require additional hardware (as a specialized server or multicast routers) or software to be used.

The guided visit (see Figure 4), has been developed integrating the Collaborative API. It uses the concept of a collaborative session to set up a group of users visiting a virtual 3D world over Internet: Miramon virtual. In [11] it is possible to perform a guided visit of San Sebastian's Technology Park using this approach.

The whole system is packed as an applet, thus, it can be used with a web browser. Once the applet and the virtual world have been downloaded to the client computers, communications are performed directly between the users involved. In low bandwidth environments the system has proven to be very efficient thanks to the use of a short messages policy in order to reduce the size of transmissions.

5. Conclusions

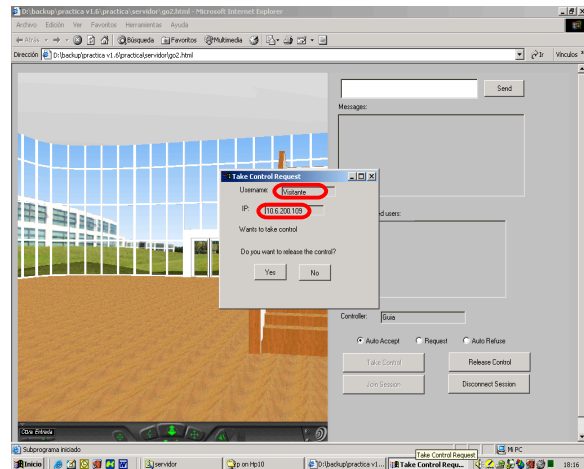


Figure 4. Guided visit through 3D worlds

To provide an appropriate communications system for cooperative 3D visualization and modelling applications we have developed the Collaborative API. It is a platform independent, generic, efficient and versatile communications library.

The Collaborative API implements a new communications architecture called dynamic client/server. This new communications architecture is based on the client/server architecture but distributes the information between the members of the collaborative session. As a consequence, the server role can be switched in run-time among the different participants of a cooperative session. Moreover, the library provides reliable multicast communications and a reliable groups management system.

Applications that use the Collaborative API benefit from an ordered dynamism between users interactions allowing them to better work in cooperation. In addition the communications infrastructure allows the use of short messages policies in collaborative applications for optimizing the use of network resources.

The library has been integrated in two applications that provide cooperative 3D visualization facilities over the Internet. The applications developed have proven the benefits of the communications paradigm, but also the power of the library that encloses it. The small size and the efficiency of the library, make of the Collaborative API a suitable utility for providing collaborative capabilities to applications that have to be used across the Internet, without any additional hardware and with low bandwidth.

6. References

- [1] B. Ban, "JGroups – A Reliable Multicast Communication Toolkit for Java", Project Home Page: <http://www.cs.cornell.edu/Info/Projects/JavaGroupsNew/index.html> (Last visited: November 2003).

- [2] Y. Amir, C. Danilov, and J. Stanton, "A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area Group Communication", International Conference on Dependable Systems and Networks (FTCS-30, DCCA-8), New York, June 25-28, 2000.
- [3] T. Clark, "Ensemble – Distributed Communication System", Project Home Page: www.cs.cornell.edu/Info/Projects/ensemble, (Last visited: November 2003).
- [4] J. Lefauchaux, "Visualisation avancée pour un travail collaboratif", MICAD 2003, April 2003.
- [5] D. Clark, "Face-to-Face with Peer-to-Peer Networking". Computer, vol. 34, n°. 1, January 2001, pp. 18-21.
- [6] A.S. Tanenbaum, *Computer Networks*, Prentice-Hall, Inc., 1996.
- [7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.
- [8] D. Borro, I. Recio, C. Pedrinaci, H. Sánchez, A. Garcia-Alonso, "Peer-to-peer techniques applied to the cooperative visualization of CAD models", MICAD 2003, April 2003.
- [9] D. Borro, I. Recio, A. García-Alonso, H. Sánchez, L. Matey, "CSCW for foundry design using Java3D", Demonstration at ACM 2000 Conference on Computer Supported Cooperative Work, Philadelphia, 2000.
- [10] J. Azpiazu, C. Pedrinaci, J. Aguado, A. García-Alonso, A. Bernaras "A new navigation paradigm for virtual reality: the guided visit through a virtual world" to be published in the Proceedings of the IEEE-VRIC 2004, Laval-Virtual, Laval, France, 2004
- [11] J. Azpiazu, C. Pedrinaci, J. Aguado, A. Bernaras, Miramón Virtual, Home page: http://www.miramon.net/miramon_virtual.html, (Last visited: April 2004)