INFORMATICA, 2005, Vol. 16, No. 2, 241–260
© 2005 *Institute of Mathematics and Informatics, Vilnius*

Transcoding Aspects for Image Tele-Collaboration

Sabah M.A. MOHAMMED, Jinan A.W. FIAIDHI
Department of Computer Science, Advanced Technology and Academic Centre
Lakehead University
955 Oliver Road, Thunder Bay, Ontario P7B 5E1, Canada
e-mail: {sabah.mohammed, jinan.fiaidhi}@lakeheadu.ca

Received: September 2004

Abstract. Tele-Collaboration will enable users in different locations to collaborate in a shared, simulated environment as if they were in the same physical room. It's the ultimate synthesis of networking and media technologies to enhance collaborative environments. When participants are Tele-Collaborated, they are able to interact with each other in a shared virtual environment. They are able to query and visualize data stores and steer complex scientific and engineering simulations. This article investigated the transcoding intermediaries required to achieve an effective tele-collaboration on the Web as well as on P2P/Grid environments. Both Scene Graphs as used by the virtual SceneBeans and the SVG DOM tree are found to be essential for the transcoding purpose.

Key words: image trascoding, P2P and grid environments, scene beans, scene graphs, SVG, DOM.

1. Collaboration Intermediaries

Peer-to-peer (P2P) and Grid technologies have recently renewed interest in decentralized systems. Although the Internet itself is the largest decentralized computer system in the world, most systems have employed a completely centralized topology in the 1990s through the massive growth of the Web. With the emergence of P2P in early 2000, the model has swung into radically decentralized architectures such as Gnutella. The P2P style interaction model facilitates sophisticated resource sharing environments between "consenting" peers over the "edges" of the Internet. Resources shared could be anything from CPU cycles, exemplified by SETI@home (extraterrestrial life) and Folding@home (protein folding), to files (Napster and Gnutella). In addition, grid computing provides consistent, inexpensive access to computational resources regardless of their physical location or access point. As such, The Grid provides a single, unified resource for solving large-scale compute and data intensive computing applications. P2P and Grid environments allow people to use mobile carts, handheld PCs, 3G Mobile Phones, PocketPCs, PDAs, Supercomputers, and Computer Networks anywhere, anytime to access seamlessly all of media existing on a particular system or the internet. Peer "interactions" involves advertising resources, search and subsequent discovery of resources, requests for access to these resources, responses to these requests and exchange of messages between peers. Indeed with the rapid increase in the amount of content on the World Wide Web and vast amount of peer grids, it is now becoming clear that information cannot always be stored in a form that anticipates all of its possible uses. Intermediaries becomes a necessity as a computational elements that convert data, on demand, from one form into another between different peers. Since peers interactions, in most P2P and Grid systems as well as on many clients on the Web, are XML-based, applications could be written in any language and can be compiled for any platform through various web intermediaries (WBI). The advent of such intermediaries is to integrate the ubiquitous information infrastructures. The new web intermediaries have larger role than the traditional client-server intermediaries and often termed as "Cybermediaries" (Sarkar *et al.*, 2003). Like the client-side design, intermediary web-based programming requires no "permission" from the Web server to creates additional functionality for the user when the data producer (e.g., server or database) or the data consumer (e.g., browser) cannot be modified. Note that it is not necessary that there is only one intermediary between the client and the service provider as it is possible to combine intermediaries in several ways.

Web Intermediaries (WBI, pronounced "webby") is an architecture and framework for the uniform creation and control of intermediary programs such as proxy servers, transcoding processors, and any kind of program that sits somewhere between two end points in a network. Network intermediaries are computational elements that lie along the path of networked interactions. Some intermediaries are generic, and are used by many different applications, while others serve some very specific purposes. Thus intermediaries are "smart pipes" that can monitor, edit, and generate content that flows between a user and a content provider. The term "intermediary" was coined by the IBM WBI development team (Barrett and Maglio, 2000). Fig. 1 illustrates the general architecture of web intermediaries (Fictcher and Waterhouse, 2002).

It must have the basic "Web Services Stack" which will allow it to seamlessly plug-in to the Web Services invocation route. The "Core Functionality" component is the main functionality of the Web Service. This could itself be a J2EE/J2ME Application, a CORBA based application, or .NET component. Similar to Web Services, it should not

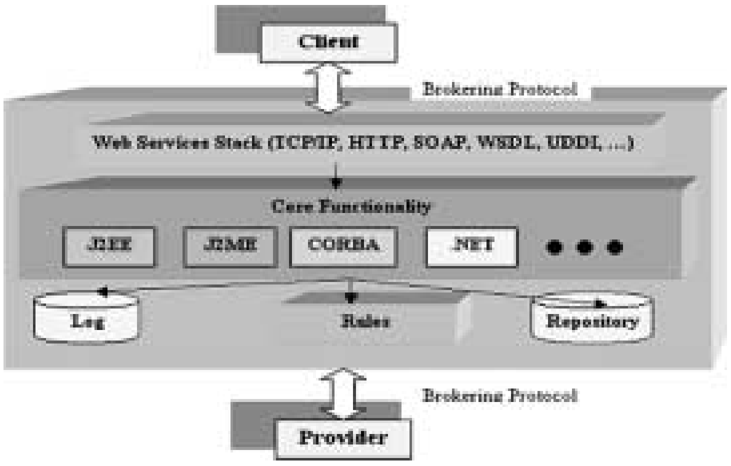


Fig. 1. The main components of a web intermediary.

matter what the platform is or what programming language was used to create this functionality. The “Rules” component can serve to define the behaviour of the Intermediary; how it interacts with different clients, its interaction with other Intermediaries, etc. The “Log” database holds all the information on the messages passing to and from the Intermediary. Finally the “Repository” can be used not only for its core functionality but also as a source of information about other Intermediaries that it needs to interact with. The brokering protocol can be a simple SOAP over http or more sophisticated protocol like JXTA message piping protocol.

In this paper, we are investigating the intermediary-based image transcoding approaches suitable for peers collaborations and present an investigation on suitable protocols protocol for achieving it within P2P/Grid and Web environments.

2. Transcoders as Intermediaries

P2P and Grid technology shifted the trend toward a common Java and XML infrastructure for Internet servers. On the client side, the trend is to have a wide range of clients oriented toward specific purposes, such as smart cards, personal digital assistants, smart telephones, set-top boxes, and smart automobiles. The usefulness of these new clients increases greatly when they have a wide range of content available to them, and this content is often provided by Internet servers. Since many of these new clients are mobile and wireless connectivity as well as the ability to support disconnected operation are important to them. Indeed, different devices require pages to be presented in different markup languages, such as Hypertext Markup Language (HTML), compact HTML 4, Handheld Devices Markup Language (HDML), Wireless Markup Language (WML) and VoiceXML. Even with a particular markup language such as WML, different presentations may be required on different devices. For example, for one WML-based phone, choices are most appropriately presented as buttons. For another phone, they might best be presented as hyperlinks. Indeed, these diverse clients have differing requirements for communicating and presenting data. When attached to Web servers, the best approach for working with these clients is to provide an easy means of translation and tailoring of data to meet specific client needs, a job that is easily handled by XML and *transcoding* technologies. Actually by making the legacy data available via XML-based Web services, systems can greatly extend its reach to its diverse customers.

Transcoding technology can circumvent some of the complexities of content adaptation. It adapts content to match constraints and preferences associated with specific environments. It can modify either the content or the rendering associated with an application. In other words, both computer users and cell phone users can view content in a way that suits their devices, without sacrificing the content itself. Thus, transcoding is vital to pervasive computing because it can bridge the gap from existing Internet Web implementation to later active and dynamic Internet services architectures.

2.1. *Image Transcoding Technologies*

Ubiquitous devices (e.g., GSM Data, HSCSD, GPRS) can use the web based applications. Such devices may vary considerably from an ordinary laptop PC: display size, mass storage capacity and CPU processing power. Wireless networks will also have at 10–100 times less bandwidth and higher latencies than fixed line networks. Many proxy services that adapt TCP/IP protocol stack and some service protocols (like HTTP) have been developed (e.g., IBM Transcoding `http://www-4.ibm.com/software/secureway/transcoder/` and IBM WebSphere®Transcoding Publisher Version 4.0 for Multiplatforms). There are also proxies that manipulate presentation of the content depending on mobile device and mobile network constraints (e.g., Spyglass Prism, `http://www.spyglass.com/solutions/technologies/prism/details.html` and APRO project, `http://www.vtt.fi/tte/projects/apro/`). Usually these proxies are statically configured to support single specific devices. Indeed the ideal approach is to support ubiquity and to adapt dynamically to changes of user's network environment and devices. Indeed, the most significant part of any network performance is spent on Image manipulation. Images represent variety of objects from visual "landmarks" for navigation to users or images that consist of some valuable information. So complete image removal for ubiquitous devices isn't a fruitful approach – instead several dedicated image manipulation methods can be used to conserve the bandwidth within ubiquitous environments (Smith *et al.*, 1998):

- *reduction of image depth*;
- *format modifications* (e.g., GIF to JPEG);
- *JPEG Q-parameter modification*;
- *image scaling*;
- *big image removal*;
- *removal of tags and elements* (e.g., applet and script elements);
- *outlining*;
- *structural changes to presentation format*;
- *background paging*;
- *estimation of the transfer cost and cost visualization to the user*.

Methods listed above partially solve particular ubiquity problems and a more effective, dynamic and reusable solution is required. In this direction the Java platforms and XML are emerging major technologies for performing reusable intermediaries functions. In the followings are the major trends of such technologies:

1. Continued integration of Java and XML into robust intermediaries.
2. Continued and accelerated standardization of Java and XML technologies for infrastructure and industries.
3. Use of transcoding and XML technologies to support a much wider range of clients of every description, both synchronous and asynchronous.
4. The move from tightly coupled applications to loosely coupled Web services.

There are two major XML based image transcoding technologies that we have sighted in literature: SVG and Scene Graph. The following two sections describe our investigation on why we prefer the Scene Graph Image Transcoding.

2.1.1. *SVG Image Transcoding*

For XML, the bulk of the infrastructure standards work is driven by the ebXML initiative (`http://www.ebxml.org`) and by the World Wide Web Consortium (W3C at `http://www.w3c.org`). In relation with graphics and images, W3C concentrated on SVG (Scalable Vector Graphics) initiative. SVG is a language for describing two-dimensional graphics and graphical applications in XML. SVG 1.1 is a W3C Recommendation and forms the core of the current SVG developments. SVG 1.2 is the specification currently being developed. The SVG Mobile Profiles: SVG Basic and SVG Tiny are targeted to resource-limited devices and are part of the 3GPP platform for third generation mobile phones. SVG Print is a set of guidelines to produce final-form documents in XML suitable for archiving and printing. Furthermore, by using an XML syntax, SVG is extremely easy to generate, search, transform and manipulate. Fig. 2 illustrates an example of representing a rectangle image in SVG. Unlike other formats, SVG becomes a powerful tool for anybody managing image content for the Web or other environments (see Fig. 3). By leveraging the force of XML and the visual strengths of dynamic and easily accessible vector graphics (i.e., SVG), the Apache XML Project's Batik team extends this power in building an industrial-grade embeddable Web graphics software solution. Batik delivers core components for three main purposes:

– generating SVG content from any Java applications,
– viewing SVG content, and

```
<svg width="400" height="450">
  <rect x="10" y="20" width="100" height="50" style="fill:red"/>
</svg>
```

Fig. 2. Representing a rectangle in SVG.

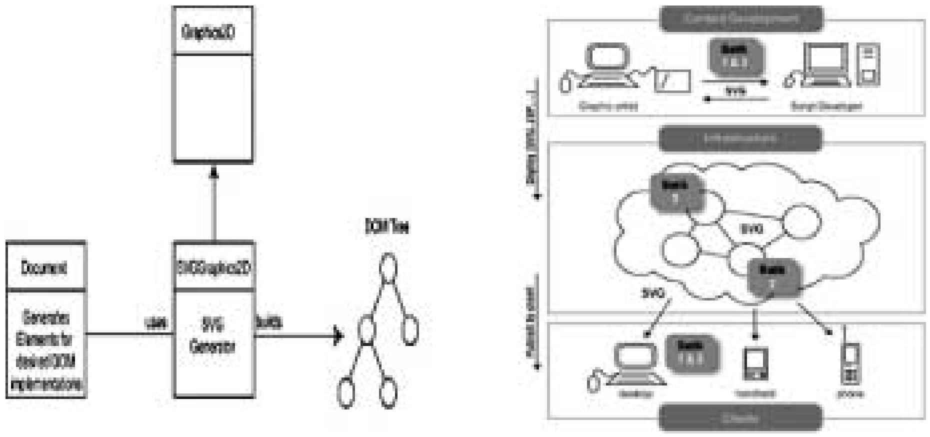


Fig. 3. SVG-based image transcoding.

– converting SVG to and from other formats.

In order to see how the SVG Generator works, the following example illustrates how you can create an instance of the SVGGraphics2D and use it as a regular Graphics2D object to draw graphics. Then, it demonstrates how you can stream out the generated DOM tree (which is the in-memory representation of the SVG document and the graphics as well).

```
import org.apache.batik.dom.GenericDOMImplementation;
import org.w3c.dom.Document;
import org.w3c.dom.DOMImplementation;
public class TestSVGGenerator {
  public void paint(Graphics2D g2d) {
    g2d.setPaint(Color.red);
    g2d.fill(new Rectangle(10, 10, 100, 100));
  }
  public static void main(String [] args) throws IOException {
    // Get a DOMImplementation
    DOMImplementation domImpl =
      GenericDOMImplementation.getDOMImplementation();
    String svgNamespaceURI = ”http://www.w3.org/2000/svg”;

    // Create an instance of org.w3c.dom.Document
    Document document =
      domImpl.createDocument(svgNamespaceURI, ”svg”, null);
    // Create an instance of the SVG Generator
    SVGGraphics2D svgGenerator = new SVGGraphics2D(document);
    // Render into the SVG Graphics2D implementation
    TestSVGGenerator test = new TestSVGGenerator();
    test.paint(svgGenerator);
    // Finally, stream out SVG to the standard output using UTF-8
    // character to byte encoding
    boolean useCSS = true; // we want to use CSS style attribute
    Writer out = new OutputStreamWriter(System.out, ”UTF-8”);
    svgGenerator.stream(out, useCSS);
  }
}
```

The Batik toolkit provides a module called Transcoder. One of the main class is the ImageTranscoder that lets developers convert a SVG document to a raster image such as PNG or JPG. The ImageTranscoder takes a TranscoderInput and a TranscoderOutput, which are respectively the input data to transcode and the output into which the resulting data will be stored. The transcoder supports different types of input such as an InputStream, a Document, or a Reader and different types of output such as an OutputStream, or a Writer. The following example is using the PNGTranscoder and shows how to transform a SVG document to a PNG image.

```
import java.io.*;
import org.apache.batik.transcoder.image.PNGTranscoder;
import org.apache.batik.transcoder.TranscoderInput;
import org.apache.batik.transcoder.TranscoderOutput;
// Create a PNG transcoder
PNGTranscoder transcoder = new PNGTranscoder();
// Create the transcoder input
String svgInputURI = ...;
TranscoderInput input = new TranscoderInput(svgInputURI);
// Create the transcoder output
OutputStream ostream = ...;
TranscoderOutput output = new TranscoderOutput(ostream);
// Transform the svg document into a PNG image
transcoder.transcode(input, output);

// Flush and close the stream
ostream.flush();
ostream.close();
```

For example, to set the encoding quality of a JPG, the following code can be used on a JPEG-Transcoder:

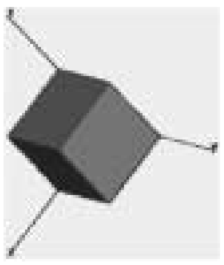
```
// Create a JPG transcoder
JPEGTranscoder transcoder = new JPEGTranscoder();
transcoder.addTranscodingHint(JPEGTranscoder.KEY_QUALITY, new Float(.8));
// ...
```

Or in order to control the size of the image, the following code can be used on any ImageTranscoder:

```
// Create an ImageTranscoder
ImageTranscoder transcoder = new ...;
transcoder.addTranscodingHint(ImageTranscoder.KEY_WIDTH, new Integer(100));
```

Although SVG technology presents an innovative solution in image transcoding, its syntax remains only rich for 2D graphics. You can certainly model 3D images (see Fig. 4) but manipulating and interacting with such 3D SVGs requires further tools such as XSLT or ECMAScript (`http://www.el-mundo.es/internet/ecmascript.html`) to act as handlers of DOM events and modify DOM nodes in order to create the effect of a shifting view in response to user input on the view UI. This is possible since SVG supports external script files.

As an XML based format, SVG also requires an XML parser to manipulate the DOM tree. SVG has several benefits as a file format, but for defining content of images especially in 3D for greater interactivity and tele-collaborative applications appear not practical. SVG files for such applications tend to grow rather large and their online parsing performance and DOM handling should become an obstacle. However, many trials have been made to enhance the overall performance of SVG based applications by using the binary form of SVG (e.g., WBXML `http://www.w3.org/TR/wbxml`, and Plazmic `http://www.svgopen.org/2002/papers/hayman__suitability_of_svg_for_wireless_applications/`), but as a compiled format of the SVG DOM, it will not enable collaborating users to change the contents which is a highly important factor in tele-collaboration.



```
<?xml version=”1.0” encoding=”utf-8”?>
<!DOCTYPE svg PUBLIC ”-//W3C//DTD SVG 1.1//EN”
 ”http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd”>
<svg width=”100%” height=”100%” viewBox=”-600 -600 1200 1200”
  xmlns=”http://www.w3.org/2000/svg” version=”1.1”>
 <desc>View cube.xml as svg</desc>
  <defs>
    <marker id=”ArrowHead”
          viewBox=”0 0 10 10” refX=”0” refY=”5”
          markerUnits=”strokeWidth”
          markerWidth=”8” markerHeight=”6”
          orient=”auto”>
          <path d=”M 0 0 L 10 5 L 0 10 z” />
    </marker>
 </defs>
 <g id=”Background”>
    <rect x=”-600” y=”-600” width=”1200” height=”1200”
 fill=”#F0F0E0” stroke=”none”/>
 </g>
 <g id=”CoordinateAxes” transform=”scale(1,-1)”>
   <path fill=”none” stroke=”#000000” stroke-width=”1” marker-end=”url(#ArrowHead)”
 d=”M 250 200 L 33.93691 -88.695299”/>
   <text fill=”red” font-size=”20pt” x=”33.93691” y=”-88.695299”
 transform=”translate(33.93691,-88.695299) scale(1,-1) translate(-33.93691,88.695299)”>x</text>
   <path fill=”none” stroke=”#000000” stroke-width=”1” marker-end=”url(#ArrowHead)”
 d=”M 250 200 L 466.06308 142.26094”/>
   <text fill=”red” font-size=”20pt” x=”466.06308” y=”142.26094”
 transform=”translate(466.06308,142.26094) scale(1,-1) translate(-466.06308,-142.26094)”>y</text>
   <path fill=”none” stroke=”#000000” stroke-width=”1” marker-end=”url(#ArrowHead)”
 d=”M 250 200 L 33.93691 430.95624”/>
   <text fill=”red” font-size=”20pt” x=”33.93691” y=”430.95624”
 transform=”translate(33.93691,430.95624) scale(1,-1) translate(-33.93691,-430.95624)”>z</text>
   </g>
   <g id=”ThreeDimensionalShape” transform=”scale(1,-1)”>
     <polygon fill=”none” stroke=”none” points=”250,200 358.03154,171.13047
  250,26.782820 141.96846,55.65235”/>
     <polygon fill=”none” stroke=”none” points=”250,200 141.96846,315.47812
 250,286.60859 358.03154,171.13047”/>
     <polygon fill=”none” stroke=”none” points=”250,200 141.96846,55.65235
 33.93691,171.13047 141.96846,315.47812”/>
     <polygon fill=”#880088” stroke-width=”2” stroke-linejoin=”bevel” stroke=”#0000ff”
 points=”141.96846,315.47812 33.93691,171.13047 141.96846,142.26094 250,286.60859”/>
     <polygon fill=”#440044” stroke-width=”2” stroke-linejoin=”bevel” stroke=”#0000ff”
 points=”141.96846,55.65235 250,26.782820 141.96846,142.26094 33.93691,171.13047”/>
     <polygon fill=”#CC00CC” stroke-width=”2” stroke-linejoin=”bevel” stroke=”#0000ff”
 points=”358.03154,171.13047 250,286.60859 141.96846,142.26094 250,26.782820”/>
   </g>
 </svg>
```

Fig. 4. Representing a Cube using SVG.

2.1.2. *SceneBeans Image Transcoding*

SceneBeans are originally introduced as a java component-based 2D image animation framework by Pryce Magee during 2001 (Pryce and Magee, 2001). SceneBean *animation* encapsulates both a scene-graph and the behaviors that animate the nodes of that graph. Most importantly, a SceneBean animation is also a scene-graph node, since this means we can compose animations, applying transformation (e.g transcoding) and further animation as required. A SceneBean is a reusable component that can be used in design/run time Java Beans environments. For the outside word a SceneBean looks like a 2-D object with well-defined properties and behaviours described in XML and constructed as a Scene Graph. SceneBean fully supports all core features that Java Beans architecture provides such as introspection, customization, persistence, event and properties model. Complete support of Java Beans architecture should allow design time environments such as bean builders operate SceneBeans exactly the same way as other beans. This should make construction of different applets and applications as building blocks possible in bean builders. In run-time environment SceneBeans as a part of some applet or application provides user with visual 2-D interface. When used in bean builders SceneBeans may provide its own 2-D GUI to customize its properties visually (Fiaidhi *et al.*, 2004). SceneBeans should be able to provide different levels of Scene Graph abstraction, from complete scene to separate nodes. From this follows that every Scene Graph node should have corresponding SceneBean class encapsulating functionality of this node. All SceneBean components are packaged in JAR file. A SceneBeans is loaded by means of custom URL from JAR file. When a new bean is added, the existing beans or at least any parents of this bean need to know about it. So new shape types, can be written as beans and added to the program. The bean will have procedures to draw itself in 2D, describe itself in the tree view and read/write (serialise) itself to the standard formats. File read/write is done by serialising the scenegraph structure. This starts with the rootBean this saves its own parameters including child nodes below it (SFNode and MFNode). These child nodes are saved by calling their own save procedures and so on recursively. So the whole scene graph can be saved and each node only needs to know how to save its own parameters. File read is done in the same way. So when a new node is added, it only needs to know how to load and save itself, No other classes and no general read/write parsers are needed.

A SceneBean object is described using an XML metadata. The top-level of the description is called <animation> which contains five types of sub-elements: a single <draw> element defines the scene object to be rendered; <define> elements define named scene-graph fragments that can be linked into the visible scene graph; <behaviour> elements define behaviors that animate the scene graph; <event> elements define the actions that the animation performs in response to internal events; and <command> elements name a command that can be invoked upon the animation and define the actions taken in response to that command. Both <draw> and <define> elements can contain the elements <primitive>, <transform>, <style> and <compose>. SceneBeans defines an object behavior with the "behavior" element and then animating the parameters of scene-graph nodes with the "animate" tag. The behavior tag is used to instantiate behavior beans: the parser maps the algorithm of the behavior to a Java class the same way as it does for scene-graph nodes, although it searches a different set of packages. Like scene object nodes, param tags are used to configure behaviors by setting their JavaBean properties. SceneBeans is a framework for two-dimensional animations based on Java Beans. An animation in SceneBeans is defined as a “scene graph” – Java Beans components that form a directed a cyclical graph (DAG). Animations are defined in XML documents and contain commands that instantiate and bind Java Beans components to each other. The XML file used by SceneBeans requires a parser to translate the XML document into interactive objects. The XML document type definition (DTD) used by the SceneBeans parser is relatively minimal compared to DTDs for similar applications, such as the SVG standard. The DTD does not prescribe a limited number of component types and their options, but instead describes compositions of components that the parser loads dynamically and manipulates generically through the JavaBeans APIs. Fig. 5 illustrates an XML metadata for an animated copter scene.

Like scene graph nodes, <param> tags are used to configure behaviours by setting their JavaBean properties. Behaviours must be identified by an id attribute so they can be referred by an <animate> element within the scene graph. Animate elements create a binding between a behaviour and a property of a bean so that the behaviour modifies the value of the property over time, therefore creating an animation. Commands that can be invoked upon an animation are introduced by <command> elements which contain one or more action elements of type <set>, <stop>, <start>, <reset>, <invoke> or <announce>. Fig. 6 illustrates the SceneBean interface for processing the copter XML file.

Actually, scene graph is a common model for storing and retrieving graphical scenes as used in computer graphics. A scene graph is a general data structure commonly used by vector-based graphics editing applications. Many graphical applications uses the model of scene graph to model flexible shapes (e.g., AutoCAD, Adobe Illustrator, CorelDraw and OpenGL). The scene graph contains the pictorial data items that can be edited, shared and displayed. Scene graphs showed clear benefits for improving rendering performance and making more optimal use of the available hardware resources. By keeping a “retained” model of the virtual world, scene graphs could make additional optimizations, such as parallel processing culling and drawing, and most importantly: state sorting. State sorting is a concept whereby all of the objects being rendered are sorted by similarities in state (texture map, lighting values, transparency, and so on). Since changing state is often an expensive operation due to hardware implementations, this is usually a big performance win, even on the newest hardware. However, the main reason for focusing on scene graph model is not only the performance gain. The primary idea behind scene graph is its high level organization. It gives all objects types, even if they are totally different, a unified access mechanism. For this reason the importance of scene graphs as a powerful tool for modeling any scene including virtual reality or 3D scenes has been extended and currently most of the notable software venders to adopt it for their graphics APIs (e.g., Java3D, VRML, Cult3D, OpenSceneGraph, Cosmo 3D, X3D MESH , IrisGL, Performer and OpenGL/Direct3D). Particularly Java3D is getting more popularity as it combines the vast knowledge of the collaborated companies which includes venders Intel, Silicon Graphics, Apple, and Sun. Java3D has been designed to be a platform-independent

```
<?xml version=’’1.0’’?>
<animation width=’’256’’ height=’’256’’>
  <behaviour id=’’rotor-spin’’ algorithm=’’Loop’’
             state=’’\$\{rotor\_state=stopped\}’’>
    <param name=’’from’’ value=’’0.0’’ />
    <param name=’’to’’ value=’’2*pi’’ />
    <param name=’’duration’’ value=’’1.0’’ />
  </behaviour>
<command name=’’start’’>
  <start behaviour=’’rotor-spin’’ />
</command>
<command name=’’stop’’>
  <stop behaviour=’’rotor-spin’’ />
</command>
<define id=’’rotor’’>
  <primitive type=’’polygon’’>
    <param name=’’pointCount’’ value=’’4’’ />
    <param name=’’points’’ index=’’0’’ value=’’(0, 0)’’ />
    <param name=’’points’’ index=’’1’’ value=’’(-16, 96)’’ />
    <param name=’’points’’ index=’’2’’ value=’’(0, 100)’’ />
    <param name=’’points’’ index=’’3’’ value=’’(16, 96)’’ />
  </primitive>
</define>
<define id=’’rotors’’>
  <style type=’’RGBAColor’’>
    <param name=’’color’’ value=’’000000’’/>
    <primitive type=’’circle’’>
      <param name=’’radius’’ value=’’12’’ />
    </primitive>
  </style>
  <transform type=’’rotate’’>
    <param name=’’angle’’ value=’’1.0’’ />
    <animate param=’’angle’’ behaviour=’’rotor-spin’’ />
    <style type=’’RGBAColor’’>
      <param name=’’color’’ value=’’888888’’/>

      <paste object=’’rotor’’ />
      <transform type=’’rotate’’>
        <param name=’’angle’’ value=’’2*pi/3’’ />
        <paste object=’’rotor’’ />
      </transform>
      <transform type=’’rotate’’>
        <param name=’’angle’’ value=’’4*pi/3’’ />
        <paste object=’’rotor’’ />
        </transform>
      </style>
    </transform>
  </define>
  <draw>
    <paste object=’’rotors’’ />
  </draw>
</animation>
```

Fig. 5. An XML metadata of an animated copter scene.

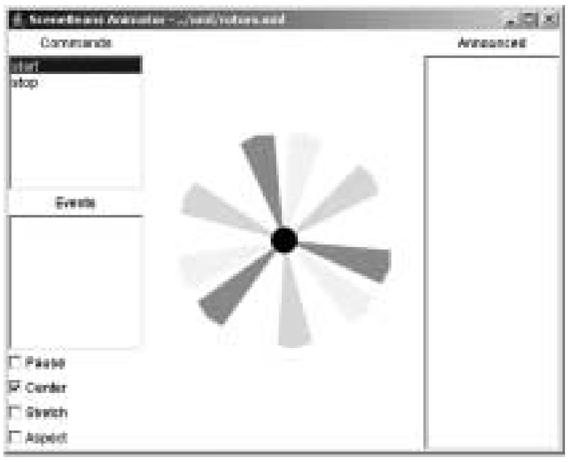


Fig. 6. SceneBeans interface for rotors.xml.

API concerning the host's operating system (PC/Solaris/Irix/HPUX/Linux) and graphics (OpenGL/Direct3D) platform, as well as the input and output (display) devices. The implementation of Java3D is built on top of OpenGL, or Direct3D. The high level Java3D API allows rapid application development which is very critical, especially nowadays.

A Java 3D scene graph consists of a collection of Java 3D node objects connected in a tree structure. These node objects reference other scene graph objects called *node component objects*. All scene graph node and component objects are subclasses of a common SceneGraphObject class. The SceneGraphObject class is an abstract class that defines methods that are common among nodes and component objects. Scene graph objects are constructed by creating a new instance of the desired class and are accessed and manipulated using the object's set and get methods. Once a scene graph object is created and connected to other scene graph objects to form a subgraph, the entire subgraph can be attached to a virtual universe – via a high-resolution Locale object-making the object *live* Prior to attaching a subgraph to a virtual universe, the entire subgraph can be *compiled* into an optimized, internal format. The Java 3D renderer incorporates all graphics state changes made in a direct path from a scene graph root to a leaf object in the drawing of that leaf object. The View object is the central Java 3D rendering object for coordinating all aspects of viewing. All viewing parameters in Java 3D are either directly contained within the View object or within objects pointed to by a View object. Java 3D supports multiple simultaneously active View objects, each of which can render to one or more canvases (see Fig. 7).

The basic idea of adopting Java3D within a flexible component based architecture can be easily done within the framework of Java Beans. The viewing of the 3D scene will be the responsibility of these beans. A bean interface will then represent an animate loop that runs continuously which alternately calls two other methods (e.g step then render). Step and render are both called on sceneBean, which is the root of the scenegraph, and are then called on the subnodes so that they are called on all nodes in the scenegraph.

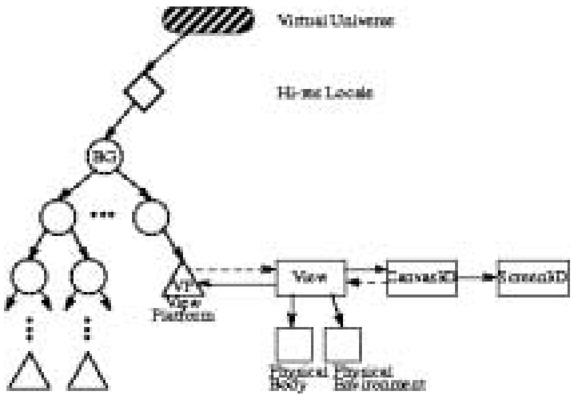


Fig. 7. Viewing a Scene using Java3D.

The reason that they are separate methods and not all done in the same method is that if there are many views we may have to render the scene many times before stepping on to the next frame. Fig. 8 illustrates the primitive idea of rendering Java3D scene within the framework of Java beans.

Such primitive implementation of the rendering process based on Java beans was introduced recently by Martin Baker at his web page (`www.martinb.com`). However, Baker's rendering APIs supports only VRML type events which are unnecessarily complex. For this reason, many rendering libraries (such as RealityLab, RenderMorphics, IRIS Performer) have developed a simpler notion of *rendering* than VRML 1.0. However, the mismatch between these rendering libraries and VRML causes performance problems and implementation complexity, and these problems become much worse in VRML 2.0 as we add the ability to change the world over time. There are many other problems associated with the use of VRML, in decreasing order of importance. These problems are:

1. **Browser Configuration.** While virtual reality browsers exist, most are imple-

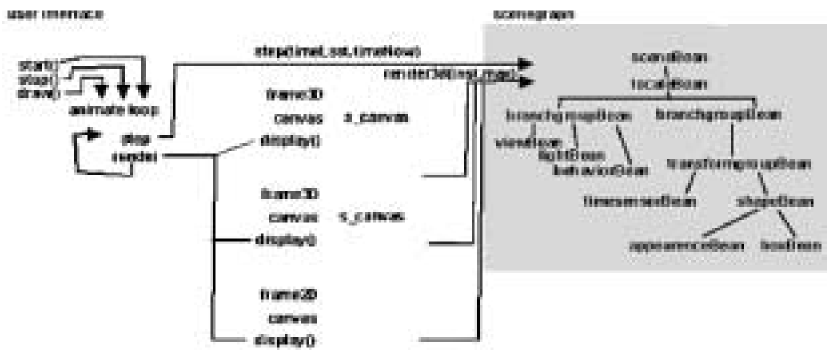


Fig. 8. Rendering a Java3D frame within the framework of Java Beans.

mented as plug-ins to Netscape and Internet Explorer, and most users are unwilling or unable to install such plug-ins.

2. **Browser Performance.** VRML files are typically large and processing is non-trivial, which result in substantial delays in both scene downloading and scene rendering.
3. **Browser Navigation.** Navigation in virtual 3D spaces is hard for the average user.
4. **Cost of Content.** 3D scene creation is much more difficult, time-consuming, and expensive than creation of textual and 2D graphical data.
5. **Scene Connectivity.** VRML hyperlinking results in a discontinuity and loss of the virtual reality feeling.

However and in order to overcome these problems associated with VRML, we can use the standard WC3 X3D or XJ3D instead of VRML in describing Java3D scene graphs. This approach has been implemented by Fiaidhi (2004) in which it was called Virtual SceneBeans. The use of Virtual SceneBeans require changing the way Baker's describe and parse the beans events. Moreover by using virtual SceneBeans, peers can interact with the image rendering engine by changing the X3D/XJ3D description or altering directly the Scene Graph. The X3D/XJ3D specification describes the map and objects that are to be rendered by the displaying engine and the X3D/XJ3D parser is responsible for interpreting the inputs into the Scene Graph which can then be viewed into the required scene. Actually, the use of Java 3D provides application programmers with two different means for reusing scene graphs (see (Ebner, 2002)). First, multiple scene graphs can share a common subgraph. Second, the node hierarchy of a common subgraph can be cloned, while still sharing large component objects such as geometry and texture objects. In the first case, changes in the shared subgraph affect all scene graphs that refer to the shared subgraph. In the second case, each instance is unique-a change in one instance does not affect any other instance. An application that wishes to share a subgraph from multiple places in a scene graph must do so through the use of the Link leaf node and an associated SharedGroup node. The SharedGroup node serves as the root of the shared subgraph. The Link leaf node refers to the SharedGroup node (see Fig. 9).

A SharedGroup node allows multiple Link leaf nodes to share its subgraph according to the following semantics:

- A SharedGroup may be referenced by one or more Link leaf nodes. Any runtime changes to a node or component object in this shared subgraph affect all graphs that refer to this subgraph.
- A SharedGroup may be compiled by calling its compile method prior to being referenced by any Link leaf nodes.

Only Link leaf nodes may refer to SharedGroup nodes. A SharedGroup node cannot have parents or be attached to a Locale. A shared subgraph may contain any group node, except an embedded SharedGroup node (SharedGroup nodes cannot have parents).

However, sharing subgraphs among peers as well as conveying the X3D image description requires a suitable protocol to enforce peers collaborations and user immersion. The next section provides our view on such protocol.

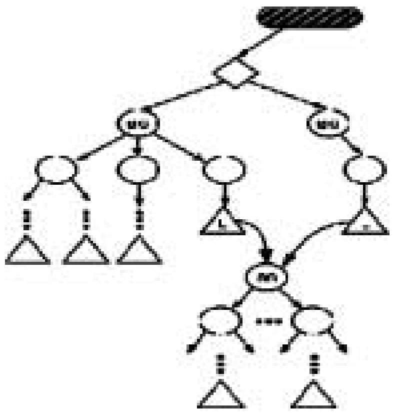


Fig. 9. Shared subgraph.

3. An Intermediary Protocol for P2P and Grid Environments

At present, most tele-collaboration applications are not designed in the approach of open system and cannot communicate with each other. It is of substantial benefits to Internet users if we can have an integrated collaboration environment, which combines image streaming, instant messaging as well as other collaboration applications into a single easy-to-use, intuitive environment. Therefore, it is very important to create a more general framework to cover the wide range of collaboration solutions and allow different users from different communities to collaborate. To integrate heterogeneous systems into one collaboration system, we need to reach the following goals (Fox *et al.,* 2003):

(1) Different kinds of application endpoints should join/leave in the same collaboration session.

(2) Different providers for multipoint multimedia and data collaboration should be connected together to build unified multimedia and data multipoint channels.

(3) A common user interface should be present for all the collaboration participants using different multimedia and data application endpoints.

Especially the first goal requires a common signalling control protocol and event bus, which specifies the message exchange procedure between different types of collaboration endpoints and session servers. Web-service seems to be the best candidate for this framework since it can run across various platforms and is easy to be extended and understood. The control protocol consists of three parts: user session management, application session management and resource contention management. Since there are various control protocols for different collaboration technologies, we need to wrap them into web-services and integrate these services in a more general framework. In addition, portlets (Hepper and Hesmer, 2003) which have become an increasingly popular concept can be used to describe visual user interfaces to these heterogeneous services. A/V and other data collaboration endpoints can be wrapped into portlets to make them modular, reusable software components. Moreover a XGSP collaboration portal can be built by aggregating these portlets. This way of building an application is termed as MVC (Model, View and Controller) and has significant advantages over traditional bundled applications and intensively used in distributed systems including JavaAWT and Swing. We can target applications to multiple clients by intercepting messages and acting as a distributed gateway between different styles of Grids and peer-to-peer networks by making the uniform bus. Thus this uniform bus acts as intermediary or a federation broker for different different distributed systems. Luckily most of such requirements are available in Narada Brokering Intermediary Protocol (`www.naradabrokering.org`) which is an open-source protocol that has features of both the Java Message Service (JMS) and the peer-to-peer technology JXTA. The Narada Brokering Protocol deals with all types of communication whether it is in the form of control packets or RTP based multimedia streams. Based on Narada Brokering, systems like, the Global-MMCS video conferencing (Uyar *et al.*, 2003) which enables heterogeneous clients to join the same real-time multimedia sessions. It provides a flexible architecture to support even more standards and applications. Thus allowing Web/Grid services for session control, media type conversion, audio mixing and video mosaicing.

For constructing a collaborative Web service which can let learners to share Virtual SceneBeans, we developed an intermediary protocol (see (Fiaidhi, 2004)). In this protocol, the resource-facing input/output ports supply the information which is to define the state of the Web Service; the user-facing input, output ports pass control information by the user, and supply information for constructing the user interfaces. The messages on these ports define the state and results of each Web Service. On one side of the port SceneBeans can be converted to a SceneGraph object (i.e., a tree with nodes representing the image) and on the other side the GVT (graphical Victor Toolkit) can reflect/view the Scene Graph structure for rendering purposes. The Scene Graph events specifications provide a generic event model that propagates changes in the scene nodes of its tree structure. Moreover, we need to use the XGSP (XML General Session Protocol), SOAP (Simple Object Access Protocol) and the RTP (Transport Protocol for Real-Time Applications) protocols to describe registration, session parameters, session membership

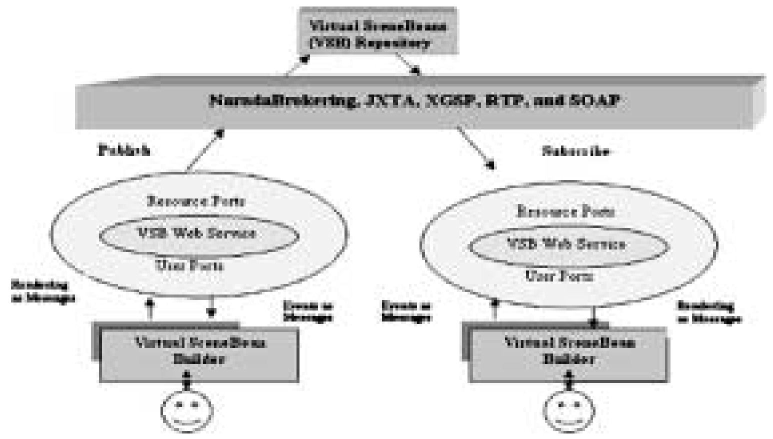


Fig. 10. Virtual SceneBeans tele-collaboration protocol.

and negotiation of the SceneBeans. Fig. 10 illustrates the proposed Virtual SceneBeans Interface Web Service.

4. Conclusions

Today, content providers on the Web grids are under constant pressure to make information available in a variety of formats and for a variety of purposes including tele-collaboration. Intermediaries extend the Client-Server web model to include content computation at client and workgroup proxy. This programming model provides a new degree of flexibility and allows applications to do things that they could not do before. Our vision is to provide a better every-day working environment, with high-fidelity scene reconstruction for life-sized 3D tele-collaboration using such intermediaries. In particular, we want to provide the user with a true sense of presence with our remote collaborator and their real surroundings, and the ability to share and interact with 3D scenes regardless of the different rendering capabilities of their viewing devices. The challenges related to this vision are enormous and involve many technical tradeoffs, particularly in scene sharing and reconstruction. In this paper we presented an investigation toward this ultimate goal. By assembling the best of available intermediary technologies in scene reconstruction, rendering, and distributed scene graph, we can achieve this goal. This recipe for achieving such goal should be based on Virtual SceneBeans with a P2P/Grid protocol which is basically built upon Narada Brokering middleware. The developed prototype for a toolkit that can be used for image tele-collaboration is still under development by our ongoing research project at Lakehead. Fig. 11 shows our toolkit interface. The Scene Graph editor used in the developed toolkit resembles a software developed by brown university (`www.cs.brown.edu/exploratories/home.html`).

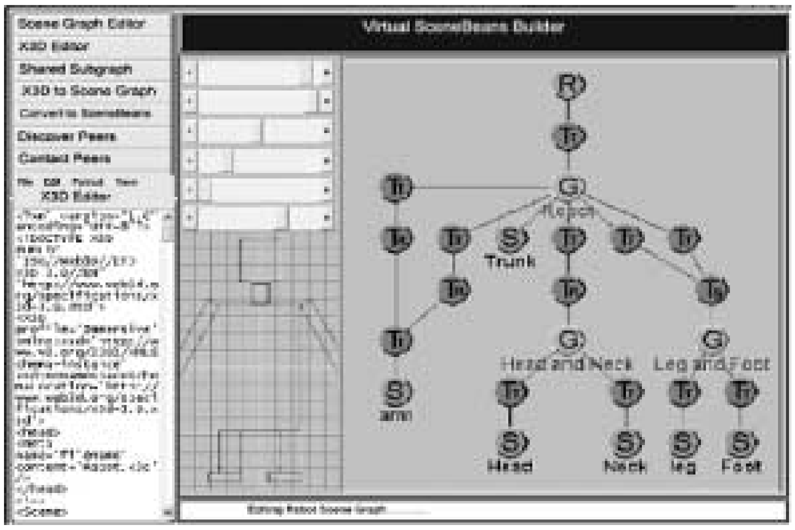


Fig. 11. The GUI for or virtual SceneBeans Builder interface.

References

Ebner, M. (2002). Evolutionary design of objects using scene graphs. In C. Ryan *et al.* (Eds.), *Proceedings of the 6th European Conference on Genetic Programming* (*EuroGP 2003*). Essex, Springer-Verlag, UK. pp. 47–58.

Fiaidhi, J. (2004). Virtual SceneBean: a learning object model for collaborative virtual learning environment. *Informatics in Education*, **3**(2).

Fiaidhi, J., S. Mohammed and S. Sisko (2004). SceneBeans: a tool for constructing collaborative multimedia learning objects. In *9th Western Canadian Conference on Computing Education* (*WCCE04*), May 6–7. Kelowna, BC, Canada. pp. 187–195.

Fictcher, P., and M. Waterhouse (2002). *Web Services Business Strategies and Architectures*. Wrox Press, ASIN: 1904284132.

Fox, G. *et al.* (2003). A web services framework for collaboration and videoconferencing. In *Workshop of Advance Collaboration Environment* (*WACE*), June 22. Seattle.

Hepper, S., and S. Hesmer (2003). Introducing the Portlet specification. *Java World Journal*, August 1 issue. `http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet.html`

Pryce, N., and J. Magee (2002). *SceneBeans*: *A Component-Based Animation Framework for Java*. Technical Report, Imperial College. `www.dse.doc.ic.ac.uk/Software/SceneBeans/downloads/scenebeans.pdf`

Sarkar, M.B. *et al.* (2003). Intermediaries and cybermediaries: a continuing role for mediating players in the electronic marketplace. *JCMC*, **1**(3).

Smith, J., R. Mohan and C.S. Li (1998). Content-based transcoding of images in the internet. In *Proceeedings of IEEE Int. Conference on Image Processing*, October (ICIP-98).

Uyar, A. *et al.* (2003). An integrated videoconferencing system for heterogeneous multimedia collaboration. In *7th IASTED International Conference on Internet and Multimedia Systems and Applications*, August 13–15. Honolulu, Hawaii.

S.M.A. Mohammed received his BSc in applied mathematics (Baghdad University 1977), and his graduate degrees in computer science from Glasgow University (MSc 1981), and Brunel University (PhD 1986). Since late 2001, Dr. Mohammed is an associate professor of computer science at Lakehead University. Formerly, from 1986–1995, Dr. Mohammed was an assistant/associate professor of computer science at Baghdad University holding the position of the Graduate Organizer in Computer Science. During 1996–2001, he served as the chair of computer science at four different universities: Amman University (1995–1996), Philadelphia University (1996–1997), Applied Science University (1997–2000), and Higher College of Technology(2000–2001). Dr. Mohammed has co-authored four text books in compilers, artificial intelligence, Java pogramming and applied image processing, published over 70 refereed publications, was the MSc advisor for 17 students and 1 PhD student, and has received research support from a variety of governmental and industrial organizations. Dr. Mohammed provided consultations for a variety of organizations. Dr. Mohammed organized two international conferences on computers and their applications during 1997 (at Philadelphia University) and 1998 (at Applied Science University) as well as a Regional Symposium on eEducation during 2001 (at Higher College of Technology). Dr. Mohammed's research interests include medical image processing, multimedia learning objects, lightweight security, artificial intelligence and fuzzy logic. Dr. Mohammed is also on the editorial boards of the International Journal of Computing and Information Sciences, Regional Editor of Journal of Information and Technology, Regional Editor of the American Journal of Applied Science, member of the Editorial board of the International Arab Journal of Information Technology and the Asian Journal of Information Technology. Dr. Mohammed is a member of the British Computer Society, member of the Canadian Image Processing & Pattern Recognition Society, member of the IEEE Signal Processing Society and member Imaging Science and Technology Society.

J.A.W. Fiaidhi is a professor of computer science at Lakehead University. She received her graduate degrees in computer science from Essex University , UK (1983), and PhD from Brunel University, UK. (1986). She served also as assistant/associate/full professor at University of Technology, Philadelphia University, Applied Science University and Sultan Qaboos University. Dr. Fiaidhi's research interests include learning objects, XML search engine, multimedia learning objects, recommender systems, software forensics, Java watermarking, collaborative eLearning systems, and software complexity. Dr. Fiaidhi is one of Canada Information Systems Professional (I.S.P.), member of the British Computer Society (MBCS), member of the ACM SIG Computer Science Education, and member of the IEEE Forum on Educational Technology.

Nuotolinio bendradarbiavimo vaizdų perkodavimo aspektai

Sabah M.A. MOHAMMED, Jinan A.W. FIAIDHI

Nuotolinis bendradarbiavimas teiks galimybę tarpusavyje nutolusiems vartotojams bendradarbiauti virtualioje aplinkoje taip, lyg jie būtų toje pačioje vietoje. Tai kompiuterinių tinklų ir žiniasklaidos sintezė siekiant patobulinti bendradarbiavimo aplinkas. Nuotolinio bendradarbiavimo dalyviai gali vydyti paiešką duomenų saugyklose ir vizualizuoti tuos duomenis, valdyti sudėtingo mokslinio ir inžinierinio modeliavimo eigą.

Straipsnyje išnagrinėtos perkodavimo priemonės, reikalingos efektyviam bendradarbiavimui Interneto bei P2P/Grid aplinkose. Nustatyta, kad tiek Scene Graphs, tiek SVG DOM yra būtini perkodavimui.