An architecture for collaborative geomodeling

Luciano P. Reis[1,2], Alberto B. Raposo[3], Jean-Claude Paul[4,5], Fabien Bosquet[6]

[1] Petrobras Research Center, Brazil
[2] Institut National Polytechnique de Lorraine, France
luciano.reis@petrobras.com.br

[3] Tecgraf, Computer Science Dept., PUC-Rio, Brazil
abraposo@tecgraf.puc-rio.br

[4] CNRS, France
[5] Tsinghua University, China
paul@tsinghua.edu.cn

[6] Earth Decisions Sciences, France
bosquet@earthdecision.com

Abstract. This paper presents an architecture for distributed synchronous collaborative visualization and modeling applied to the geosciences. Our goal is to facilitate the creation of heterogeneous collaboration sessions, in which participants may use different versions of a core CAD application, configured with specific functionalities and multimedia user interfaces, through the composition of run-time plugins. We describe the domain requirements, the architectural concepts that facilitate the integration of our collaboration plugins with the core application, and the management of communication channels to allow the definition of role-based control policies adapted to specific types of sessions.

1 Introduction

Geomodeling, the computer-aided design of geological objects and their properties [17], involves a large spectrum of skills spread over different domains: geophysics, geology and reservoir engineering. A numerical earth model is shared by people with different types of specializations and evolves continuously through a team effort. In the oil and gas industry, during the exploration and production of a reservoir, new data is constantly acquired, and the model needs to be frequently updated as new decisions have to be taken based on the most up-to-date information.

Effective geomodeling is strongly graphics-based. High-performance graphics make it possible for the professionals involved in the process to interactively visualize and edit the integrated three-dimensional models. Visualization is used as a powerful tool for data understanding and insight, as a support for interactive modeling, and as a common language for communication and collaboration within multidisciplinary teams. Currently, virtual-reality applications are starting to be used to enhance comprehension and to improve precision in some modeling tasks [11, 14].

Given the geographical dispersion of operations and professionals in the industry, and the increasing availability of computing, graphics and networking resources,

remote collaboration offers great potential to improve distributed cooperation and decision-making. The scarcity of geomodeling experts also makes this technology very important for consulting and training. However, currently only a few commercial modeling applications are starting to offer some collaboration functionalities, mainly the synchronization of points-of-view among remote users .

A key problem faced by the companies is how to integrate different types of tools, legacy and new, to compose a comprehensive software solution that provides a coherent and efficient environment for remote collaboration.

Therefore, the objective of this work is the development of an architecture for remote synchronous collaborative modeling and visualization applied to oil and gas exploration and production. This application domain is closely related to other areas, like collaborative engineering and collaborative scientific visualization. However, some specific requirements influence the design of the proposed architecture. In particular we are interested in facilitating the coordinated use of heterogeneous user interfaces and interaction paradigms by participants in multi-disciplinary collaborative sessions with the support of multimedia communication, taking into account the different types of collaborative activities to be performed, the roles of the participants and the communication and cooperation channels involved.

Unlike related collaboration solutions [1, 19, 25], we do not expect the development of tools compliant to a predefined architecture. Instead, our purpose is to facilitate the transformation of a well-designed operational application, which follows established design principles [9] and is extensible by run-time plugins, into a collaborative system through the introduction of cooperation and communication mechanisms. Our implementation is based on Gocad [10], a pioneering geological modeling application.

Along with the core application and its functional plugins, the proposed collaboration solution involves the development and integration of:

- A collaboration plugin responsible for providing session management services and for supporting synchronous collaboration through the creation of "communication channels" (commands, camera, telepointers, 3D annotations, avatars, model distribution, audio and video) among distributed instances of the application, with broadcasting and floor control mechanisms;
- A custom tool for multimedia communication, also integrated as a plugin, providing audio and video channels subject to the defined control policies;
- A virtual-reality plugin compatible with the collaboration functionality provided;
- A real-time data-acquisition plugin, used for automatic integration into the model of data arriving from remote well-drilling sites [2].

In this article, after an overview of some application scenarios, we emphasize the description of the coordination mechanisms proposed for dealing with the different channels used for cooperation and communication.

2 Collaborative geomodeling

Before we proceed to the description of the proposed architecture, let us discuss some typical application scenarios and related requirements.

2.1 Collaboration Scenarios

Consulting

User *A* is building a model, and faces a problem on how to perform a certain task. *A* then creates a collaboration session (a conference), invites user *B* to join from a remote site, transfers the model and explains the problem, turning the model around, zooming in the area where the problem occurs, pointing to things and making annotations that *B* can see in a synchronized way, as if both were looking at the same display. *B* may also manipulate the camera and make annotations directly on the three-dimensional model, which are seen by *A* (Figure 1). For the two participants only, conversation could take place over the phone. However, integrated videoconferencing, automatically established among the participants of a conference, simplifies and improves the communication.

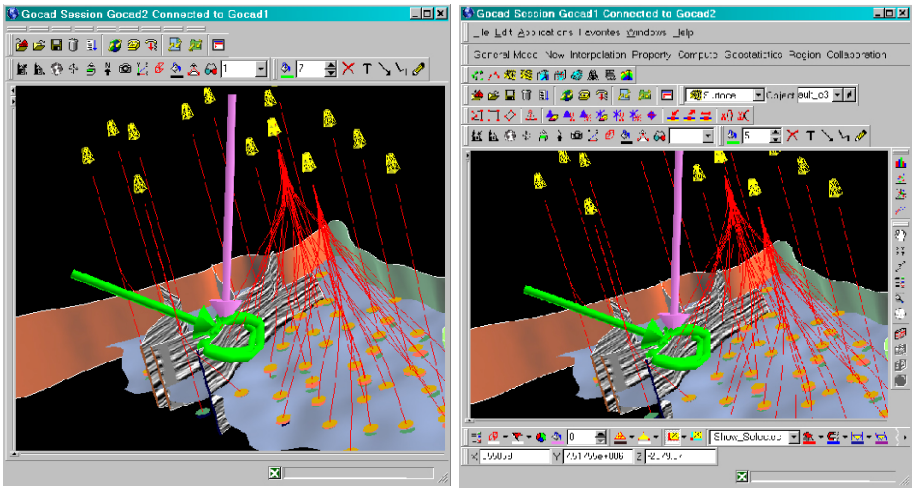

Fig. 1. Screenshots of two users discussing a model, using 3D cursors and annotations (differentiated by colors).

Conflicting camera movements are detected immediately, and a direct negotiation for the control can be done during the discussion over the audio channel, without the need for explicit passing of the camera control. However, if more than two participants are involved, negotiation of control through a verbal protocol becomes awkward, interfering with the main discussion, and a turn-taking mechanism becomes necessary. If both participants need to edit the model, floor passing is also used to regulate the transfer of the control of modeling commands, avoiding possible inconsistencies due to conflicting operations.

Well planning

An important collaboration scenario is the design and real-time steering of oil wells. While a directional (non-vertical) well is drilled, data acquired at the well bit in the subsurface needs to be received at the office from the remote drilling location and incorporated into the model, shared by local and remote geoscientists, in nearly real time to support collaborative decisions about the steering of the well trajectory.

This activity involves at least two sites: one or more modeling experts working together at the office, using a desktop version of the application loaded with a well-planning plugin with specific functionalities, and an operator at the drilling site, typically using a less powerful computer and a lightweight version of the application to visualize the shared model, with restricted editing rights. All participants use the plugin for real-time data acquisition. Usually very limited network bandwidth is available at the drilling site, requiring strict control of the use of audio and video. Our system has been frequently applied to this scenario for the survey of actual operations.

Virtual-reality (VR) applications are now starting to be used for the design and steering of complex horizontal wells [14, 13]. However, while some modeling tasks like the design of the well path or a local modification of a surface may be facilitated by the immersive interface, other global modeling tasks become more difficult with an "inside the model" point of view. Also, the immersive interface usually needs to provide restricted functionalities, for ergonomic reasons. We propose that the use of VR as part of heterogeneous collaboration sessions can bring in the benefits but avoid the shortcomings. In this case, a participant in the session uses the VR plugin in an immersive virtual environment, collaborating with other desktop participants. Among other usability issues, camera position events cannot be exchanged between the desktop and the VR users, since their navigation metaphors are totally different.

2.2 Requirements and consequences

We have observed and taken part in operational collaboration sessions in the scenarios described above. These ethnographic observations have suggested some specific requirements for the proposed architecture, both from the users' and from the system developers' points of view, which guided our design choices described below.

Small groups: the activities considered involve a small number of participants. Therefore, scalability is not an issue in this domain. This allows the adoption of a centralized coordination scheme, simplifying the solution.

Graphic performance: visualization and modeling require very responsive interaction (maintenance of high frame rates), demanding an appropriate treatment of the graphics-related communication channels. This implies that the implementation of coordination policies cannot degrade performance. For this reason, for graphic interaction events, we propose a floor control mechanism based on the setting of input and output switches for each channel in the host application, thus avoiding the processing of the real-time events by the external coordination logic.

Integrated multimedia communication: audio and video are fundamental components of a synchronous collaborative modeling system. In some systems this is provided through external tools, requiring separate session management and controls. We consider that audio and video need to be provided as integrated communication channels. As the use of multiple audio and video streams poses strong requirements over bandwidth and processing consumption, conference participants need to be able to control individual connections, subject to the role-based policies defined for the conference.

Reuse of single-user functionality: application deployment is one of the most challenging aspects of groupware development [7]. Our target applications require the integration of a very large set of tools and multidisciplinary modeling functionalities, typically developed over many years. Therefore, the objective of the architecture proposed is to allow the flexible extension of an operational single-user application into a collaborative system through the incorporation of plugins, not requiring recompilation and redistribution.

Concurrency control: geological modeling involves the construction of diverse types of mathematical objects (surfaces, meshes, solid models), with complex geometry and topology, interdependency relationships, and a very large number of elements (a single version of some objects may easily approach the available memory size). Currently it is not possible to assume that modeling operations may be always rolled back efficiently. In this context, effective consistency maintenance guaranteeing high responsiveness and concurrency is still not feasible [23].

Therefore, the approach presently adopted for concurrency management in modeling is to actually avoid conflict through the use of mutual exclusion by the floor-control setting for the modeling commands (or, optionally, to rely on a social protocol). As the mechanism proposed is extensible, more sophisticated floor strategies [5, 6] and concurrency control [23, 8, 21] can be used in the future.

Awareness: for effective collaboration, group awareness needs to be provided by different means [21]. The list of participants in a session, their roles and control rights over the different communication channels need to be easily accessible. Annotations, telepointers and avatars need to be associated to the participants with visual cues, like labels and colors.

Heterogeneous operation conditions: networking resources for distributed participants are typically very heterogeneous. To guarantee the required responsiveness we have opted for a replicated architecture [4], in which the basic cooperation mechanism is the broadcasting of commands and interaction events over the communication channels, requiring low network bandwidth.

3 Collaboration architecture

The proposed architecture was designed in the context of the scenarios and requirements described above. A collaboration plugin (NetGocad), loaded at runtime, allows the transformation of the single-user application (Gocad) into a distributed col-

laborative system through its extension with broadcasting and control capabilities, by means of the incorporation of CORBA objects and of a configurable coordination component.

The separation of coordination policies from computation, as proposed elsewhere [15, 3], and the definition of collaboration types, participant roles and floor control over the channels through an object-oriented scheme, implemented with a simple and powerful interpreted extension language (Lua), greatly simplifies the evolution of the system.

3.1 Gocad

Gocad (Geological Objects Computer Aided Design) is a CAD software, originally developed by an international research consortium [10], that allows the construction of earth models for geophysics, geology and reservoir engineering applications. Its architecture is based on the systematic use of Design Patterns [9] such as: Abstract Factory, Builder, Chain of Responsibility, Command, Composite, Factory Method, Interpreter, Iterator, Observer, Proxy, and Singleton. It also provides a flexible development framework to allow the creation of plugins, which can be dynamically loaded at runtime inside the application shell.

The implementation of NetGocad, described in the next section, is therefore facilitated by this framework, in particular by the use of the Command, Abstract Factory and Observer Patterns.

Gocad uses the Command Pattern to isolate the processing of operations from their invocation at the user interface. All menu operations generate string commands that are then executed, facilitating the creation of a mechanism to broadcast them to remote servers. The use of Abstract Factories and Observers allows the run-time redefinition of classes of the main application by the plugin. This mechanism is used to create new observers for broadcasting commands and graphic events.

3.2 NetGocad

Figure 2 shows a simplified diagram with NetGocad's main classes, responsible for the collaboration functionality. Most of them are implemented in CORBA, chosen as the distribution middleware because it is language-independent and multiplatform, and facilitates the integration with interoperability solutions used in the industry [18].

The central class in this architecture is the Participant, which acts as the main server for remote client requests, invoking operations in the local instance of the application. The Conference, instantiated by the Manager, keeps track of a group of Participants. It provides the same interface for channel events (section 3.4) as Participants do but broadcasts them to its members, either directly or through LuaConference, as discussed below. The abstract class Partner acts as a superclass for both Participants and Conference.

The main CORBA classes are described below:

Partner: Declares the interface for the methods that treat the communication channel events, actually implemented by Participants and Conferences. Clients keep references to a Partner.

Participant: Implements methods for treating communication channel events by invoking the appropriate operations at the host application (command execution, setting of the camera position, etc.).

Conference: Manages a group of Participants and broadcasts channel events to them. Also keeps track of floors: if a communication channel is controlled, the participant controlling the floor is the only one allowed to send events through this channel, as described in the next sections.

Manager: Manages the overall conferencing system. For a given domain (a company, for instance), it is responsible for registering and listing participants, creating and listing conferences, and allowing clients to connect to remote participants and conferences. It is the only published object, instantiated by a daemon, and serves as the entry point for the creation of collaboration sessions.

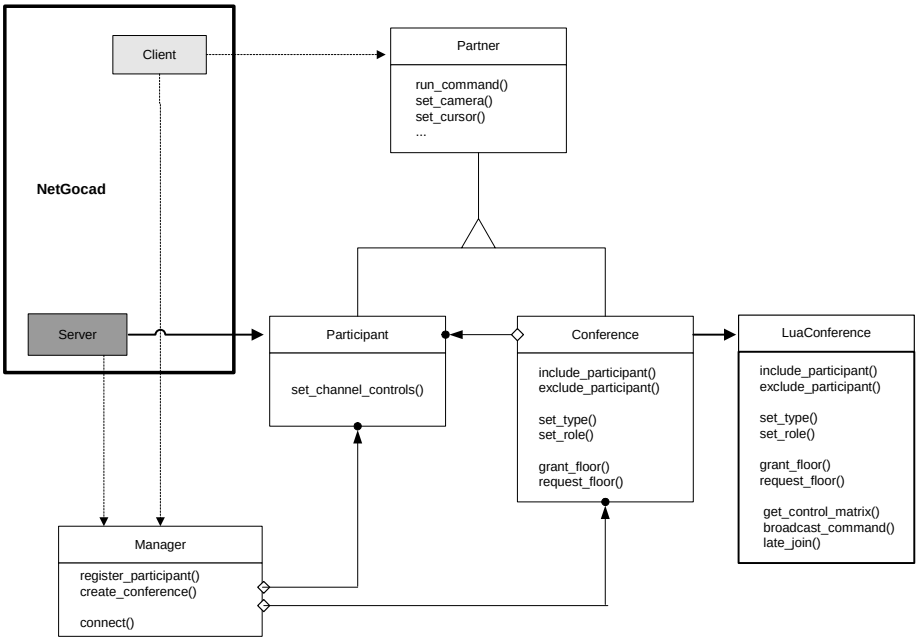

Fig. 2. NetGocad main classes (simplified).

Client, Server and LuaConference are not CORBA classes:

Client: Singleton class [9] implemented in C++. Instantiated by the application, provides methods for connecting to the Manager and to Partners, keeping references to them. When a user performs an operation locally (executes a command, a camera movement, etc.), a corresponding *observer* (redefined by the plugin for the applica-

tion) will use the Client instance to invoke the same operation on the Partner it is connected to.

Server: Singleton class implemented in C++ used by the application to instantiate a Participant and to register it with the Manager.

LuaConference, a module implemented in the Lua extension language [12, 16], is created by each Conference at runtime and becomes responsible for the coordination policies. It loads the definitions of collaboration types and participant roles (section 3.5), and interacts with the Conference to provide configurable services (session management, floor control, definition of conference types and participant roles).

3.3 Operation

For a collaboration session to take place, a daemon instantiating a Manager needs to be running, responsible for registering and listing available Participants and Conferences, and for providing connection services.

When a group of people wants to collaborate, first somebody creates a conference, choosing a collaboration type among the ones loaded by LuaConference, and automatically becomes the conference owner. Then the other participants call the conference, choosing one of the roles specified for the current conference type. If the owner accepts a call, the caller is allowed to join and receives rights over the communication channels according to the chosen role (Figure 3). Afterwards, whenever a participant sends an event to the conference over a communication channel, it is broadcasted to all members enabled to receive it.

If somebody starts audio or video communication, the appropriate streams are created, according to the policy in place, through the videoconferencing tool (section 4).

3.4 Communication Channels

We define a “communication channel” as an abstract path for conveying information among instances of the application (commands, camera positions, telepointers, etc.), subject to a control policy. To each channel corresponds a method responsible for treating events, declared in the Partner class and implemented by Participants (except for audio and video, which are treated separately).

For the definition of the control logic in LuaConference, all channels are treated as elements of a homogeneous array, for which control matrices are defined. However, for the implementation of broadcasting mechanisms, we subdivide channels into *interaction channels*, *command channels* and *streaming channels*.

Interaction channels carry events that are generated by direct manipulations inside the 3D camera (camera movements, telepointers, avatars), that need to be processed at high rates for smooth graphic interactivity. Three-dimensional cursors (telepointers) identifying their users by color or label (Figure 1) allow participants to point to shared objects. Avatars display the position of the camera and the frustum of a participant, in conferences involving participants with non-synchronized points of view.

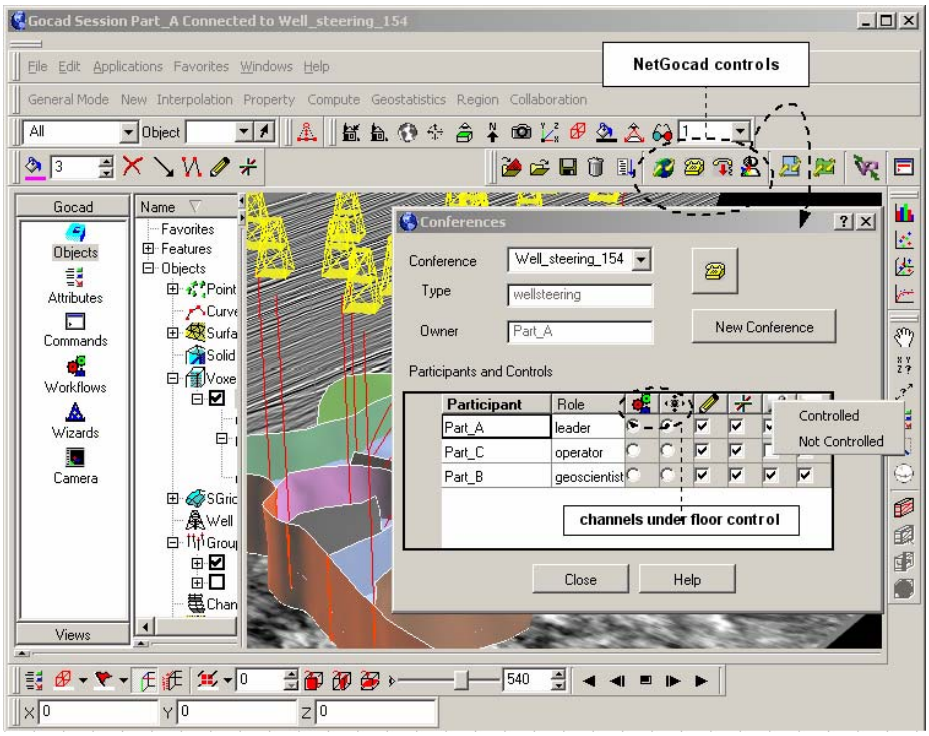

Fig. 3. Conference user interface.

These events are not passed through the conference virtual machine (LuaConference). Instead, their broadcasting is done by the CORBA Conference and controlled by the switches (control matrices) set for the channel, as described below. If all participants are free to send and to receive the events, broadcasting is done through CORBA's Event Service. Otherwise, this service cannot be used, because it provides no event filtering. The Notification Service (an extension of the Event Service) does allow event filtering and independent quality of service (QoS) control for each channel – an important feature, since different channels have different performance requirements. In this case, filtering would be based on the connection matrix, but currently this service is not employed in our implementation because it is not available in the CORBA version used [22].

Command channels carry the application commands, directly sent by the Conference to LuaConference, which is then responsible for broadcasting. The host application commands can be classified in a resource file associating lists of commands to labels. These lists are defined for the main application and for each plugin in a configuration file, and are then obtained by LuaConference and used during broadcasting for parsing and filtering commands. A mandatory class is "donot_broadcast", defining all the commands that cannot be broadcasted (registration, connection, exit, model transfer, creation of interaction tools, etc.).

This mechanism can be used for the logical subdivision of the command channel into separate channels, so that each can be subject to independent floor control. This

is currently done with three-dimensional annotations (freehand drawing, polygonal lines and arrow symbols) which, although similar in effect to other graphic interaction channels, are in fact generated by the application as commands only when the interactive creation of a primitive is complete.

The same type of command processing could be used for the specification of floors on objects and tools through the control of the commands that manipulate them.

All the commands broadcasted since the beginning of a conference are logged, to allow late joining (logged commands are sent by the Conference to new participants joining a session).

Streaming channels are subject to the same floor control mechanism but are handled separately (as described in section 4), due to the specific requirements they pose (in particular, audio and video streams are created in a peer-to-peer mode).

Control matrices

In a conference with *n* participants, for each channel, *n*x*n* matrices of boolean values specify the connectivity state of each participant. Three types of matrices are used: *Key*, *Intention* and *Connection*. Key matrices specify the rights of each participant to send (*K_out*) and to receive (*K_in*) information to one another, according to the participants' roles in a certain conference type. Intention matrices specifiy the instantaneous intentions of participants to send (*I_out*) and to receive (*I_in*) information (given that they have this right). Connection continuously expresses enabled connections, as the result of the compilation of Key and Intention matrices.

A connection is established when the send and receive Keys and Intentions of correlated participants are true. That is, for each position in the connection matrix,

$C_{ij} = (I\_out_{ij} \wedge K\_out_{ij}) \wedge (I\_in_{ji} \wedge K\_in_{ji})$

The specification of the rights of each *role* to send/receive events to/from each other *role*, for each collaboration type, is done as described below (section 3.5). The *n*x*n* matrices for the participants are then dynamically assembled by LuaConference as they enter and leave the session, based on their roles, and communicated to the host application by the Conference. Intentions are directly specified by the participants through the user interface controls (enabled or not according to the key values).

3.5 Collaboration specification

The control model adopted is inspired in COCA (Collaborative Objects Coordination Architecture) [15], but with some essential differences:

- as graphic performance is a main concern, our floor control mechanism changes the scheme defined in COCA based on the processing of all events through a virtual machine, for the control of input and output switches (tested by the event observers) at the host application by the plugin;

- instead of using logic-based rules (as in COCA) or declarative programs (as in DCWPL [3]) to define control policies – which are powerful and flexible but can become quite difficult to write and adapt by non-skilled programmers – we use some

elegant mechanisms provided by the Lua language (tables in particular) to create a simple object-oriented syntax for the specification of collaborations, which are interpreted by the LuaConference virtual machine at runtime.

This scheme allows the definition of the different types of *Collaborations* that a conference can assume. Collaborations contain *Roles* (associated to the participants) and communication *channels*, for which *floors* can be defined. Collaborations specify an extensible set of default functions to provide floor services, session management services and regulation services (specification of conference types and participant roles), which can be redefined for particular collaboration types.

Communication channel names are associated to the host application's channel indexes. For each channel of a collaboration type the rights of each role to send/receive to other roles are specified (Key matrices). By default all channels are open (true values in the matrix do not need to be specified, only blocked connections need to be specified with false values).

Intention matrices by default are also filled with true values. False entries are used to specify connections initially blocked (for instance, for audio and video channels, so that no more than the desired streams are created in the beginning of a videoconference).

The controllable channels (the ones for which floors can be established by the conference owner during the conference) are defined; all others will remain free (with no floor control, although participants may still opt not to send or receive over the channel). Default roles for the conference owner (the participant who started the conference) and for new participants are also defined.

The communication between LuaConference and the Conference class in the host application follows the conventional mechanisms for Lua.

As an example, let us show part of a simplified definition for a well steering session, with some very basic roles and polices. In this example, VR users cannot send or receive camera events to/from anybody, and operators do not receive or send video streams at the beginning of the session (but can receive the floor afterwards).

```
                                                    -- correlate channel names to application indexes
appl_channels = {cmd=0, cam=1, annot=2, cursor=3, audio=4, video=5, avatar=6}

WellSteering = Collaboration {                      -- define a collaboration type
    type = "wellsteering",
    channels = {"cmd", "cam", "annot", "cursor", "audio", "video"},     -- controllable channels
    floors = {"cmd", "cam"},                        -- channels initially  under floor control
    LeadGeoscientist = Role {type = "leader"},      -- define roles
    Geoscientist = Role {type = "geoscientist"},
    VrGeoscientist =  Role {type =  "vr"},
     OnSiteOperator = Role {type = "operator"},
    K_in = { ["cam"]= {["vr:all"]=false} },         -- define role-based matrices for
    K_out = { ["cam"]= {["all:vr"]=false} },        --      each channel (all other role pairs,
    I_in= { ["video"]= {["operator:all"]=false} },  --       for all channels, are true by default)
    I_out= { ["video"]= {["operator:all"]=false} },
    default_owner_role = "leader",                  -- default role assumed by the owner
    default_participant_role = "geoscientist",      -- default role for other participants
}
     -- Obs: K_out, K_in, I_out, I_in are true for all other role pairs, for all channels
```

Collaboration methods can then be redefined in Lua, in an object-oriented way (for instance, *ClassRoom:grant_floor*).

LuaConference interface

The interface provided by LuaConference to the host application implements the collaboration services (session management, floor control, etc.). Below is an extract of some of the main methods:

```
-- Session Management
function include_participant(participant, role)
function exclude_ participant (participant)

-- Floor
function grant_floor(ch, participant, requester)
function request_floor(ch, requester)
function get_floor_controller(ch)

-- Regulation
function set_type(type)                          -- set collaboration type
function get_type()
function get_collaboration_types()
function init_channels()
function set_role(participant, role)
function get_role(participant)
function get_collaboration_roles(collab_type)    -- return list of roles for the collaboration

// Broadcast commands
 function broadcast_command(command, sender)

 // Channels control
function get_control_matrix(M, ch)               -- assembles the nxn participants matrix for the
                                                 --    channel, given the role-based control matrix M
                                                 --    (one of K_out, K_in, I_out, I_in)
 // Late join
function late_join(participant)                  -- send all logged commands to the new participant
```

4 Videoconferencing

Audio and video communication channels are provided in NetGocad through the use of a custom multiplatform videoconferencing tool, CSVTool (Collaboration Supported by Video) [20]. This allows for a tight integration of this service, with no duplication of session management functionalities, and the direct control of audio and video streams according to the coordination policies defined.

The tool is integrated into the system through a separate plugin (gCSV) to avoid the establishment of a dependency. If the plugin is not present, the commands relative to videoconferencing are simply ignored.

CSVTool is implemented with JMF (Java Media Framework) [24] and can be used in two modes: integrated into a collaborative application or as standalone videoconferencing tool. Independently of the operation mode, it is divided into two modules, the server and the client. The server is responsible for the management of the participants and videoconferences, as described in the next section.

All the information exchanged among clients, except the audio and video streams, passes through the server. The most common messages are addition and removal of participants, which imply the creation or removal of streams. The server is not prone to traffic overburden because it does not receive the "heavy traffic" – the streams – which is transmitted directly between clients, in a peer-to-peer fashion.

The client/server communication is implemented in CORBA, and the communication among clients for stream transmission is made in RTP (Real- Time Protocol).

CSV operation with NetGocad

When a NetGocad participant starts a videoconference, the Conference launches a CSV server through its owner (Figure 4) and informs all Participants to start CSV clients, which automatically connect to the server.

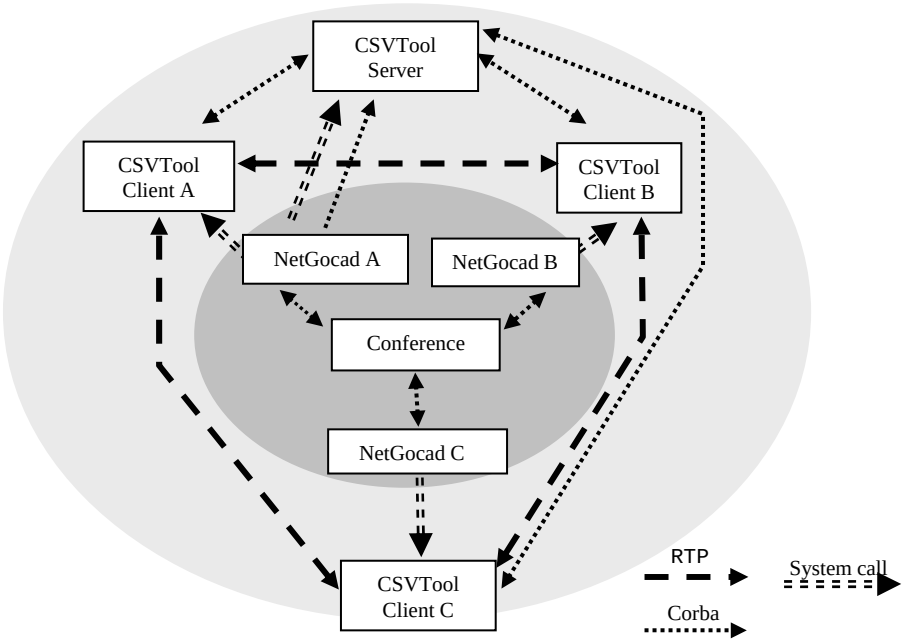

Fig. 4 CSV with NetGocad.

The CSV server is then responsible for managing the clients and audio and video streams. All the communication between the NetGocad Conference and the CSV Server is done in CORBA, through the conference owner.

Control matrices move between the CSV clients and the server, whenever necessary, by means of message exchange. Key and Intention matrices are sent by the NetGocad Conference to CSV, which uses them to set the connections' state.

The CSV tool updates the received Key matrices to indicate the availability of capture devices at each participant's machine, tested at start time. Conceptually, dis-

abling a connection due to a participant's role in a certain conference type is equivalent to switching off the device needed to send or to receive the related information.

One interesting feature provided is that the video stream sent by each participant can be switched from the image captured by the camera to the captured screen, so the tool can also be used for the remote display of a user's desktop. This is very useful for explanations about the operation of the application, or for consistency checks.

Complementary to the connection matrix, an External Intention matrix is computed in CSV to reflect connections that are not established because just one side decided not enable the stream. This information is used for the selection of the appropriate icon to represent the state of the connection at the user interface (Figure 5). As users have individual windows relative to one another, the intentions for each channel direction can be explicitly indicated. For the other channels, the setting of controls is done in a one-to-all basis (Figure 3).

5 Conclusion and future work

In this paper we presented an architecture that enables the transformation of a single-user application into a collaborative system by means of the integration of runtime plugins. We developed a control mechanism suited to the requirements of the application domain (collaborative three-dimensional geomodeling and visualization) and to the multimedia communication channels employed, based on the use of matrices associated to the channels, defining role-based rights and the dynamic intentions of the participants

The control scheme proposed, integrated into the application through a simple and flexible scripting language, generalizes the treatment of the different channels considered taking into account their specific performance demands. This scheme can be adapted to any application that follows some established design principles, particularly the use of the Command, Abstract Factory and Observer patterns.

We have concluded the integration of the tools and control mechanisms described, and are currently developing specific control policies for the scenarios discussed and others.

There are many issues that we would like to consider next. We want to assess the usability of the system under heterogeneous configurations, especially with the use of the VR plugin. This raises many usability issues that will require further development of the immersive interface. We also need to conduct evaluations using an adequate methodology tailored for groupware, an issue now being studied.

Other difficult CSCW concepts we want to explore in the future are the use of private views, concurrency control and asynchronous collaboration supported by the workflow engine used by the core application.

We expect that in the long term the integration of these features will create new collaboration tools which will certainly have a strategic importance in the industry.

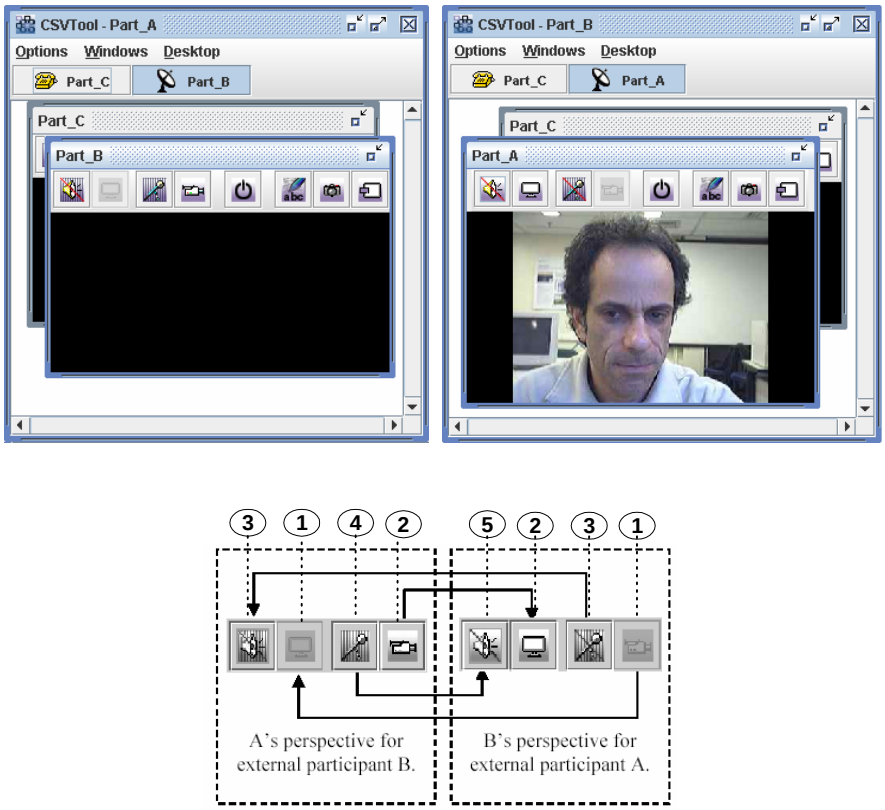

Fig 5. CSVTool interface. Each audio/video control button may assume five different configurations: 1.Disabled: the respective stream cannot be activated because the capture device is unavailable or is disabled due to the participant's role in the current conference type – the button becomes gray; 2. Active: the connection is active; 3. Off: both participants do not want to activate the stream –the button is dashed and hachured; 4. Waiting: the local participant wants to activate the streams, but the remote participant does not – the background of the button is hachured; 5. External: the remote participant wants to activate the streams, but the local participant does not – the button is dashed. Other buttons activate additional resources such as textual chat and snapshots.

References

1. Anupan, V. and Bajaj C.: SHASTRA: An Architecture for Development of Collaborative Applications. IEEE Multimedia, Vol. 1, Number 2 (1994) 39-49
2. Campos, J.L.E.: Real-Time Well Drilling Monitoring using gOcad. 22nd GOCAD Meeting (2002) web site: www.ensg.inpl-nancy.fr/GOCAD/meetings/Nancy2002/
3. Cortez, M. and Mishra, P.: DCWPL: A Programming Language for Describing Collaboration Work. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work (1996) 21-29.

4. Dewan, P.: Architectures for Collaborative Applications. In: Beaudouin-Lafon, M. (ed.): Computer Supported Co-operative Work, Vol. 7 of Trends in Software. John Wiley & Sons (1999) 169-193.
5. Dommel, H.P. and Garcia-Luna-Aceves, J.J.: Floor Control for Multimedia Conferencing and Collaboration. Multimedia Systems, Vol. 5, Number 1 (1997) 23-38
6. Edwards, W. K.: Polices and Roles in Collaborative Applications. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work (1996) 11-20.
7. Ehrlich, K.: Designing Groupware Applications. In: Beaudouin-Lafon, M. (ed.): Computer Supported Co-operative Work, Vol. 7, Trends in Software. John Wiley & Sons (1999) 1-28.
8. Ellis, C.A and Gibbs, S.J.: Concurrency Control in Groupware Systems, SIGMOD Conference, Vol. 18 (1989) 399-407
9. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley (1995)
10. GOCAD Research Consortium, web site: www.gocad.org
11. Gruchalla, K.: Immersive Well-Path Editing: Investigating the Added Value of Immersion, In: Proceedings of IEEE Virtual Reality (2004) 157-164
12. Ierusalimschy, R, Programming in Lua, Lua Org (2003)
13. Inside Reality, web site: www.oilfield.slb.com/content/services/software/virtual/index.asp
14. Leikness, S., Osvoll, I.: Success Factors in Troll Geosteering. Offshore Technology Conference (2005)
15. Li, D., Muntz, R.: Coca: Collaborative Objects Coordination Architecture. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work (1998) 178-188
16. Lua, web site: www.lua.org
17. Mallet, J.L., Geomodeling, Oxford University Press (2002)
18. Open Spirit, web site: www.openspirit.com
19. Pang A., Wittenbrink, C.M., Goodman T.: CSPray: A Collaborative Scientific Visualization Application. In: Proceedings Multimedia Computing and Networking, Vol. 2417 (1995) 317-326
20. Pozzer, C. et al :A Multi-user Videoconfernce-Based Collaboration Tool: Design and Implementation Issues, In: Proceedings of the 9th International Conference on CSCW in Design (2005) 547-552
21. Prakash, A.: Group Editors. In: Beaudouin-Lafon, M. (ed.): Computer Supported Cooperative Work, Vol. 7 of Trends in Software. John Wiley & Sons (1999) 103-133
22. Puder, A., Romer, K.: Mico: An Open Source CORBA Implementation", Morgan Kaufmann (2000)
23. Sun, C. and Chen, D.: Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems. ACM Transactions on Computer-Human Interaction ,Vol. 9, Issue (2002) 1-41
24. Sun Microsystem, Java Media Framework Home Page, web site: java.sun.com/products/java-media/jmf/
25. Tay, F.E.H., Roy, A: CyberCAD: A Collaborative Approach in 3D-CAD Technology in a Multimedia-Supported Environment. Computers in Industry, Vol. 52, Number 2 (2003) 127-145