Thin Client Collaboration Web Services

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

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

Marlon Pierce
Community Grid Laboratory, Indiana University, U.S.A 501 N Morton, Suite 224, Bloomington IN 47404
mpierce@cs.indiana.edu

Abstract

In this paper we introduce some collaboration applications, and the needs in changing them to Web Services. We propose and describe the idea of Thin Client Collaboration Web Services, and explore some potential scenarios of this idea in which it shows its merit and the freedom resulted in collaboration.

Such a Web Service has two sets of ports: User-facing Input/Output ports and Resource-facing Input/Output ports. The user-facing I/O contacts a Web Service viewer, and the resource-facing I/O contacts a collaboration application. Hence, the role of the Web Service is to transcode in both directions between the two sets of ports with respect to displays and events, so that the user accesses the Web Service viewer as if the collaboration application itself.

We use three collaborative applications as resources, and projects in SVG as the demonstration of our initial effort in the implementation of a General Thin Client Collaboration Web Service.

1. Introduction

There has been a lot of software developed for collaboration over the Internet. Application areas include online conferencing, distance education, e-Science, e-Business, etc. Applications already in industry, such as WebEx [1], Tango and Anabas [2], have proved to be useful, efficient, and beneficial. It means, among other things, distance and location is no longer a barrier or restriction, and time, resources and money are saved.

The Web Service Architecture is designed to promote software's usefulness, interoperability, availability, extensibility, etc. If the collaboration software is made to be Web Service (WS), totally or partially, it will be more powerful and benefit everyone – different research groups, institutions, organizations, and even ordinary users.

In this paper, we describe some possible ways of making such software to be Web Service, propose and focus on discussing the idea of *Thin Client Collaboration Web Services*, and explore some potential scenarios of this idea in which it shows its merit on leveraging the merit of the Web Service Architecture, and on more freedom in collaboration resulted from it.

The idea of *Thin Client Collaboration Web Services* is that, instead of packaging the whole package of a collaboration software application as Web Service, it separates the native interface from the rest of the software, correlates the native interface (in whatever format and language) to a web service friendly user interface (such as in SVG format and Java language), including its screen display and events originated from the triggering of the widgets inside the display, and let the user access the result user interface as if it were the native interface, relying on the fact that the Web Service has made them one-to-one correspondence functionally, both on the corresponding parts of the displays, and on the corresponding events of the widgets.

Extensible Markup Language (XML) [3] is an indispensable description language for Web Service, and Scalable Vector Graphics (SVG) [4] is a subset of XML. SVG has advantages in representing screen displays because it is vector oriented. Other than all kinds of SVG viewers, Web browsers like Internet Explorer can render SVG files directly inside their windows. It seems that SVG format is the right one we should choose, and Java is the right language for the purpose, because it has been designed and developed with the Web in mind. So hereafter, when we refer to

WS user interface, we mean that it is in SVG format and Java language, though others could be potential alternatives.

The Web Service in question has two sets of ports: User-facing Input/Output ports and Resource-facing Input/Output ports [5, 6]. The user interface discussed above corresponds to user-facing I/O, and the native interface corresponds to resource-facing I/O. Hence, the role of the Web Service is to transcode in both directions between the two sets of ports with respect to displays and events, so that the user accesses the user-facing I/O as if he/she were accessing the resource – the collaboration software application – itself.

We use our three collaborative projects – collaborative PowerPoint [7], collaborative OpenOffice [8] and collaborative IDL ReviewPlus [9] – as examples of resources. Each of these projects consists of a Master client and a Participant client. In execution, there can be multiple instantiations of the Participant clients. In a collaboration session, the Master captures events and sends out event messages through a message broker to the Participants for rendering the same result displays. We use our projects in SVG as the demonstration of our initial effort in the implementation of the idea of *Thin Client Collaboration Web Services*. We describe briefly the design and implementing issues of the SVG projects in this paper, give some results, and point out the future work.

2. Problems

While it is possible to package the whole package of the collaboration software application as Web Service, the advantage of doing so really depends on situations and there are problems with some situations, as we discuss below.

If the package is small in size and its deployment structure is not complicated, it is good to do so.

If the package is big in size, it would be difficult to do so. Take our collaborative OpenOffice project as an example. Our developed code for collaboration is not big, but it is based on the whole underlying OpenOffice source code, making use of its fundamental functions to achieve the collaboration. Even though it is open source, the source code of OpenOffice consists of millions of lines of code and it has been developed by groups of talented people years of hard work. Theoretically, if we were to package this project as Web Service, we have to package the underlying basis, too; or at least part of it, if we were lucky enough to know which necessary parts to include and those parts are absolutely not calling or referencing the code of the rest all the time. How hard would that be, if it is not impossible?

If the package is related to proprietary software, it would be more difficult. Take our collaborative PowerPoint project as an example. Our developed code for collaboration is not big, but it is based on the functionality of the underlying Microsoft PowerPoint application, making use of its functionality to achieve the collaboration. It is proprietary and it is big, too. Again, theoretically, if we were to package this project as Web Service, we have to package the functionality of this proprietary product as well. What is the possibility of success in this matter?

If the package itself is complicated in architecture and deployment, possibly involving fire walls, it would be at least difficult. Take our collaborative ReviewPlus project as an example. ReviewPlus [10] itself is big, and it calls functions from another independent big package called MdsPlus [11], and also others. More complicated yet, ReviewPlus contacts with other servers, such as the event server, and exchange information with them. It is highly possible that they are behind different fire walls, as the suggestion of ReviewPlus itself, because it is deployed behind at least one level of fire walls. Once again, theoretically, if we were to package this project as Web Service, we have to deal with all the complexity as well. How much effort would that cost and how many people and groups would be involved, given that everybody is willing to cooperate?

What is more, if the package was developed in a popular language like Java, C++, or C#, where accompanying tools for Web Service have been developed, it is easier for the job; but what if the package was developed in a language that has not had such additional tools yet, like IDL?

On all accounts, it is desirable to find another way for Web Service that avoids the difficulty and complexity, harnesses the resource, and accesses the user interface. This leads to our next description of the idea of *Thin Client Collaboration Web Services*.

3. Thin Client Collaboration Web Services

As we can see, collaboration software applications are developed on different platforms, using different models, paradigms, architectures, and methodologies, and in different programming languages. One common feature is that they usually have rich user interfaces where users can access input/output information and achieve collaboration between peers. We refer to these interfaces as *native interfaces*, and the applications as *resources* in this text.

These resources are usually developed in multi-tiers architecture, say three-tiers: back end tier, middle tier, and front end tier, with databases on the back end and native interfaces on the front.

The idea of *Thin Client Collaboration Web Services* is that, instead of packaging the whole package of a collaboration software application as Web Service, it separates the native interface from the rest of the software; correlates the native interface (in whatever format and language) to a web service natural and friendly user interface (such as in SVG format and Java language), including its screen display and events originated from the triggering of the widgets inside the display; and lets the user access the result user interface as if it were the native interface, relying on the fact that the Web Service has made them one-to-one correspondence functionally, both on the corresponding parts of the displays, and on the corresponding events of the widgets.

The structure of the Web Service has two sets of ports: User-facing Input/Output ports and Resource-facing Input/Output ports [5, 6], as shown in Figure 1.

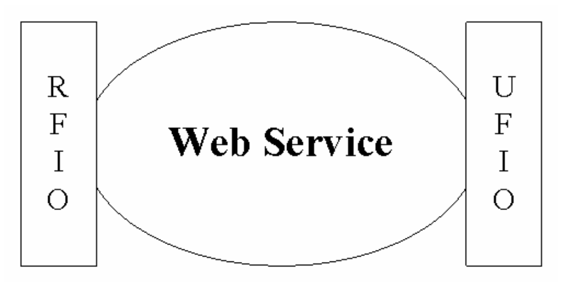


Figure 1. The structure of a web service with user-facing input/output ports and resource-facing input/output ports

The WS user interface discussed above corresponds to user-facing I/O, and the native interface corresponds to resource-facing I/O. Hence, the role of the Web Service in question is to transcode in both directions between the two sets of ports with respect to displays and events, so that the user accesses the user-facing I/O as if he/she were accessing the resource – the collaboration software application – itself.

Thus, the Web Service doesn't have to deal with all the difficulty and complexity of the resource; it just has to contact with the native interface of the resource and do some transcoding between the two sets of I/O ports, and lets the rest be the encapsulation of the resource.

On one direction, the Web Service takes the output display of the resource to the input port (I) of the resource-facing I/O, translates the information in the native interface to the equivalence in SVG format, and directs the result to the output port (O) of the user-facing I/O, where the SVG viewers, Web browsers can render the SVG file directly inside their windows for the user to view.

On the other direction, the Web Service takes the event message from the SVG viewer/Web browser, which is resulted from the triggering of the widget event by the interaction of the user, to the input port (I) of the user-facing I/O, translates the event message to the equivalence of the event/event structure of the native interface of the resource, and directs the result to the output port (O) of resource-facing I/O, where the resource gets the event message and automates the execution under the instructions in the message.

We use our three collaborative projects – collaborative PowerPoint, collaborative OpenOffice and collaborative IDL ReviewPlus – as examples of resources. We also have done projects in SVG: a converter which converts HyperText Markup Language (HTML) file to SVG file, and a converter which converts files in Windows Metafile Format (WMF) [12] to SVG format. Here is a clue of implication: OpenOffice can save its presentation file (*.sxi) as PowerPoint file (*.ppt), and PowerPoint can save its presentation file (*.ppt) as HTML file, WMF file, and so on. The indication is that, our trial in the SVG projects is the initial effort of the implementation of the idea of *Thin Client Collaboration Web Services*.

4. Thin Client Web Services in Collaboration

In this section, we give some collaboration scenarios in which the thin client web services play their roles and show their potentials. We use our three collaborative projects as resources. Each has a master client that generates event message, and at least one participant client that consumes the event message. The master and participant(s) communicate with and cooperate on the event message. The thin client web services contact only with the native interfaces of the clients and access/control them in collaboration. The scenarios bring interoperability, flexibility and diversity to collaboration and can be potential solutions for many technical problems regarding to synchronous/asynchronous issues in collaboration.

4.1. Scenario 1

All the resources of our collaborative applications are developed in this manner: The Master client controls the process of a session of collaboration, captures events and sends event messages to all Participant clients through NaradaBrokering (NB) event broker [13, 14]; the Participants do not interfere with the process, they just receive the event messages and render the same output displays as the Master under the instructions in the messages. This way, the states of all the clients keep the same at every event so that collaboration is achieved.

This scenario follows the manner of the resources: only one instance of the Web Service is hooked up with the Master client of the resource, and the controller or lecturer is controlling the process through a WS viewer; at least one instance of the Web Service is hooked up with each Participant client, and the audiences or students are viewing the displays via the WS viewers. This is depicted in Figure 2.

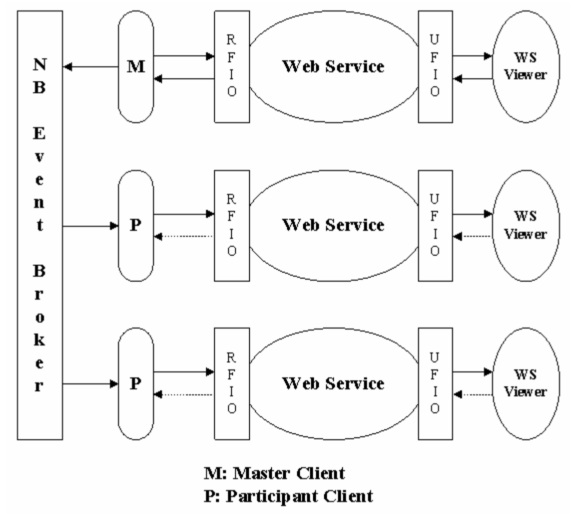


Figure 2. Instances of a web service in collaboration, with only one instance for the master client and at least one instance for each of the participant clients

In the Figure, two participant clients are shown. In reality, it can be any number: one, two, or many, as many as the NB event broker can support; theoretically, the number is unlimited. Also in the Figure, even though only one instance of the web service is shown with each participant client, it could be multiple instances for each. What is more, the clients of the resource and the web service need not be deployed on the same location.

The WS Viewers in the picture can be any SVG rendering tools: SVG viewers, Web browsers, Personal Digital Assistants (PDA), or other mobile devices [15, 16], because of one factor – the output of the web service to the viewers is in SVG format. This makes universal access to resources possible, and so for universal collaborations.

Since the Participant client is designed to be passive and is not allowed any input to it other than the event message from the Master, the input of event from the audiences or students along the direction through the Web Service has no effect on the Participant client, as indicated by the dotted arrows in the Figure.

We can originally design such a Web Service for a sole resource – collaborative PowerPoint, collaborative OpenOffice, collaborative IDL ReviewPlus, or any other collaboration application. Later on we can aggregate/relate these element Web Services to a *general Web Service*, which will dispatch function calls to the elements on conditions, so that the *general Web Service* can be used for all these resources.

4.2. Scenario 2

If we hook up two or more instances of the Web Service to the Master client in Figure 2, the collaboration pattern changes to a more dynamic and democratic environment for presentation, as shown in Figure 3.

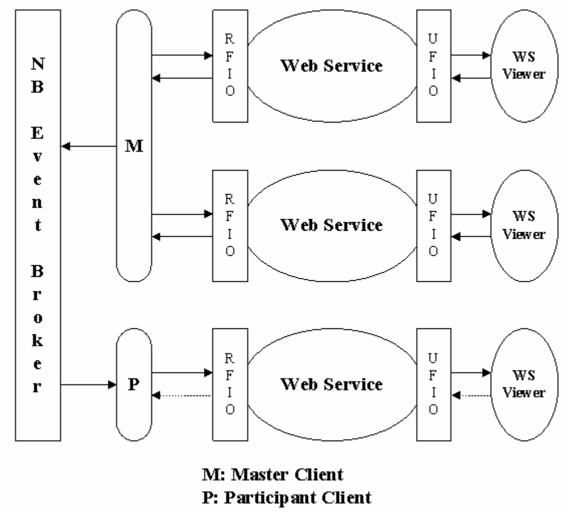


Figure 3. Instances of a web service in collaboration, with two or more instances for the master client and at least one instance for the participant client

This scenario allows two or more speakers/lecturers to jointly present a presentation/lecture to the audiences/students, with everybody possibly on different locations. This scenario is also suitable for a study group discussing some content on the web, with the rest as silent observers.

It works like this: the same native interface of the output display of the Master client feeds to every input port of the resource-facing I/O of the instance of the Web Service hooked up with the Master client; each instance then does the transcoding job and supplies the result SVG file to its associated WS viewer through the output port of the user-facing I/O of the instance;

each WS viewer then renders the same output display as that of the Master client.

Each instance of the Web Service with the Master client takes the event message from the WS viewer, which is resulted from the triggering of the widget event by the interaction of the user, to the input port (I) of the user-facing I/O, translates the event message to the equivalence of the event/event structure of the native interface of the resource, and directs the result to the output port (O) of the resource-facing I/O, where the Master client gets the event message and automates the execution under the instructions in the message.

Thus, whenever the display of the Master client changes, it reflects to each of the instances; every instance sends some event messages to the Master during a session, and the union of all the event messages reflects the sequence and action of the joint presentation. To the Master client, it is just like one person is controlling the process of the session.

At each step of the collaboration, the Master and Participant clients share the same state due to the communication of the event message. All the instances of the Web service hooked up with the Master client share the same state too, in the form of presenting the same resulting SVG file to the viewers.

4.3. Scenario 3

If we hook up multiple instances of the Web Service to the Master client only in Figure 3, it is reduced to Figure 4.

This scenario is a special case of scenario 2. It is adequate for a study/research group discussing some content of their own interest on the web, with everyone possibly on different locations.

The working mechanism is the same as scenario 2. Here, it just makes use of the Master client of the resource and lets the instances of the Web Service share the output displays of the client and control it jointly through events.

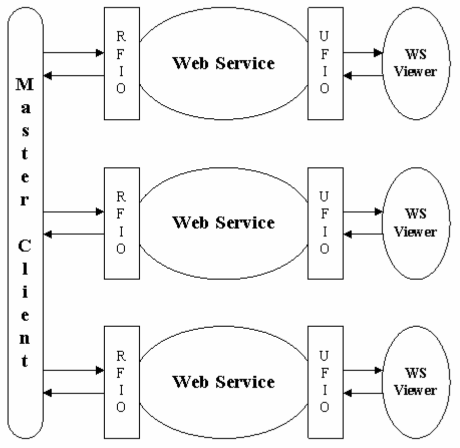


Figure 4. Instances of a web service in collaboration, with multiple instances hooked up with the master client only

4.4. Scenario 4

Diverse visual aids in presentation can convey more information, make it more effective, give deep impression and enlighten the soul. Therefore, it is a good use to bring diverse resources and Web Services together in a presentation, as shown in Figure 5.

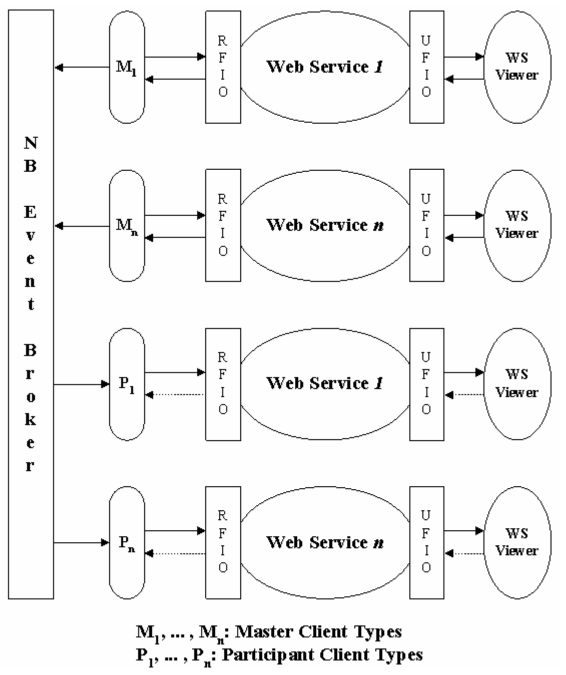


Figure 5. Instances of diverse web services in collaboration, with each instance type hooked up with its corresponding resource type

In this picture, the pair (M_i, P_i) represents a resource type, with M_i be the Master client of the type, and P_i be the Participant client; instances of Web Service i hook up with this type of clients.

Let us conjure up an example to illustrate. Suppose in a presentation we use the resource of collaborative PowerPoint applications (M_1, P_1) to do the main course of the presentation, use the resource of collaborative OpenOffice (M_2, P_2) to give additional information corresponding to each slide in the previous, such as references, episodes, exhaustive details, etc., and use the resource of collaborative IDL ReviewPlus (M_3, P_3) to supply scientific and engineering graphs, charts and images necessary to each slide, in 2D or 3D. This sounds more interesting and attractive.

During a presentation, the speaker controls at will three WS viewers corresponding to three instances of Web Service 1, 2, and 3 which are hooked up with the resource type 1, 2, and 3, respectively. Accordingly, the audiences use three WS viewers to read the output displays of each. In a simple case, it could be three instantiations of a Web browser on the screen of a monitor, for everyone.

As always, at each collaboration step, the states of (M_i, P_i) would be the same on the current event message. Any two types of resources – (M_i, P_i) and (M_j, P_j) – would not interfere with each other, if we put extra identifying information at the head of each event message, e.g. "ppt", "office" or "idl", because after checking this identifying information, if it is not supposed for a Participant client type, it would just ignore it. Thus, the states of one resource type are independent of the other.

We can originally design each Web Service for a sole resource – collaborative PowerPoint, collaborative OpenOffice, collaborative IDL ReviewPlus, or any other collaboration application. Later on we can aggregate/relate these element Web Services to a *general Collaboration Web Service*, which will dispatch function calls to the elements on conditions, so that the *general Collaboration Web Service* can be used for all these resources, and all the WS viewers, too. This makes things clearer and saves efforts in finding and binding Web Services.

4.5. Potential Problems and Solutions

By now we have focused on describing the main features of the scenarios. We can foresee some potential problems in them, and we have accordingly planed the solutions for them, as follows:

Problem 1: On the native interface (display) of the Master client of a resource, after one event is triggered (e.g. a click on a button), and before the output comes out and the display is updated, the cursor on the display window is usually changed to a waiting sandbox to prevent any further input; or in some other cases, even if the sandbox cursor is not shown, any further input is just ignored.

What if additional events are issued on the WS viewer in between?

For instance, if two consecutive button clicks – one on button 1 and one on button 2 which becomes the additional event – happened on the current display on the WS viewer, when these two events get transcoded through the Web Service and get to the native interface, this could cause trouble: because there is a time interval between the two events, after the first one is executed by the Master and its native display is updated, the updated display may be a totally different one that contains no button at all, then the event of button 2 makes no sense at all to the new display.

Solution: Consider the behavior of the native display of the resource, we can add a similar mechanism to the Web Service that changes the cursor on the WS viewer to a waiting sandbox after an input is put on the input port of the user-facing I/O, and keeps that status until the corresponding output is coming back from the resource and put on the output port of the user-facing I/O. This prevents additional events from happening before the current one is complete from the view of the overall system, by making those attempts impossible.

Problem 2: In scenario 2 and 3, where multiple speakers jointly present a presentation or discuss subjects within a group, if two or more speakers issue events simultaneously or very closely, that will cause trouble as in Problem 1, since to the Master client, the result is just like one speaker is controlling. Worse yet, it could cause unpleasant results between the speakers; just imagine that, before a speaker finishes his/her part, another issues an event intentionally or accidentally and changes the display; the rest, though innocent, also become potential "suspects" to the first one. It is at least interrupting.

Solution: As in some audio/video conferencing systems, we can add a resource competition mechanism to the Web Service; this resource is in the form of an icon of a microphone. When a speaker is in possess of the icon, the rest are disabled, that is, the events they issued are ignored, only the events from the current speaker can get through the Web Service and reach the Master client. After the speaker finishes his/her part, he/she releases the icon, and all the speakers get a chance to compete to get the icon and speak next. Of course, for the current speaker, problem 1 and its solution apply.

Problem 3: The NB event broker is supposed to be a common message broker deployed on Grid for public use. It may be the case that multiple conferencing sessions are going on using NB as the underlying communication media, with overlaps in time with one another.

How to avoid event messages from different sessions interfering with each other?

Solution: First, conference sessions should be advertised on a session management service, such as GlobalMMCS [17], as to schedules, titles, and unique session numbers. Secondly, we can arrange the Web Service to get the session number for a session and let it inform the native clients of a resource so that they agree on that session number in recognizing the supposed event messages. Any alien session number in the message will cause the message to be ignored. This session number should be added to the head of each event message. This way, each conferencing session is sifting their own messages and working on them clearly in the open Grid environment.

Problem 4: In scenario 4, how to avoid event messages from any two different types of resources – (M_i, P_i) and (M_j, P_j) – interfering with each other?

Solution: As always, at each collaboration step, the states of (M_i, P_i) would be the same on the current event message. Any two types of resources – (M_i, P_i) and (M_j, P_j) – would not interfere with each other's execution due to the mixed event messages from NB, if we make the Master client M_i put extra identifying information for its type at the beginning of each event message (after the session number), e.g. "ppt", "office" or "idl": because after checking this identifying information, if it is not supposed for the corresponding Participant client type P_i, it would just be ignored by P_i. Thus, the states of one resource type are independent of the other.

5. Deployment and Usage of Collaboration Web Services

Web Services along with Peer-to-Peer Grids play important roles in collaboration. Web Services enable developers and users to integrate functionality across businesses and organizations.

Suppose that we have developed our general Collaboration Web Service.

The information of this Web Service, such as its Universal Resource Identifier (URI) endpoint, its exposed methods, etc., is described in the Web Service Description Language (WSDL) file, and the Web Service is deployed and published to a Service Broker with this file. The users or applications can find this Web Service using Universal Discovery, Deployment and Integration (UDDI), and then bind to it and use it via the internet [18], as in Figure 6.

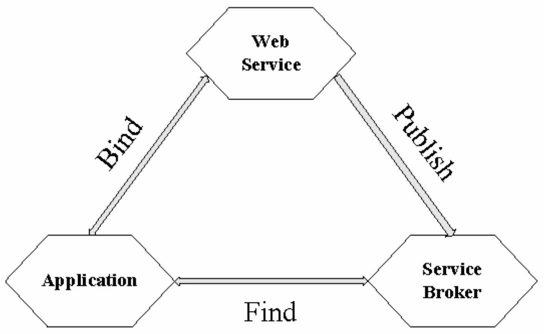


Figure 6. Web service published to service broker and used later in application

On one end, a client of a resource (collaborative applications) finds and binds to the general Collaboration Web Service and cooperates with it through its resource-facing I/O; on the other end, a WS viewer performs the same procedure to bind to the general Collaboration Web Service and cooperates with it through its user-facing I/O.

More specifically, as described in the scenarios before, the speaker's WS viewer is bound to the instance of the general Collaboration Web Service that has the Master client of a resource bound to it, and each audience's WS viewer is bound to the instance of the general Collaboration Web Service that has the Participant client of the corresponding resource bound to it. As always, the Master client and the Participant client(s) of each resource type collaborate on event messages via the underlying communication of the NB event broker.

Thus, collaboration is achieved through the general Collaboration Web Service.

6. Initial Effort in the Implementation of a General Collaboration Web Service

In this section we first describe our initial effort in the implementation of a general Collaboration Web Service: a converter which converts HTML file to SVG file, and a converter which converts files in WMF format to SVG format. This effort is in the direction we have described earlier from the resource to the viewer, or from the resource-facing I/O to the user-facing I/O; it is motivated by the clue of implication: OpenOffice can save its presentation file (*.sxi) as PowerPoint file (*.ppt), and PowerPoint can save its presentation file (*.ppt) as HTML file, WMF file, and so on. The indication is that, our trial in the SVG projects is the initial effort in the implementation of a General Collaboration Web Service.

Secondly, we describe part of our future work: the transcoding of events from the viewer to the resource, or from the user-facing I/O to the resource-facing I/O.

6.1. Converting HTML File to SVG File

We have done a project in SVG: a converter which converts HTML file to SVG file.

Basically, the converter consists of two functional units: an HTML parser and an SVG converter. The HTML parser takes an HTML file as input, parses it to get and categorize the HTML tags and their associated properties. The SVG converter works on the HTML tags and properties and converts the information to the equivalent SVG representation. In most cases, it converts the information contained in a pair of HTML tags to the resulting SVG equivalence which is contained in a pair of SVG tags.

We are not attempting to give much detail about the programming and implementing of the project, but give some results which are enough to indicate the effort.

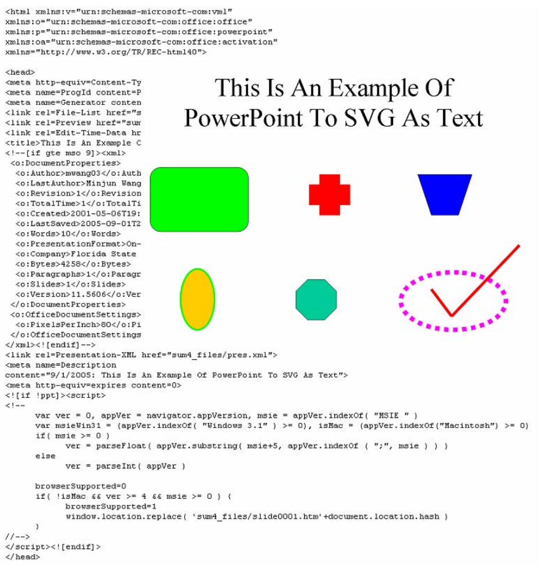
In Figure 7, we give the display of the rendering of an HTML file in Internet Explorer web browser, at the same time we give the source of the HTML file in the background of the figure.

In Figure 8, we give the display of the rendering of the converted SVG file from the HTML file in Internet Explorer web browser, at the same time we give the source of the SVG file in the background of the figure.

6.2. Converting WMF File to SVG File

We have done another project in SVG: a converter which converts WMF file to SVG file.

As the former one, the converter also consists of two functional units: a WMF parser and an SVG converter. The WMF parser takes a WMF file as input, parses it to get and categorize the codes for its element types (e.g. a line) and their associated properties. The SVG converter works on the codes for types and properties and converts the information to the equivalent SVG representation, with each piece of information contained in a pair of SVG tags.

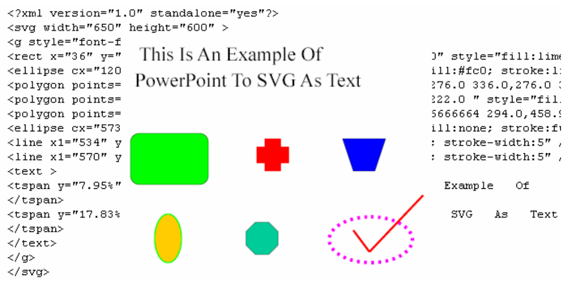


```
<html xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:p="urn:schemas-microsoft-com:office:powerpoint"
xmlns:oa="urn:schemas-microsoft-com:office:activation"
xmlns="http://www.w3.org/TR/REC-html40">

<head>
<meta http-equiv=Content-Ty
<meta name=ProgId content=P
<meta name=Generator conten
<link rel=File-List href="s
<link rel=Preview href="sum
<link rel=Edit-Time-Data hr
<title>This Is An Example C
<!--[if gte mso 9]><xml>
 <o:DocumentProperties>
  <o:Author>mwang03</o:Auth
  <o:LastAuthor>Minjun Wang
  <o:Revision>1</o:Revision
  <o:TotalTime>1</o:TotalTi
  <o:Created>2001-05-06T19:
  <o:LastSaved>2005-09-01T2
  <o:Words>10</o:Words>
  <o:PresentationFormat>On-
  <o:Company>Florida State
  <o:Bytes>4258</o:Bytes>
  <o:Paragraphs>1</o:Paragr
  <o:Slides>1</o:Slides>
  <o:Version>11.5606</o:Ver
 </o:DocumentProperties>
 <o:OfficeDocumentSettings>
  <o:PixelsPerInch>80</o:Pi
 </o:OfficeDocumentSettings
</xml><![endif]-->
<link rel=Presentation-XML href="sum4_files/pres.xml">
<meta name=Description
content="9/1/2005: This Is An Example Of PowerPoint To SVG As Text">
<meta http-equiv=expires content=0>
<![if !ppt]><script>
<!--
      var ver = 0, appVer = navigator.appVersion, msie = appVer.indexOf( "MSIE " )
      var msieWin31 = (appVer.indexOf( "Windows 3.1" ) >= 0), isMac = (appVer.indexOf("Macintosh") >= 0)
      if( msie >= 0 )
            ver = parseFloat( appVer.substring( msie+5, appVer.indexOf ( ";", msie ) ) )
      else
            ver = parseInt( appVer )

      browserSupported=0
      if( !isMac && ver >= 4 && msie >= 0 ) {
            browserSupported=1
            window.location.replace( 'sum4_files/slide0001.htm'+document.location.hash )
      }
//-->
</script><![endif]>
</head>
```

Figure 7. Rendering of an HTML file with the source of the HTML file showing in the background



```
<?xml version="1.0" standalone="yes"?>
<svg width="650" height="600" >
<g style="font-f
<rect x="36" y="                                 )" style="fill:lim
<ellipse cx="120                                 ill:#fc0; stroke:l
<polygon points=                                 276.0 336.0,276.0
<polygon points=                                 222.0 " style="fil
<polygon points=                                 5666664 294.0,458.
<ellipse cx="573                                 ill:none; stroke:f
<line x1="534" y                                 : stroke-width:5" ,
<line x1="570" y                                 : stroke-width:5" ,
<text >
<tspan y="7.95%"                                   Example   Of
</tspan>
<tspan y="17.83%                                    SVG    As    Text
</tspan>
</text>
</g>
</svg>
```

Figure 8. Rendering of the converted SVG file from the HTML file, with the source of the SVG file showing in the background

The difference here is, WMF file is represented in hexadecimal codes for both its element types and properties. Each element in the file is defined by a type code followed by codes for its properties, such as number of bytes, coordinates, colors, etc. The meanings of all the codes are defined in WMF file format description [19]. Only with that explanation for each byte of the codes can each code be meaningful to devoted people. Otherwise, those arcane codes are just gibberish; only part of the range of the codes falls into the printable ASCII representation, and event that part when printed is just random gibberish, as we show in the background of some of the following figures.

We are not attempting to give much detail about the programming and implementing of the project, but give some results which are enough to indicate the effort.

In Figure 9 and 11, we give the displays of the rendering of two WMF files in Windows Picture and Fax Viewer; at the same time we give the sources of the WMF files in the backgrounds of the figures, which are only the parts for the printable characters and signs of those arcane codes.

In Figure 10 and 12, we give the displays of the rendering of the converted SVG files from the WMF files in Internet Explorer web browser, at the same time we give the sources of the SVG files in the backgrounds of the figures.

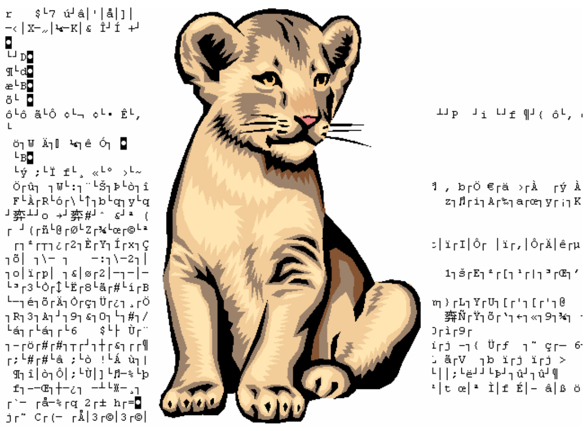
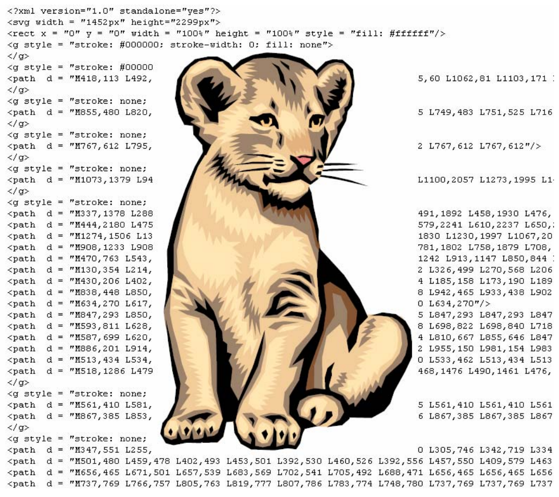


Figure 9. Rendering of a WMF file with the printable source of the WMF file showing in the background
[Note: The picture of the little lion was free downloaded in WMF format from a web site, its author unknown]



```
<?xml version="1.0" standalone="yes"?>
<svg width = "1452px" height="2299px">
<rect x = "0" y = "0" width = "100%" height = "100%" style = "fill: #ffffff"/>
<g style = "stroke: #000000: stroke-width: 0: fill: none">
</g>
<g style = "stroke: #00000
<path  d = "M418,113 L492,                              5,60 L1062,81 L1103,171
</g>
<g style = "stroke: none;
<path  d = "M855,480 L820,                              5 L749,483 L751,525 L716
</g>
<g style = "stroke: none;
<path  d = "M767,612 L795,                              2 L767,612 L767,612"/>
</g>
<g style = "stroke: none;
<path  d = "M1073,1379 L94                              L1100,2057 L1273,1995 L1
</g>
<g style = "stroke: none;
<path  d = "M337,1378 L288                              491,1892 L458,1930 L476,
<path  d = "M444,2180 L475                              579,2241 L610,2237 L650,
<path  d = "M1274,1506 L13                              1830 L1230,1997 L1067,20
<path  d = "M908,1233 L908                              781,1802 L758,1879 L708,
<path  d = "M470,763 L543,                              1242 L913,1147 L850,844
<path  d = "M130,354 L214,                              2 L326,499 L270,568 L206
<path  d = "M430,206 L402,                              4 L185,158 L173,190 L189
<path  d = "M838,448 L850,                              8 L942,465 L933,438 L902
<path  d = "M634,270 L617,                              0 L634,270"/>
<path  d = "M847,293 L850,                              5 L847,293 L847,293 L847
<path  d = "M593,811 L628,                              8 L698,822 L698,840 L718
<path  d = "M587,699 L620,                              4 L810,667 L855,646 L847
<path  d = "M886,201 L914,                              2 L955,150 L981,154 L983
<path  d = "M513,434 L534,                              0 L533,462 L513,434 L513
<path  d = "M518,1286 L479                              468,1476 L490,1461 L476,
</g>
<g style = "stroke: none;
<path  d = "M561,410 L581,                              5 L561,410 L561,410 L561
<path  d = "M867,385 L853,                              6 L867,385 L867,385 L867
</g>
<g style = "stroke: none;
<path  d = "M347,551 L255,                              0 L305,746 L342,719 L334
<path  d = "M501,480 L459,478 L402,493 L453,501 L392,530 L460,526 L392,556 L457,550 L409,579 L463
<path  d = "M656,465 L671,501 L657,539 L683,569 L702,541 L705,492 L688,471 L656,465 L656,465 L656
<path  d = "M737,769 L766,757 L805,763 L819,777 L807,786 L783,774 L748,780 L737,769 L737,769 L737
```

Figure 10. Rendering of the converted SVG file from the WMF file, with the source of the SVG file showing in the background

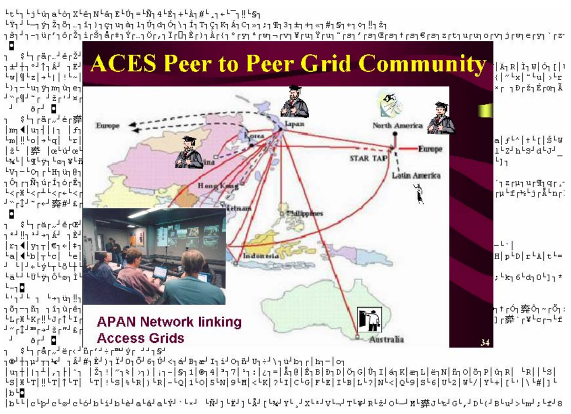
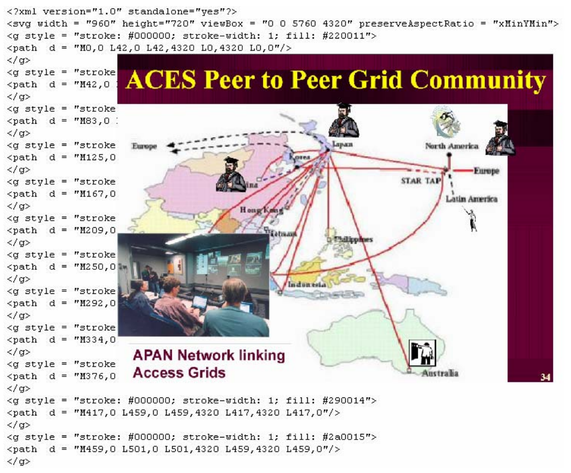


Figure 11. Rendering of a WMF file with the printable source of the WMF file showing in the background



```
<?xml version="1.0" standalone="yes"?>
<svg width = "960" height="720" viewBox = "0 0 5760 4320" preserveAspectRatio = "xMinYMin">
<g style = "stroke: #000000; stroke-width: 1; fill: #220011">
<path  d = "M0,0 L42,0 L42,4320 L0,4320 L0,0"/>
</g>
<g style = "stroke
<path  d = "M42,0
</g>
<g style = "stroke
<path  d = "M83,0
</g>
<g style = "stroke
<path  d = "M125,0
</g>
<g style = "stroke
<path  d = "M167,0
</g>
<g style = "stroke
<path  d = "M209,0
</g>
<g style = "stroke
<path  d = "M250,0
</g>
<g style = "stroke
<path  d = "M292,0
</g>
<g style = "stroke
<path  d = "M334,0
</g>
<g style = "stroke
<path  d = "M376,0
</g>
<g style = "stroke: #000000; stroke-width: 1; fill: #290014">
<path  d = "M417,0 L459,0 L459,4320 L417,4320 L417,0"/>
</g>
<g style = "stroke: #000000; stroke-width: 1; fill: #2a0015">
<path  d = "M459,0 L501,0 L501,4320 L459,4320 L459,0"/>
</g>
```

Figure 12. Rendering of the converted SVG file from the WMF file, with the source of the SVG file showing in the background

6.3. Future Work

Substantial work needs to be done in the implementation of a General Collaboration Web Service we have described, such as converting other kinds of file format to SVG format, automating the Web Service with all kinds of resources as well as with all kinds of WS viewers, and so forth.

One part of our future work resides in the transcoding of event from the viewer to the resource, or from the user-facing I/O to the resource-facing I/O.

Similar experiment has been tried in the Universal CAROUSEL Access project of the Community Grid Lab at Indiana University [20], in which SVG events from a PDA are transcoded to the equivalents of a SVG Viewer on a desktop and control its process wirelessly [15, 16].

We can surely get a clue from this.

7. Conclusion

In this paper we introduced some resources of collaboration applications, and the needs in making them to be Web Services. We proposed and described the idea of Thin Client Collaboration Web Services, and explored some potential scenarios of this idea in which it shows its merit and the freedom resulted in collaboration.

Such a Web Service has two sets of ports: User-facing Input/Output ports and Resource-facing Input/Output ports. The user-facing I/O contacts a WS viewer, and the resource-facing I/O contacts a collaborative application. Hence, the role of the Web Service is to transcode in both directions between the two sets of ports with respect to displays and events, so that the user accesses the WS viewer as if the resource itself.

We used our three collaborative projects as examples of resources, and projects in SVG as the demonstration of our initial effort in the implementation of a General Thin Client Collaboration Web Service. We described briefly the SVG projects, gave some results enough to indicate the effort, and pointed out the future work.

References

[1] WebEx Collaboration Environment
http://www.webex.com
[2] Anabas Collaboration Environment
http://www.anabas.com
[3] Extensible Markup Language (XML) 1.0 (Third Edition)
http://www.w3.org/TR/REC-xml/
[4] Scalable Vector Graphics (SVG) 1.0 Specification
http://www.w3.org/TR/2001/PR-SVG-20010719/
[5] Geoffrey Fox, Hasan Bulut, Kangseok Kim, Sung-Hoon Ko, Sangmi Lee, Sangyoon Oh, Shrideep Pallickara, Xiaohong Qiu, Ahmet Uyar, Minjun Wang, and Wenjun Wu, "Collaborative Web Services and Peer-to-Peer Grids", *Proceedings of 2003 Collaborative Technologies Symposium.* http://grids.ucs.indiana.edu/ptliupages/publications/foxwmc03keynote.pdf
[6] Fran Berman, Geoffrey Fox, and Tony Hey, *Grid Computing: Making the Global Infrastructure a Reality*, John Wiley & Sons Ltd, Chichester, West Sussex PO19 8SQ, England, 2003.
[7] Minjun Wang, Geoffrey Fox, and Shrideep Pallickara, "Demonstrations of Collaborative Web Services and Peer-to-Peer Grids", *Journal of Digital Information Management*, June 2004, Volume 2, Issue 2, pp. 93-96.
http://grids.ucs.indiana.edu/ptliupages/publications/P2PGrids_JDIM_PDF.pdf
[8] Minjun Wang and Geoffrey Fox, "Design of a Collaborative System" for Open Office, *Proceedings of IASTED KSCE 2004 Conference*, US Virgin Islands, November 2004.
http://grids.ucs.indiana.edu/ptliupages/publications/OpenOfficeCollaborativeSystemFINAL.pdf
[9] Minjun Wang, Geoffrey Fox, and Marlon Pierce, "Grid-based Collaboration in Interactive Data Language Applications", *Proceedings of IEEE International Conference on Information Technology*, Las Vegas, Nevada, April 4-6, 2005.
http://grids.ucs.indiana.edu/ptliupages/publications/GridCollabIDL_ITCC2005.pdf
[10] ReviewPlus Data Visualization Software User Manual
http://web.gat.com/comp/analysis/uwpc/reviewplus/manual/
[11] MDSplus: Introduction
http://www.mdsplus.org/intro/index.shtml
[12] Microsoft Windows Metafile
http://wvware.sourceforge.net/caolan/ora-wmf.html
[13] Geoffrey Fox, Shrideep Pallickara, and Xi Rao, "A Scalable Event Infrastructure for Peer to Peer Grids", *Proceedings of 2002 Java Grande/ISCOPE Conference*, Seattle, November 2002, ACM Press, ISBN 1-58113-599-8, pp. 66-75.
http://grids.ucs.indiana.edu/ptliupages/publications/ScaleableEventArchForP2P.doc
[14] Shrideep Pallickara and Geoffrey Fox, "Efficient Matching of Events in Distributed Middleware Systems", *Journal of Digital Information Management*, June 2004, Volume 2, Issue 2, pp. 79-87.
http://grids.ucs.indiana.edu/ptliupages/publications/jdim-vol2-num2.pdf
[15] Sangmi Lee, Geoffrey Fox, Sunghoon Ko, Minjun Wang, and Xiaohong Qiu, "Ubiquitous Access for Collaborative Information System Using SVG", *Proceedings of SVGopen conference*, Zurich, Switzerland, July 2002.
http://grids.ucs.indiana.edu/ptliupages/projects/carousel/papers/draft.pdf
[16] Sangmi Lee, "A Modular Data Pipelining Architecture (MDPA) for Enabling Universal Accessibility in P2P Grids", Florida State University, Ph.D. Dissertation, July 3 2003.
http://grids.ucs.indiana.edu/ptliupages/publications/Sangmi_Lee_thesis.pdf
[17] Global Multimedia Collaboration System
http://www.globalmmcs.org/
[18] H. M. Deitel, P. J. Deitel, J. P. Gadzik, K. Lomeli, S. E. Santry, and S. Zhang, *Java Web Services for Experienced Programmers*, Prentice Hall, New Jersey, 2003.
[19] Microsoft Corp., *Microsoft Windows Programmer's Reference*, Microsoft Press, Redmond, 1990.
[20] Community Grid PDA project
http://grids.ucs.indiana.edu/ptliupages/projects/carousel/