Design of a Collaborative System

Minjun Wang
EECS Department, Syracuse University, U.S.A
Community Grid Laboratory, Indiana University, U.S.A
501 N Morton, Suite 222, Bloomington IN 47404, U.S.A
minwang@indiana.edu

Geoffrey C. Fox
Community Grid Laboratory, Computer Science Department, School of Informatics and Physics Department, Indiana University, U.S.A
gcf@indiana.edu

Abstract

In this paper we introduce the design of a new collaborative system in distance education, e-Learning and online conferencing – collaborative Impress applications. Impress is a presentation application in Open Office/Star Office; it has similar functionality as Microsoft PowerPoint.

Making Impress collaborative across computers is useful in e-Learning, web conference and distance education.

We have developed collaborative Impress applications which make use of the functions of Impress, and collaborate between the Master and Participating clients so that they share the same presentation slide display.

We use a common message broker, Narada Message Broker, as the underlying messaging environment to communicate messages between the clients in a session.

We realize the shared event idea in the collaboration. It uses event messages in the controlling of a presentation process. Compared to shared display, the short text messages save great bandwidth over the Internet. This realization is fast and efficient. Shared event model should be a new paradigm in collaboration.

Key Words

Collaborative System, Open Office, Shared Event.

1. Introduction

Open Office [1] is an open source office suite; it has similar functionality as Microsoft Office suite. Because it is free downloadable, and every one has an opportunity to develop and contribute to the resource using a broad range of programming languages and protocols, it has the features such as availability, usability, extensibility, popularity, and versatility. It is people's common property and wisdom.

Impress is a presentation application in Open Office/Star Office; it has similar functionality as Microsoft PowerPoint.

Making Impress collaborative across computers is useful in e-Learning, web conference and distance education.

We have developed collaborative Impress applications which make use of the functions of Open Office/Star Office, and collaborate between the Master and Participating clients so that they share the same presentation slide display.

We use a common message broker, Narada Message Broker [2], as the underlying messaging environment to communicate messages between the clients in a session.

We realize the shared event idea in collaboration. It uses event messages in the controlling of a presentation process. Compared to shared display, the short text messages save great bandwidth over the Internet. This realization is fast and efficient. Shared event model should be a new paradigm in collaboration.

This realization of collaborative Impress applications implies a complementary effort to and also a further research area in Open Office/Star Office.

2. Shared Event Model

A commonly used model in collaboration is Shared Display. In this model, some amount of screen image in a format (e.g. bitmap) is sent over the networks between the collaborating computers each time when the image of the screen is changed, either partially or totally.

The *Remote Desktop Connection* of Microsoft Windows XP and *VNC (Virtual Network Computing)* [3] are using this model.

It is appropriate in situations especially where the screen output is random, like online meeting, discussion, sharing data (text, graph, image, etc.,) or impromptu presentations using some software.

The disadvantages of Shared Display include:

- It consumes big bandwidth of the networks because of the image transferring. Thus it is relatively slow and the latency is big.
- The feeling of using it is not smooth. The waiting time depends on the amount of image

of a screen that needs to update. The worst case is when the image changes abruptly, say, the whole screen.

Let's examine a case of using presentation files in collaboration, such as Microsoft PowerPoint (.ppt) or Open Office/Star Office Impress (.sxi) files. Each slide in a presentation file is different and the content of a whole screen needs to update. In this case, the disadvantages of Shared Display behave to the worst.

To solve this problem, a new collaboration model is meant to take place and play an important role.

We believe a "lightweight" model, *Shared Event Model*, will work efficiently and gracefully.

The idea is to catch event messages in the driving side (let's call it *Master Client*) of collaboration during a presentation session, send them through common message brokers to the other accepting side (*Participating Clients* or *Participants*), and render the slide displays over there. The event messages are short text strings, as "Presentation Open", "Slide Change", etc.

The presentation files are deployed or downloaded to same directories on the hosts of both the Master and Participating clients before a session begins. This way, the collaborative applications can locate, open and navigate through them.

This model overcomes the disadvantages of Shared Display, and therefore has these advantages:

- It is fast and efficient, because the small text string messages greatly reduce the network traffic, and because it makes full use of the computing power of both sides.
- It gives consistent and smooth feelings of presentations. The size of the messages are small and approximately equal, so the time for transferring them should remain in a relatively constant range, just as mentioned in the paper "The Rule of the Millisecond."[4]

At the same time, because this model makes use of modern common message brokers, it shares the advantages of the brokers such as tolerance and quality of service. It also contributes to Peer-to-Peer Grids computing [5].

Collaboration like web conference, distance education and e-learning is a trend in today's information revolution, and the usage of presentation files accounts for a considerable proportion in all the visual aids. So, Shared Event model will play an important role and show its power.

3. Collaboration Structure

In the collaborative Impress applications, one type is of Master client, and the other is of Participating client, or Participant. Both of the client types cooperate with a common message broker, Narada Message Broker, as the underlying communication environment to communicate messages between the clients in a session.

The master client lectures and broadcasts its event messages to all participating clients.

The applications make use of the functions of Open Office/Star Office, and collaborate between the Master and Participant clients so that they share the same presentation slide display.

On the hosts of both the master and the participating clients, one should have installed Open Office/Star Office suite or just the Impress application, and should deploy or download beforehand copies of presentation files to be in a lecture. The Impress presentation files are deployed in consistent directories between the hosts of the master and the participating clients.

Thus, the structure makes it possible to collaborate between the clients by communicating only text messages. The master client captures events like "file opened", "slide changed" during a session of a lecture, translates them into text messages and sends the messages to the participating clients through Narada message broker. The participating clients then render the show of the lecture according to the directions of the received messages. This way, they work synchronously in collaboration.

This is illustrated in Figure 1.

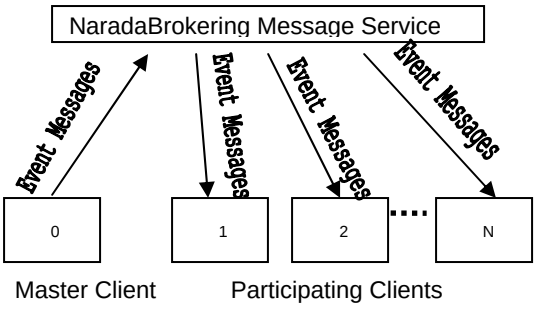

Figure 1. The collaboration between the Master client, Participating clients via the NaradaBrokering Message Service

4. New Concept for Collaboration

Some features in Open Office/Star Office are meant to be elegant for universal programming and the Web. They include Universal Network Object (UNO) technology, diverse programming environment, fine-grained Application Programming Interfaces (API), and Frame-Controller-Model Paradigm. They form a new concept for modern programming, and global Collaboration.

4.1 Universal Network Object

UNO is a component technology that is designed for universal programming and application. Components in UNO can interact with each other across programming

languages, component technologies, computer platforms, and networks.

UNO works with programming languages such as C++, Java, Java Script, Visual Basic, VBScript, and Delphi. UNO is the base component technology for Open Office/Star Office; it can also cooperate with other component technologies like Java Beans and Microsoft COM/DCOM. UNO is available on UNIX, Linux and Windows platforms, thus has the features of availability and popularity. UNO makes it possible that components are able to collaborate through networks. In a word, just as its name implies, UNO enables objects to function well across networks, and makes universal programming and application of objects a reality.

Through UNO technology, application programs connect to local or remote instances of Open Office/Star Office from C++, Java, or COM/DCOM. The programs then access the functionality of the instances using their APIs, control and automate the process, either sequentially or interactively. The purpose of UNO is to treat applications/components as reusable objects, which are accessible universally through the underlying infrastructure networks, as long as those objects are cooperative by providing programming interfaces, type libraries, etc.

The APIs of Open Office provide comprehensive specification of its programming features. In our collaborative Impress applications, both the Master and Participating clients connect to an instance of Impress application via UNO; access the functions in its API; capture and render the events thereof, respectively.

4.2 Diverse Programming Environment

Open Office offers Diverse Programming Environment. It enables people to develop codes in languages such as C++, Java, Java Script, Visual Basic, VBScript, and Delphi; on platforms such as UNIX, Linux and Windows. It has features of diversity, versatility, and popularity. Therefore, people can program in their most familiar languages and on their most convenient platforms. This is a factor for productivity and quality of software, and therefore a factor for contribution to human's common property.

People can add new functions to Open Office; integrate with Java Integrated Development Environment (IDE) through UNO components, and work with Office documents in Java Frames. For example, people can program new file filters, database drivers, linguistic extensions, or even complete groupware applications.

We can connect to a local or remote instance of Open Office from C++, Java, or COM/DCOM. Other than as powerful as the others, the extended Java API of Open Office is neat, efficient and secure. As an instance, it has very similar methods as COM/DCOM for connecting objects, like

```
queryInterface(),
addRef(),
deleteRef(),
…
```

People who are familiar with COM/DCOM technology will feel comfortable in coordinating Open Office objects in developing and using its functionality, yet take all the advantages of Java language, such as its features for the Web and Internet.

4.3 Fine-grained API

Open Office defines a comprehensive specification describing its programmable features. It is called Application Programming Interfaces (API). These Interfaces are fine-grained – each method (function) is for a sole and clear purpose; relative methods are grouped in a class; relative classes form a package, they call it a module; relative modules form a parent module, and parent modules can have their own parent along the tree structure, until a root module is reached, as in

com.sun.star.frame.XDesktop

From Software Engineering point of view, this design has at least the following strengths:

- Because of its fine-grained API, it is of high level of reuse, and therefore it will survive through time.
- Program can just integrate with small and necessary parts of the interfaces to fulfill its functionality, instead of having to include conglomerate blocks which contain lots of unnecessary functions. Thus fine-grained API makes program more effective and efficient.
- It makes the software highly extensible. Extensibility is of vital importance in modern software industry.
- It makes the software very easy to manage and maintain. The cost of software management and maintenance always is a big part in the life cycle of software engineering.

In both the master and participating clients of our collaborative Impress applications, the programs make use of the functions in the API, leverage the power of it, and collaborate with each other to share the screen simultaneously. As an example, we list the event listener interfaces and their corresponding event types we tried in our programs, in table 1.

Table 1. Event listener interfaces and corresponding event types

XPropertyChangeListener	PropertyChangeEvent
XSelectionChangeListener	EventObject
XFrameActionListener	FrameActionEvent
XKeyListener	KeyEvent
XMouseListener	MouseEvent
XMenuListener	MenuEvent
XWindowListener	WindowEvent
XContentEventListener	ContentEvent
XFocusListener	FocusEvent
XFormControllerListener	EventObject
XModeChangeListener	ModeChangeEvent
XChangeListener	EventObject
XContainerListener	ContainerEvent
XEventListener	EventObject
XTerminateListener	EventObject

4.4 Frame-Controller-Model Paradigm

The Model-View-Controller (MVC) design pattern [6] is popular and widely applied to interactive software development. Based on this pattern, Open Office adopts a new paradigm in its developing. It is called Frame-Controller-Model (FCM) Paradigm. We discuss them and point out the advantages of FCM paradigm in Open Office programming next.

In MVC, the Model is the application object; the View is its screen presentation; and the Controller is the encapsulation of a response strategy that defines the way a user interface responds to user input.

MVC changes the previous monolithic programming style in which the model, view and controller are undistinguishable and mingled together to be one unit or object; MVC decouples the model, view, controller from the megalithic lump to make software more flexible and reusable.

A model can have multiple views, and new views can be added. The appearances of the views reflect the state of the model. Between the view and model, there is a Subscribe/Notify mechanism. Each view subscribes and listens to the model. Whenever the values of the model have changed, it notifies the views about this. The views then access the model and update their screen appearance.

The controller is a response mechanism; it defines the way the user interface (the view) responds to user input. The controller object encapsulates the response strategy which represents an algorithm. The view associates with a controller instance at a time. It can have multiple controllers in store, and can add new controllers. A view can change a controller instance at run time, thus change the way it responds to user input dynamically. The controllers associated with the view may have different strategies or variant algorithms about the response of user interface to user input. The view switches to a different controller object either statically or dynamically, without changing its screen appearance.

The Frame-Controller-Model (FCM) paradigm in Open Office has some common properties and works similarly as MVC, but it has its own specialties and is more suitable especially for its Universal Networking Object (UNO) programming.

In FCM, the Model is the document object; it has document data and also methods that access the data. The methods can change the data directly without having to use a controller object. The controller is the screen interaction with the model; it observes the changes made to the model, and manages the presentation of the document. The frame is the controller-window linkage; it contains the controller for a model, and has knowledge about the window, but not the functionality of the window. That functionality is encapsulated in the underlying windows system – whatever platform it is. This decouples specific windows implementation from the frame, thus makes it possible to use a single frame implementation for different windows in Open Office. The specific windows work with the frame to make the screen presentation.

People can develop new models for new document types in Open Office without having to worry about the frame and the management of the underlying windows system. Each model can have multiple controllers associated with it. The controller depends on the model, and controls the manner of the presentation of a model. A controller can be replaced by another one without changing the model or frame. New controllers can be created for a model.

In programming, from a model object, we can use the method getCurrentController() of the API to get the controller object associated with this model; and from this controller, we can use the method getModel() to get the model object. Likewise, we can use the method getFrame() to get the frame object from the controller object; and we can use the method getController() to get the controller object from the frame object. From the frame object, we can even get the Container Window of the frame and Component Window of the Component using the methods getContainerWindow() and getComponentWindow(), respectively. Through those window objects, we can do jobs related to the management and control of the window system.

This is convenient and powerful. The FCM paradigm is just the right thing for Universal Network Object (UNO) programming.

5. The Client/Server Communication Bridge

The master client connects to Open Office/Star Office which serves as a server, listens to events fired there during a session, and sends the event messages to a message broker for broadcasting to participating clients for rendering the screen displays as that of the master

client. So, the master and participants are working synchronously in a session.

The client communicates information with the office server through TCP/IP socket. The office server listens to client TCP/IP connections using a connection URL as parameter, indicating hostname/IP address, port number, protocol, etc.

We launch the office server in listening mode by issuing the command line:

```
soffice -accept=socket, host=localhost, port=8100; urp;
StarOffice.ServiceManager
```

Here, the office server is running on the local host, listening to socket on port number 8100 for connection, using UNO remote protocol for communication. We make the client and server running on the same host for convenience, though they can run on different hosts in UNO programming environment.

As in other object oriented languages, objects are used in UNO programming to perform specific tasks. They are referred to as services in UNO context. A service manager is a factory of services, which creates services and other data used by the services. A component context consists of the service manager.

Both the master client and the office server have their own component context and service manager. The client creates services, or UNO objects, through the service manager in the client process, and the server creates UNO objects in the server process. Only the UNO objects created by the service managers can talk to each other across process boundaries.

The user control office files on the Office server, opening, loading, or accessing the data. In our case, the user controls the Impress presentation files on the Impress Office server, and the master client catch the events fired over there. In order to do this and work with the data located on the Office server, the client needs to establish a communication bridge between them and get the server's service manager.

We are describing the steps next, using explanations and snippets of codes in our programs.

First, the client creates a local UNO component context as follows:

```
XComponentContext xLocalContext =
com.sun.star.comp.helper.Bootstrap.createInitiComponentContext(null);
```

This local component context contains a service manager which is necessary to create services to talk to the server's component context. We can get this local service manager from it as follows:

```
XMultiComponentFactory xLocalServiceManager =
xLocalContext.getServiceManager();
```

The local service manager then creates a service called com.sun.star.bridge.UnoUrlResolver, which is an object to be used in the connection:

```
Object urlResolver =
xLocalServiceManager.createInstanceWithContext("com.sun.star.bridge.UnoUrlResolver", xLocalContext );
```

From this object, the interface of XUnoUrlResolver can be retrieved, which supplies methods to resolve the initial object with the server:

```
XUnoUrlResolver xUnoUrlResolver =
(XUnoUrlResolver) UnoRuntime.queryInterface(
XUnoUrlResolver.class, urlResolver );
```

The method resolve() of the interface is called to resolve the initial object with the server, using the same connection URL as that of the server when it is launched:

```
Object initialObject =
xUnoUrlResolver.resolve( "uno:socket, host=localhost,
port=8100; urp; StarOffice.ServiceManager" );
```

Now we have set up a bridge between the client and the server. The client makes use of this initial object associated with the server, accesses the data and controls the functions of the server as if they were its own. It needs to use the default context of the server by getting the Property Set and then the required property with it:

```
XPropertySet xPropertySet =
(XPropertySet)UnoRuntime.queryInterface(XPropertSet.class, initialObject);
Object context =
xPropertySet.getPropertyValue("DefaultContext");
```

Then it gets the server's component context:

```
XComponentContext xRemoteContext =
(XComponentContext)UnoRuntime.queryInterface(XComponentContext.class, context);
```

Finally, it gets the server's service manager:

```
XMultiComponentFactory xRemoteServiceManager =
xRemoteContext.getServiceManager();
```

Now, the client has a reference to the server's service manager. Thereafter, the client can use the reference to get the server's "Desktop" (com.sun.star.frame.Desktop) object and its interface, which is used to load, access documents (such as presentation files), and get the current one.

```
Object desktop =
xRemoteServiceManager.createInstanceWithContext(
"com.sun.star.frame.Desktop", xRemoteContext);
```

```
XDesktop xDesktop = (XDesktop)UnoRuntime.queryInterface(XDesktop.class, desktop);
```

With the XDesktop interface, the client can call its methods such as getCurrentFrame(), to get the server environment's Frame, Controller and Model (FCM), either directly or indirectly. With the FCM paradigm, as we discussed previously, the client can take control of the process of the server.

6. The Master Client

After the procedures described in the previous section, the Master client has set up the remote bridge and taken control of the programming features via FCM paradigm.

The master client gets the current frame, which in Impress corresponds to the current opened presentation file. It keeps testing for the current. If a change is detected, that means either a new presentation file is opened, or another opened one is switched to. The master client then gets the URL of this current presentation file through a method called getURL() in the interface of the Model.

The master client also registers listeners at the remote bridge to listen to events fired at the Office server. One of the registered listeners is the "Property Change Listener," which listens to property change events of an object. The client makes the listener listen to changes of "Current Page" of the current presentation file object.

```
PropertyChangeListener propertyChangeListener = new PropertyChangeListener();
xPropertySet.addPropertyChangeListener("CurrentPage", propertyChangeListener);
```

Whenever a presentation slide changes in the Impress server, the listener catches the event and notifies the event handler to do further processing. The event handler first gets the slide number using method getPropertyValue("Number") of XPropertySet interface. Then, it deals with the current slide number by adding appropriate XML (eXtensible Markup Language) [7] tags and its properties to address session information such as session identifier, topic title, source, destination and the like, as in

```
<event sessionID = "aSessionNumber" topic = "aTitile" to = "receiver" from = "sender"> a slide number </event>
```

So that each group of people in a session can send and receive messages correctly in a concurrent sessions support and public message broker environment such as NaradaBrokering Message Service.

The master client deals with the URL of the current presentation file in the same way, as in

```
<presentation sessionID = "aSessionNumber" topic = "aTitile" to = "receiver" from = "sender"> a URL of a presentation file </presentation>
```

As soon as such an XML message is generated, the master client sends it to the Narada Message Broker for broadcasting to all subscribed participating clients for rendering concurrently.

7. The Participating Clients

When the Narada message broker receives event messages from the Master client, it notifies the participating clients and broadcasts the messages to them.

Each participating client connects to, controls, and makes use of Office server. An instance of the Office application is installed on the host of the participating client, and the presentation files have been downloaded or deployed beforehand to the same directories as those on the host of the Master client. Each client processes the received event messages and renders the display simultaneously with the Master client.

To connect to the Office server, the participating client goes through the same procedures described previously, as the Master client does, to create a remote bridge, get the server's component context and service manager. It then gets control of the server's Frame, Controller and Model, and makes use of the FCM paradigm to use the server's functionality to control the rendering process.

When the client receives a message from the Narada message broker, it parses it and gets the different parts of information such as event type and its properties, or a URL of a presentation file. It then calls the functions of the server, such as loadComponentFromURL(), to open/switch to a presentation; it calls the method getDrawPages() of the XDrawPagesSupplier interface, the method getByIndex(index) of the XDrawPages interface, and the method select(xDrawPage) of the XSelectionSupplier interface, to navigate to a specific slide of an opened presentation, etc. The event type is the key to call different processing functions, and its associated properties are used in the functions to generate the correct presentation results. This rendering process is automation; the functions of the Office server are called under the instructions of the event messages.

Thus, the participating clients render the presentations being presented independently and simultaneously.

8. Future Work

We plan to improve our collaborative application system in the future by doing the following:

- Integrating the collaborative Impress applications with an Audio/Video system, such as Anabas Collaboration Environment [8]. This

is to bring multimedia into virtual classrooms and online conferencing.

- Making the collaborative system work with a session server, so that presentation/conference sessions can be registered with the session server, and subscribed by subscribers.

9. Conclusion

In this paper we have elaborated on the design, mechanism, technology and paradigm used in the Collaborative Impress applications, and the Narada message broker as the underlying communication system. We introduce the whole package as new distance education, e-Learning and online conferencing tools. Like anything else, it has limitations and advantages.

The limitations are: the Open Office/Star Office suite has to be installed on the hosts of both the master and the participating client, this may be difficult for hand-held devices, like PDAs (Personal Digital Assistants) [9]; the presentation files of the lectures have to be deployed or downloaded beforehand on the hosts of both of the clients.

However, with the advantages of the small text based message transferring, the robustness of Narada message broker, and the free downloadable feature and high availability of Open Office as its basis, the package of the Impress collaborative system will be suitable in situations like online conferencing, distance education, e-Learning and more. We believe it will contribute to those areas.

References

[1] OpenOffice.org
http://www.openoffice.org/

[2] G.C. Fox and S. Pallickara, The Narada event brokering system: Overview and extensions, *proceedings of 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*, Las Vegas, USA, 2002, 353-359. http://grids.ucs.indiana.edu/ptliupages/projects/NaradaBrokering/papers/naradaBrokeringBrokeringSystem.pdf

[3] Virtual Network Computing
http://www.uk.research.att.com/archive/vnc/

[4] Geoffrey Fox, The Rule of the Millisecond, for CISE Magazine March/April 2004
http://grids.ucs.indiana.edu/ptliupages/publications/ciseja no4.pdf

[5] G.C. Fox, H. Bulut, K. Kim, S. Ko, S. Lee, S. Oh, S. Pallickara, X. Qiu, A. Uyar, M. Wang, W. Wu, Collaborative web services and peer-to-peer Grids, *Proceedings of 2003 Collaborative Technologies Symposium (CTS'03)*, Orlando, USA, 2003.
http://grids.ucs.indiana.edu/ptliupages/publications/foxwmc03keynote.pdf

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (201 W. 103rd Street, Indianapolis, IN 46290: Pearson Education Corporate Sales Division, 2002).

[7] Extensible Markup Language (XML)
http://www.w3.org/XML/

[8] Anabas Collaboration Environment.
http://www.anabas.com

[9] S. Lee, G.C. Fox, S. Ko, M. Wang, X. Qiu, Ubiquitous access for collaborative information system using SVG, *Proceedings of SVG open conference*, Zurich, Switzerland, 2002.
http://grids.ucs.indiana.edu/ptliupages/projects/carousel/pagers/draft.pdf