An Object-Oriented Groupware Framework for Developing Collaborative 3D-Modelers

Luiz Gonzaga da Silveira Jr
Wu, Shin-Ting

Electrical and Computer Engineering Faculty
State University of Campinas
P.O.Box 6101 13083-970
Campinas, SP - Brazil
{gonzaga,ting}@dca.fee.unicamp.br

Abstract. We present an object-oriented framework to support the development of collaborative (real-time shared) 3D modeling systems. For data integrity reasons, our research aims at enabling geographically dispersed end-users to create, to inspect, and to modify interactive and synchronously the shareable 3D geometric data in different (heterogeneous) computing environments.

1 Introduction

3D modeling systems play an important role in manufacturing, architecture, engineering analysis and simulation, science and education. Many of these application areas rely on people working and collaborating together, with the 3D information being exchanged on asynchronous mode. The typical way of communication is sending facsimiles of drawings or transferring files. However, the quality of communication within a group of end-users may be improved by allowing collaborative and interactive work on a shared 3D-model over a network [13].

One of the key decisions for implementing a synchronous collaborative 3D application is its underlying architecture. There are three paradigms: centralized, replicated and hybrid approaches. In the centralized architecture there is only one instance of the application in a server and its output is sent to the display of each client in the way similar to the X-window display protocol. Alternatively, in the replicated architecture one instance of the application runs on the client's workspace to process locally replicated data. For consistency and synchronization reasons, hybrid approaches that take the advantages of these two paradigms emerged. The commercial product OneSpace [13] provides an infrastructure that allows heterogeneous CAD systems to exchange the design data in IGES3D and STEP formats over a network and allows low-powered systems to visualize the shared model data.

We present in this paper another hybrid paradigm for collaborative 3D modeling systems. Our approach lets the end-users not only inspect but also interact with the geometric model data, regardless the computing power of their workstation. In this approach the modeling software is centralized in a server and the end-users interact with a simplified model, renderable on any machine with graphics capabilities. The problem, however, is to provide an intuitive direct 3D interaction support in each client's workspace for benefiting from the modeling and displaying functionalities of the modeling kernel.

Our solution relies on the use of metaphors with simple geometry to convey complex geometric operations. Such metaphors should be easily displayable by any 3D graphics resources and be capable of invoking (remote) functionalities at the model's accuracy for performing desired operations. For visual feedback purpose, the metaphors should also have capabilities for redrawing by themselves with local graphics resources, when their attributes are changed. In this case, one may decouple a high-precision geometric representation from a set of graphics primitives. Only a reliable communication protocol between them needs to be established for accomplishing every interaction cycle.

The development of such a collaborative 3D modeling system requires distinct technologies, ranging from data sharing, communication, and coordination control to single-user interactive graphical interfaces from which an end-user can manipulate the shared 3D information. Hence, the developers of collaborative 3D modelers must be proficient in several knowledge areas, including distribution and concurrency strategies, shared data management, and 3D computer graphics, in order to make the best use of available software and hardware resources. It makes the application development a difficult and error-prone task.

This motivates us to design a framework for supporting the development of a collaborative 3D modeling system with our proposed hybrid architecture – `3dig` framework. The `3dig` framework is an application framework comprising of a set of abstract and concrete classes, which can be specialized and/or instantiated to be tailored to a particular set of needs for producing a custom collaborative 3D modeling system. It contains rigid aspects (*frozen spots*) common to all applications, and variables aspects (*hot spots*) tuneable to each application [5, 3, 4].

2 Previous Work

Concerning with groupware system one may distinguish three architectures:

replicated architecture - each end-user interacts with an individual instance of the application. The application-domain model is fully replicated, such that each application processes input events and displays the information independently. Only the most fundamental data elements are shared. When an event is dispatched, the consistency and concurrency mechanisms ensure that all replicas are atomically updated and the shared data are kept in a consistent state (Figure 1).

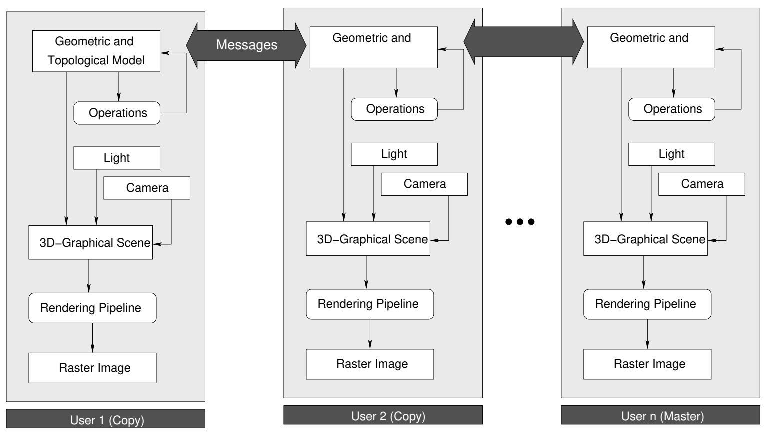

Figure 1: Replicated architecture for collaborative systems

centralized architecture - only one instance of the application executes on a central server and only the interfaces are distributed. When an event is generated, it is collected and passed to the central server where it is processed. Any necessary visual feedback in each workspace is also coordinated by the central server (Figure 2).

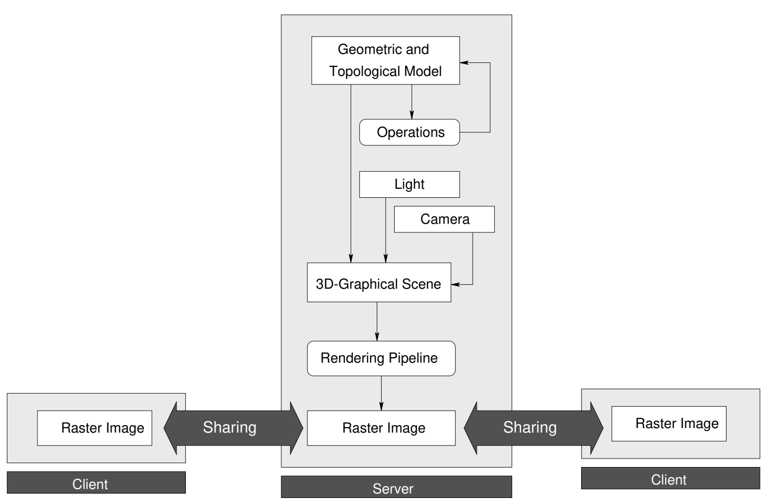

Figure 2: Centralized architecture for collaborative system

hybrid architecture - results from the combination of the both centralized and replicated architectures.

The advantages of the replicated approach are that the network traffic is reduced because communication does not go through a central mediator, and that the system is more robust with respect to network and machine failure. As drawbacks we may mention the difficulties for solving the concurrent (user) requests and for implementing consistency mechanisms to ensure e.g. the robustness on geometric operations.

In the centralized scheme the concurrency and consistency problems are less critical, since the information is located in one place. However, the system is vulnerable to the failure of the central server and this server, together with the network, is overloaded by updates that are not necessarily shared by every end-user.

Both centralized and replicated architectures can be found in the streaming-based systems (such as video/audio conference), the collaborative editors and multimedia systems. Examples of products with centralized architecture are NetMeeting [14], SharedX[7], and XTV [2], and with replicated one are GroupKit [17] and DistEdit [11]. The replicated approach is preferred when the efficiency is required, while the centralized one is a usual choice when simplicity is demanding.

With regard to collaborative 3D graphics applications, the replicated architecture is mostly used. As examples for this mode we may cite the projects Repo-3D [12], DIV [10], IntelligentBox[16] and TOBACO [18].

To our knowledge, OneSpace [13] is a collaborative CAD system with hybrid architecture. It distinguishes two classes of applications: servers and clients. The server includes modeling kernel and holds the 3D geometric product model, whereas the client only needs to have rendering capabilities. The servers exchange the information among them in neutral CAD data formats, and transmitted its data to a client in a viewable tessellated format (Figure 3).

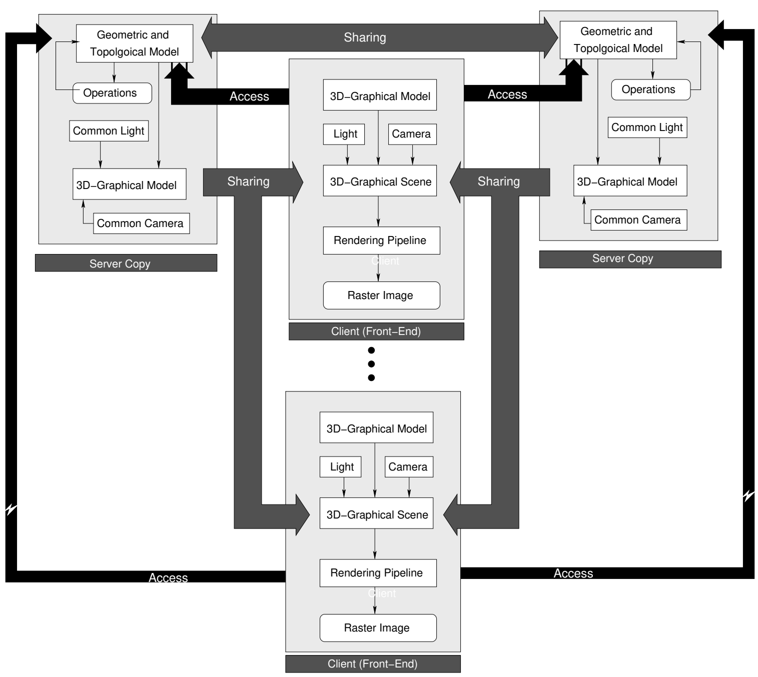

Figure 3: OneSpace: a hybrid architecture.

3 An Object-Oriented 3D Modeling Groupware Framework

The general scenario for interactive and collaborative 3D modelers is a shared workspace where a dispersed group of end-users work together on creating and modifying 3D geometrical models.

Because of round-off error propagation, processing of the same sequence of geometric operations on distinct machines may deliver different results. It may be prohibitive in some modeling applications. On the other side, each end-user should be able to inspect and analyze the shared 3D model on her/his needs without affecting other users. For these reasons, we propose a hybrid architecture for collaborative 3D modeling systems, where two equivalent database are kept: a geometrical and a graphical database. The geometrical database is common to all end-users, whereas the (viewable) 3D graphics data are replicated for visualization and manipulation (Figure 4).

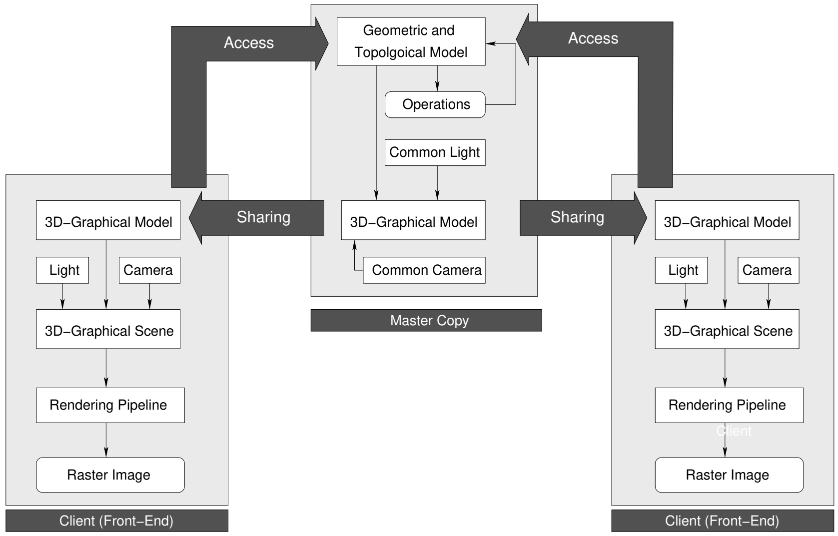

Figure 4: `3dig` framework: a hybrid architecture

To support the development and implementation of a collaborative 3D modeling system whose underlying architecture is hybrid, we propose an object-oriented interactive 3D groupware framework, `3dig framework`, which not only provides an efficient inspection and visualization of 3D models on each of participant workspaces, but also ensures the data consistency at any manipulation as well. The `3dig framework` is comprised of four modules, whose functionalities are provided through an object-oriented interface (Figure 5):

① `3D-model manager` - provides a simple interface to the application-dependent 3D shared models and several services for control data sharing and concurrent access in order to maintain the data consistency.

② `graphics facilities` - consist of 3D graphics functions, including graphics interactions, rendering and pictorial group metaphors.

③ `group manager` - holds information about work session, membership policies, and the role of each member in the group in order to coordinate the teamwork.

④ `distributed platform` - makes the information available to every group member through a network.

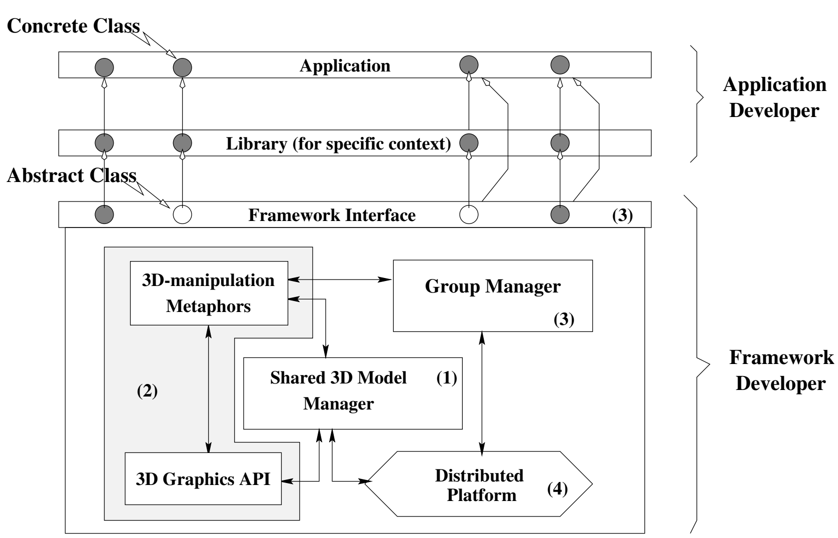

Figure 5: The components of `3dig` framework

Since graphics facilities in any participant machine (client) and the control flow between them and the 3D-model manager (server) are the central issue of this paper, the former two modules are described in details in the following sections. For completeness, some functionalities of the latter two modules are also given.

3.1 The 3D-Model Manager

More natural and precise semantic feedback can only be provided, when the user interface "knows" the underlying geometric model and benefit from geometric techniques such as intersection, deformation, and simulation methods associated with sophisticated geometric shapes. Because of a variety of geometric representations, the concept of the 3D model manager in our framework was one of the challenging issue. It is of particular interest:

- to distinguish the rigid aspects (*frozen spots*) of a geometric modeling system from the customizable ones (*hot spots*) and
- to establish the mapping between those aspects and the generic functionalities of interfaces with 3D interaction support.

Due the limitations of a raster graphics display, the collection of primitives provided by a 3D geometric modeler is usually much more specialized than the ones provided by commonly used 3D graphics libraries, such as OpenGL[15]. In a raster precision we may describe any complex object from a finite number of points, straight lines, and polygons. A hierarchical data structure comprising of these primitives is, in the most cases, sufficient for graphics purposes. Hence, it is a *frozen spot* in our framework. Such a hierarchical structure can be represented through a structural composite pattern [6] where primitives and composed primitives may be dealt with uniformly (Figure 6).

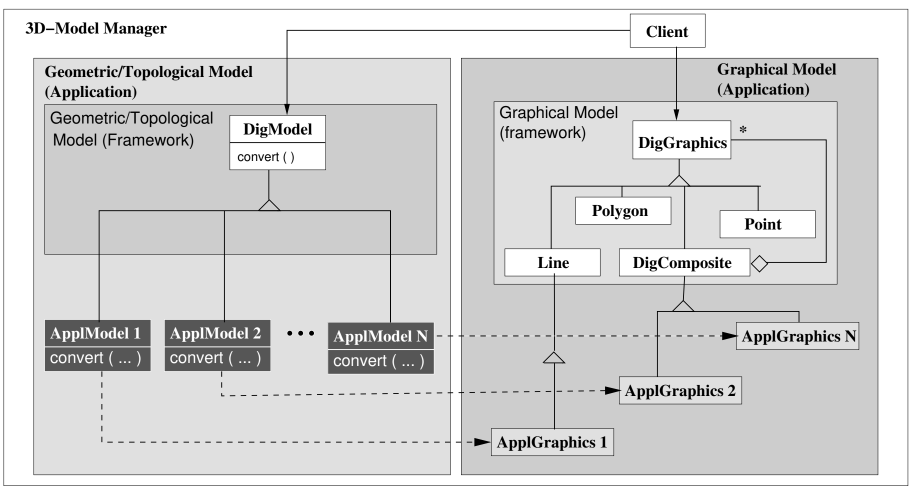

Figure 6: Relationship between geometric and graphics data.

The `DigGraphics` (abstract) superclass holds the default behavior (attributes) and declares the common interface of its subclasses (methods). The subclasses `Point`, `Line`, and `Polygon` represent leaf primitives and define the graphic attributes and methods for their particular behavior. The `DigComposite` subclass is a recursive aggregation of the `DigGraphics` class, allowing more complex objects to be defined.

To be flexible enough for accommodating distinguishing features of a variety of geometric models, the geometric data and geometric methods in our framework are *hot spots*, customizable by the application developers. They are instances of `ApplModel` class, which is subclass of `DigModel`. To ensure a uniform data handling over the network and to be generic enough for being displayable on any workstation (Section 3.2), one of the common method to all geometric objects is the `conversion` algorithm, from its native data format into our graphics model format. Then, for each `ApplModel` class is required define an equivalent `ApplGraphics` class, which holds graphics data of the geometric object.

For any object, the 3D-model manager keeps both its geometric and graphics representations and ensures their consistency. Whenever a change is occurred in geometric data, its graphics data are also updated. While the geometric data and methods reside in a server, the graphics data are replicated for each end-user workspace where they are rendered (Figure 7). In this way, the speed with which each client re-renders the graphics objects after any view change depends only on the local computational power. Nothing needs to be transmitted over the network. If the geometric data are updated in a server, the changed part must be converted into graphics data format and multicasted from the server to the clients. In this case, instead of the whole graphics scene or complex native 3D geometric data, only the modified graphics data are transmitted.

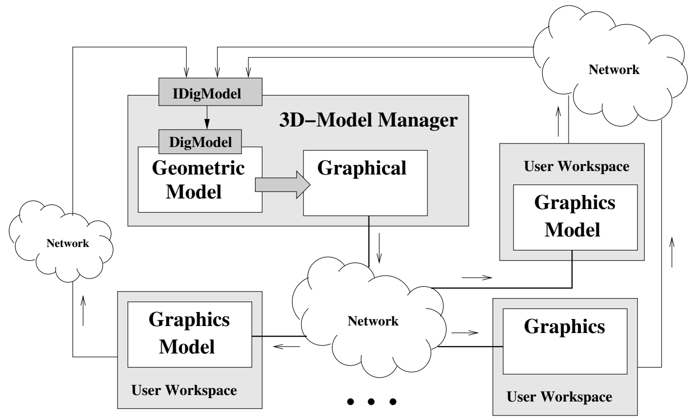

Figure 7: Distribution of 3D data over network.

The `IDigModel` interface defines the client software's user interface. It provides the commands for accessing the methods of the underlying geometric model. It allows the end-users to make changes on the 3D-model as it resides in the client's workspace. As the instances of the subclasses of `DigModel` are *hot spots* of our framework, `IDigModel` must also be a *hot spot*. The remote access may be through a (remote) proxy [6] in the client workspace. We call this proxy the `DigModelProxy` interface (Figure 8). It is a local (virtual) representative of the `DigModel` class, hiding the fact that objects may reside in another address space, often in another machine.

A broker pattern [1] may be applied for implementing the control of the transparent communication between a proxy and its real objects. Knowing the the real object location, the requests and their arguments are collected and encoded by the proxy in a client before being sent to the (real) object in a server, where the operations are actually accomplished and, if necessary, the graphics data updated.

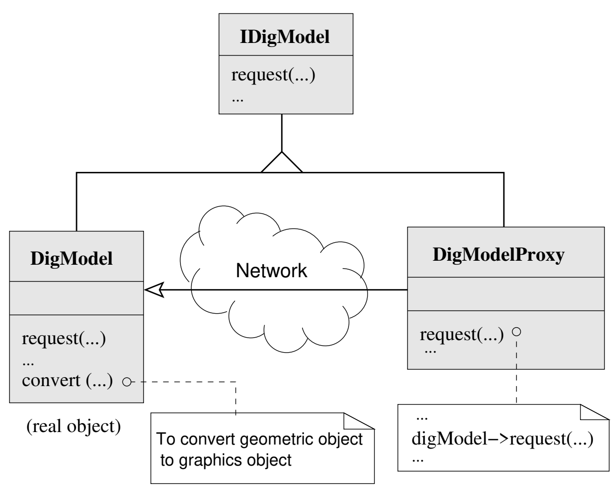

Figure 8: The remote proxy structure

Summarizing, the 3D-model manager includes a modeling software to be customized by the application developers to produce custom applications and a software to manage a graphics model which is common to all of custom applications. The set of commands provided by the modeling software is accessible remotely through the `DigModelProxy` interface in any client workspace and any change on the geometric data is communicated from a server to its clients by sending the corresponding updated graphics data. In this approach changes on the graphics data on a client's workspace do not affect the network traffic.

3.2 Graphics Facilities

Under graphics facilities we include a set of 3D graphics functions that support visualization, direct interactions with the 3D models presented in the canvas, and groupware awareness. There are several graphics libraries, such as OpenGL [15], that try to accomplish real-time visualization purposes, but they provide very little support for interaction beyond simple picking and/or selection mechanism. It is because that "they fall short of exact 3D object representations" [19]. To remedy this deficiency, higher-level programming toolkits, e.g. Open Inventor, were proposed. The basic idea relies on the integration of application-dependent models and graphics interface into the same development environment. As already summarized in Section 2, to move from a single-user to a multi-user platform the replicated architecture was adopted in several groupwares. Every group member runs their own instance of application in a process on their own machine and the processes are connected to each other over the network via some communication mechanisms. In this way the network traffic may be reduced drastically – instead of a stream of data only operations and arguments are sent over the network.

Hybrid architecture differs from the replicated one. One difference comes from the site of the shared geometrical and topological information and the graphics data: the former is located in a server and the latter is replicated for each client. This strategy avoids unnecessary network traffic. Any change on the graphics data, such as a view change, may be processed locally. Another difference arises from the fact that there is a dedicated server for running 3D modeling software. It ensures data integrity, since the geometric processing is independent from the computing environment of any client. However, to support such application-graphics loosely coupled systems, we should establish a communication protocol between the servers and the clients in order to provide more natural and precise semantic feedback while an end-user is interacting.

Metaphors play an important role in helping users to understand the actions that they should perform in order to correctly interact with computer systems. A metaphor may be implemented as a widget, which encapsulates geometry (pictorial representation) and behavior (constraint) used to control or display information about application-dependent data. In our project we are particularly interesting in graphics widgets (for 2D graphics devices in client's workspace), which (1) improves group awareness, and (2) permits a virtual 3D cursor acting on the shape or the position of a 3D graphics object exactly as the user finger would operate a concrete one – hence, a nice visual feedback may be provided while a user is interacting with the system for inspection or manipulation.

Because the semantics of a metaphor's geometry and movements depends on the application context, it should be provided by the custom application developers as needed or desired. Moreover, the relationship between the controlled objects and the metaphor should also be established on the need of the custom application. Once defined those relationships, the 2D input events are captured and converted into semantic meaningful operations whenever the end-user interacts with a metaphor. Then, the metaphor requests the correct manager to apply those operations on the associated data and to update the corresponding graphics data. The `Manipulator` class implements the metaphor objects (Figure 9).

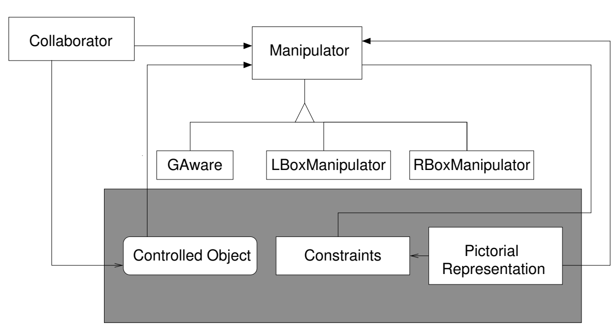

Figure 9: Manipulator Class

In our framework three classes of controlled objects are distinguished: geometric, group awareness, and viewing/lighting objects. For controlling geometric objects the 3D-model manager is invoked by the metaphors to handle the required operation properly, while the group awareness depends on the information managed by the group manager (Section 3.3). Finally, the viewing and lighting parameters are controlled by the graphics facilities resident in each client's workspace. For the sake of clarity, we defined three classes of manipulators, `RManipulator`, and `Lmanipulator` for manipulating geometric and viewing/lighting objects , and `GAware` for giving group awareness, respectively.

`RManipulator` carries out operations in the 3D-objects by requesting the methods provided through the `DigModelProxy` interface. `GAware` class bridges the graphics entities with the group session information, such as the current state and the location of a group member. And `LManipulator` class invokes the local graphics software for altering the cameras and lights parameters. For supporting a common view of a model it is allowed that the viewing/lighting parameters are sent on demand from one client to the others (Figure 10).

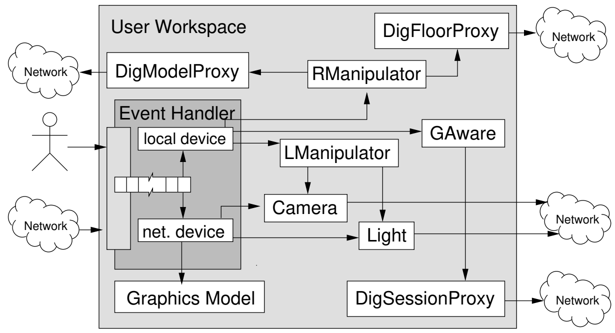

Figure 10: End-user workspace

3.3 Group Manager

A work session is a group of end-users working together on a specific problem. Our framework provides the `DigSession` class for managing the group members and their roles in a work session according to the policies specified in the `DigSessionConfig` class.

The end-users are classified into three classes: (1) `Member`, whose elements have only viewing access right and may manipulate their own viewing and lighting parameters; (2) `Collaborator`, a subclass of `Member`, whose elements have the viewing and manipulating access right; and (3) `Chairman`, whose elements coordinate the session and its internal activities (each session has one chairman).

Because of potential conflicts that may arise from simultaneous access of a geometric object by multiple end-users, concurrency control to avoid/solve resource contention is required. Such a control is called floor control, once each object has its own floor. When a floor is associated to a member, she/he becomes the owner of the corresponding object and nobody can access that object until the floor is released. The floors are instances of the `DigFloor` class.

For each work session, the chairman establishes:

- **integrity criteria** - the validity conditions for minimum and maximum number of members in a session (e.g. one session requires at least two members to be established, otherwise it is released or suspended). They can be

  ① *hard* - the session is released if the integrity criteria are not satisfied.

  ② *soft* - the session is suspended if the integrity criteria are not satisfied.

- **membership policy** - establishes how an end-user join and leave a session. All end-users can negotiate an invitation to join a session. We have considered three policies

  ① *static* - end-user must join a session by previous negotiation and before the work has been started.

  ② *dynamic and closed* - each end-user must be explicitly invited to join the session.

  ③ *dynamic and open* - end-users can join a session on invitation or by own initiative at any time.

- **floor control policy** - establishes how an end-user can access a geometric object. The most known policies are

  ① *request-and-get* - allows a requesting member to get the floor immediately, possibly by preempting the current floor-owner.

  ② *request-and-wait* - allows the requesting member to get the floor only when it is available.

  ③ *no-floor* - there is no policy for attending multiple simultaneous requests.

Since a group manager can present multiple work sessions, we defined the abstract class `DigSessionFac` that declares an interface for creating each work session. As already explained, a work session is implemented by the concrete class `DigSession`. Therefore, the group manager calls the operations in `DigSessionFac` for building a work session, without being aware of the concrete class it is using.

Figure 11 depicts the main components of the group manager of our framework.

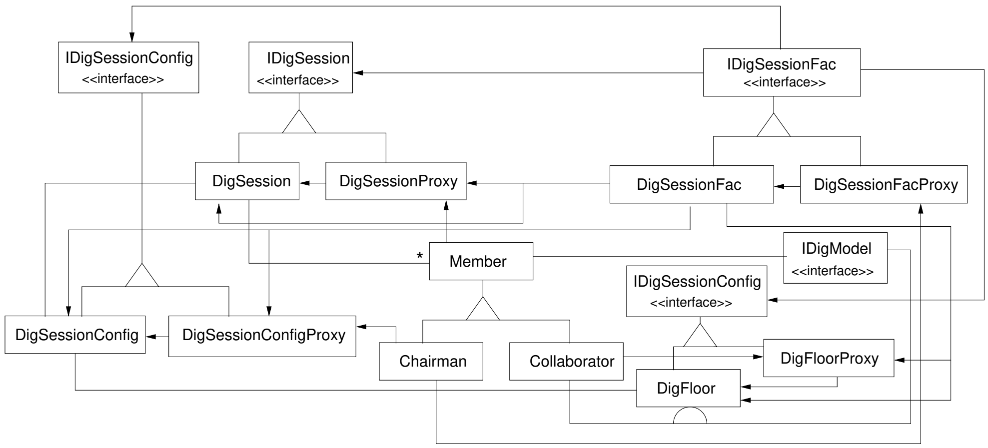

Figure 11: Group Manager.

3.4 Distributed Platform

The distribution and communication facilities of our framework are conforming CORBA (Common Object Request Broker) distributed platform standard [9]. CORBA is an object-oriented infrastructure that allows objects communications, independent of the specific platform and programming language used to implement these objects. Indeed, it is a middleware that enables heterogeneous objects to transparently invoke remote operations and receive replies in a distributed environment (Figure 12).

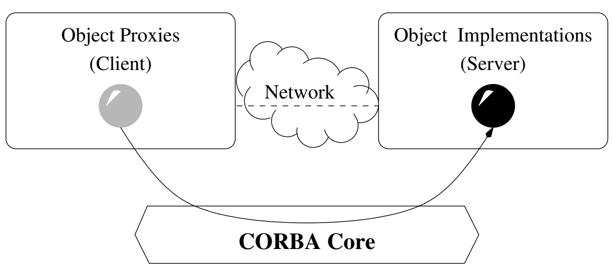

Figure 12: Object Request Broker (CORBA)

The implementation of the interfaces for invoking remote commands, such as `IDigModel`, may benefit from IDL, also provided by CORBA. From specifications in IDL (Interface Definition Language), the broker and proxy functionalities are generated automatically for controlling the communication between a server and a client.

The required multicasting mechanism for updating graphics model whenever the geometric data is changed may be implemented with the CORBA Event Service [8]. This Service decouples the event producers and consumers and provides facilities for reliable one-to-many communication through one (or more) event-channel. In our framework we adopted the push-model configuration with one event-channel for handling the graphics replication. The graphics model resides in the 3D-model manager is the master one and the replicated models in the clients are slaves. Whenever a change occurs in the geometric data, the master graphics model posts the event-data (updated graphics data) into the channel and the slaves consumes the event-data from this channel (Figure 13).

For transmitting over the network the hierarchical structure of graphics data (Figure 6) is converted into a flattening structure as shown in Figure 14.

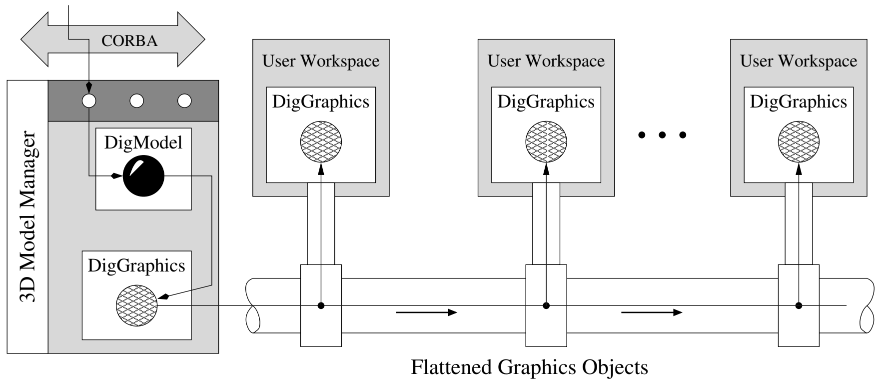

Figure 13: Graphics objects replication

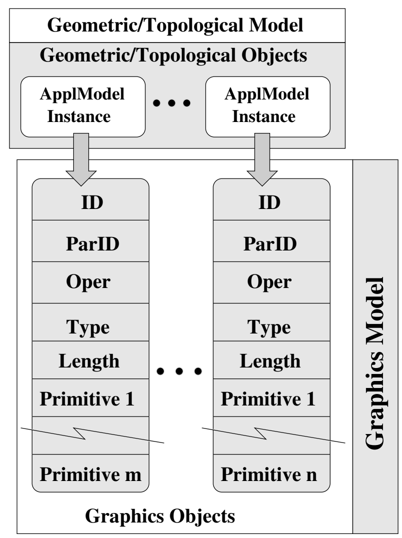

Figure 14: The flattening of graphics data.

4 Implementation

We have implemented a subset of the proposed framework facilities. To reduce the implementation effort, several available technologies were integrated in the framework to provide its rigid aspects. A CORBA implementation called MICO and OpenGL (Mesa3D) were used to provide distribution and 3D graphics processing, respectively. A set of manipulators, implemented with help of FaMa, were incorporated to realize necessary metaphors.

The critical problem we faced in our implementation was the choice of an appropriate window manager system for each client's workspace to handle uniformly the events from both the input devices (e.g. mouse and keyboard) and the CORBA. The `Gtk−−` toolkit (a C++ wrapper of `gtk` toolkit) with a special widget to support OpenGL(Mesa3D) - `gtkglarea−−` (a C++ wrapper of `gtkglarea` widget) was our solution. It multiplexes the incoming events from two sources and put them in two separate queues for correct processing.

For test the concept of our framework, a custom application has been produced with the `3dig` framework. It is a collaborative triangle modeler. The modeling kernel only provides five methods for creating and manipulating triangles: `create`, `remove`, `rotate`, `scale`, and `translate`. The `Trackball` metaphor was chosen for denoting the operation `rotate`, whereas the `Box` metaphor is used for conveying two operations, `scale` and `translate`.

Figure 15 shows a screenshot of the application with free clients running on distinguishing machines. Observe that, although the geometric data (a set of triangles) are shareable, the viewing and lighting parameters are specified locally on each end-user's workspace on her/his need. However, when the shared viewing is demanding in the information exchange a client may send its parameters to its partner over the network.

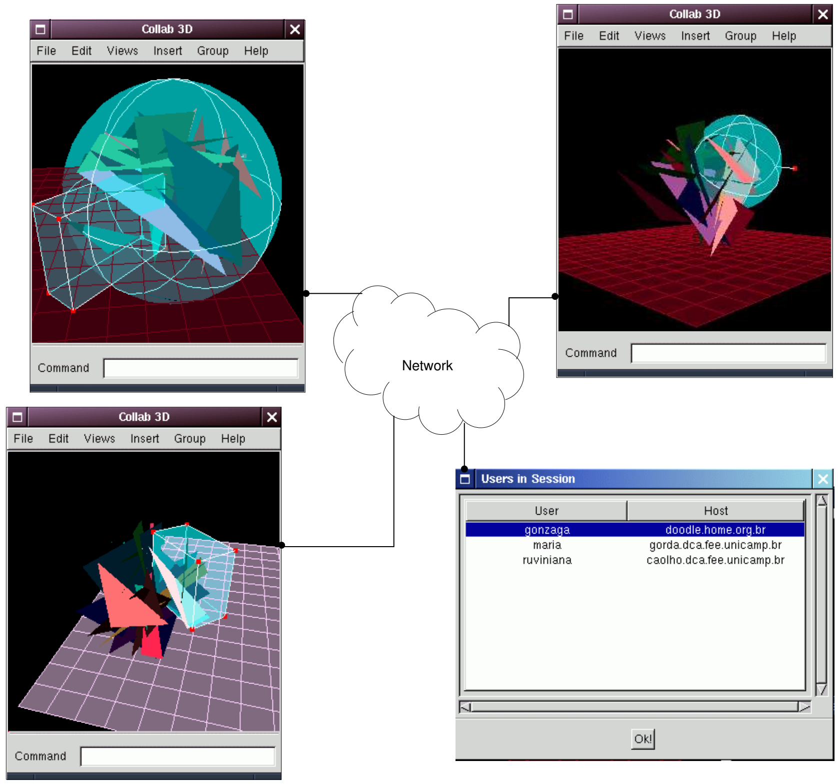

Figure 15: A collaborative triangle modeler.

When an end-user (on the right top) rotates a triangle with the help of the `Trackball` metaphor, every client's graphics data are updated automatically (Figure 16). This example shows a shared viewing, giving the same viewpoint of scene for both users.

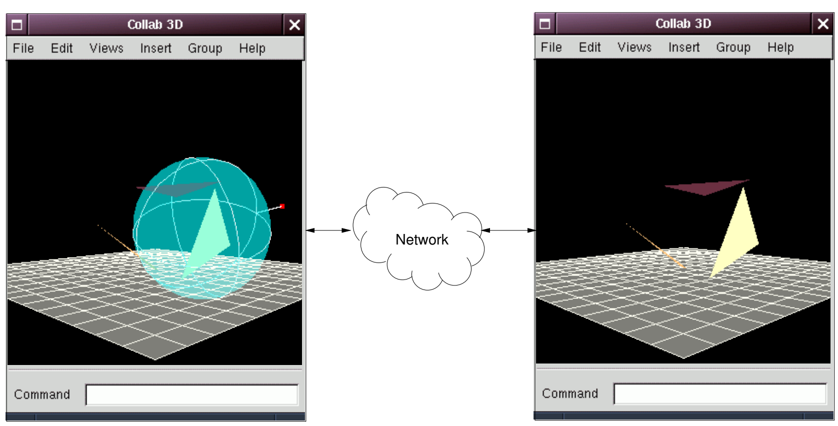

Figure 16: Automatic updates on every workspace.

5 Concluding Remarks

The object-oriented framework with a hybrid architecture has been designed to provide a high-level interface for support the development of a custom collaborative 3D modeling system. Its concept is based on several industry standards including OpenGL for graphics processing and CORBA for communication and distribution of geometric objects.

The key component in our framework is the `Manipulator` class which bridges the graphics components on each client's workspace with the complex information centralized in servers. Hence, high processing capability on each client's machine is less demanding. Only conventional graphics functionalities are sufficient.

Two issues, however, must be addressed in the further work for evaluating the performance of our framework with the proposed hybrid architecture: (1) how flexible it is for accommodating any modeling kernel, and (2) how well the end-users can interact with the modeling kernel over the most used network connections.

References

[1] F. Buschmann, R. Meunier, H. Rohnert P. Sommerlad, and M. Stal. *A System of Patterns: Pattern-Oriented Software Architecture*. John Wiley and Sons Ltd, Oct. 1996.

[2] G. Chung, K. Jeffay, and H. Abdel-Wahab. Dynamic participation in computer-based conferencing system. *Journal of Computer Communications*, 17(1):7–16, Jan. 1994.

[3] M.E. Fayad, D.C. Schmidt, and R.E. Johnson. *Building Application Frameworks: Object-Oriented Foundations of Framework Design*. John Wiley & Sons, 1999.

[4] M.E. Fayad, D.C. Schmidt, and R.E. Johnson. *Implementing Application Frameworks: Object-Oriented Frameworks at Work*. John Wiley & Sons, 1999.

[5] Mohamed E. Fayad and Douglas C. Schmidt. Object-oriented application frameworks. *Communications of the ACM*, 40(10):32–38, Oct. 1997.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Desin Patterns – Elements of Reusable Object-Oriented Software*. Addisson Wesley, 1995.

[7] D. Garfinkel, B.C. Wleti, and T.W. Yip. Hp sharedx: A toll for real-time collaboration. *HP Journal*, April 1994.

[8] OMG Object Management Group. Corba event service specification - v1.0. `http://www.omg.org`, 1993.

[9] OMG Object Management Group. The common object request broker: Architecture and specification - version 2.0, 1995.

[10] G. Hesina, D. Schmalstieg, A. Fuhrmann, and W. Purgathofer. Distributed open inventor: A practical approach to distributed 3d graphics. In *Virtual Reality Software & Technology '99 (VRST'99)*. ACM, Dec. 1999.

[11] M. Knister and A. Prakash. Distedit: A distributed toolkit for supporting multiple group editors. In *Third Conf. Computer-Supported Cooperative Work*, pages 343–355, Oct. 1990.

[12] B. MacIntyre and S. Feiner. A distributed 3d graphics library. In *SIGGRAPH '98*, pages 361–370, 1998.

[13] Mackrell. Cocreate onespace - a collaborative design infrastructure. `http://www.cocreate.com`, Jan. 1999.

[14] Microsoft. The Windows NetMeeting Zone. `http://www.netmeeting-zone.com`, Ago 1999.

[15] J. Neider, T. Davis, and M. Woo. *OpenGL - Programming Guide - Release 1*. Addison Wesley Co., 1993.

[16] Y. Okada and Y. Tanaka. Collaborative environments of intelligentbox for distributed 3d graphics applications. *The Visual Computer*, 14:140–152, 1998.

[17] M. Roseman and S. Greenberg. Building Real Time Groupware with GroupKit - A Groupware Toolkit. *ACM Trans. Computer-Human Interaction*, 3(1):66–106, Mar. 1996.

[18] ZGDV Rostok. Tobaco - cooperative modeling with corba.

[19] J. Wernecke. *The Inventor Mentor*. Addison Wesley, 1994.