SceneBeans: A Tool for Constructing Collaborative Multimedia Learning Objects[#]

Jinan FIAIDHI, Sabah MOHAMMED and Stephan SISKO
Department of Computer Science, Lakehead University

ABSTRACT

Interactive multimedia elements allow information to be presented in a comprehensible format, attuned to the way the students' minds work. Unfortunately, the actual profit of the resulting learning systems is largely reduced by poorly represented interactive objects as well as poor interlinking between such objects. In particular, such objects appear isolated: they neither can be modified sufficiently (e.g. by choosing parameters or enhancing functionality) nor be interlinked properly with their context (e.g. by synchronizing with a guided tour or metadata). Innovative enabling technologies like XML and wireless communication may for the first time provide a facility to interact with online applications anytime anywhere. In this article, we are presenting a prototype implementing a collaborative multimedia objects based on SceneBeans. The structure and behavior of the multimedia learning objects is described in XML using ScenBeans and hence requires no programming experience. The prototype illustrates our vision of linking such multimedia learning objects to a major learning object repository.

Keywords: *Collaborative Learning, Multimedia Learning Objects, SceneBeans.*

---

1. Introduction

As academic communities move from institutional classroom instruction to single-learner web-based distance learning, students risk losing critical opportunities to collaborate with other students. Yet research in education has shown that classroom learning improves significantly when students participate in learning activities with small groups of peers [1]. Students learning in small groups encourage each other to ask questions, explain and justify their opinions, articulate their reasoning, and elaborate and reflect upon their knowledge, thereby motivating and improving learning. They can bring different strengths and expertise to bear. Recognition of the educational value of student collaboration has led to the introduction of conventional groupware tools - such as chat, threaded discussions, and email - into distance-learning environments. While these tools can facilitate didactic interactions between learners, they cannot ensure productive learning dialogues between participants and they do not address how to provide as rich a learning environment for asynchronous students. We believe that facilitating problem-based learning between peers by having them solve real world problems is a key to providing effective web-based distance learning for both synchronous and asynchronous students.

Today's collaborative learning and problem-solving environments afford the opportunity to bring together different learners to jointly tackle a problem. A student in one location can connect over the web and interact with students in other locations. Current collaborative environments concentrate on providing communication between participants and tools to facilitate collaborative activities such as shared whiteboards and shared applications. As the use of collaborative environments becomes more ubiquitous, we can expect many of the same problems facing colleagues physically meeting together to arise in cyberspace. The use of ubiquitous computing will help more in the organization and mediation of interactions wherever and whenever these situations might occur [2]. However, the Learning Object (LO) model provides a framework for the exchange of learning objects between systems. If LOs are represented in an independent way, conforming instructional systems can deliver and manage them. The learning object initiatives, such as IEEE's LOM, Educom's IMS, or eduSource CanCore are a subset of efforts to creating learning technology standards for such interoperable instructional systems [3]. The LO content to be described is normally built of hypermedia elements (texts, images, audio, video, animations) which are stored in a modularized way. Actually all of the methods used to specify the LO metadata make use of metadata in the traditional sense of describing the hypermedia as static data. The usage of dynamic multimedia learning objects, such as animations, requires a new sort of metadata, which must be dynamic in order to facilitate the I/O behavior of a dynamic LO [4]. There is no universally acceptable way to represent such dynamic behavior and the only innovative standard was given by ElSaddik et al [4] in their *Smart LO model*, where the dynamics representation requires variety

[#] Accepted by the 9th Western Canadian Conference on Computing Education WCCCE, May 6,7, 2004, BC, Canada.

of tools (e. xLOM Editor, Content Customizer and Visualization Module) as well as a dedicated collaborative infrastructure (e.g. JASMINE)[5] . Our primary goal in this article is to adopt the model SceneBeans as an alternative standard for representing dynamic multimedia metadata and use it within a general-purpose collaborative and ubiquitous environment to test it affectivity.

2. Multimedia Content Packaging:

Designing structured multimedia authoring systems is a great challenge for the learning industry, which has to handle large amount of information. Several years ago, the notion of document classes were introduced for static documents in order to enhance document productivity and quality. With the advent of standards like XML and the increasing diversity of media types, there is also a need to have classes for multimedia documents. Typical examples of document classes are slideshow presentations, technical documentation or courseware. In addition, XML also allows the specification of the structure of document classes independent of their final presentation. Indeed, developing high quality educational multimedia content requires more than subject matter expertise: it demands programming skills, graphic and animation skills, instructional design skills and a significant investment of time. It is rare that a single faculty member or even a single department would possess these skills and resources. That is why open course collaboratories of faculty, staff and students have great promise as a source of shareable, high quality instructional materials. The new idea of learning objects and the multimedia learning objects follows such a constructivist view of learning where students/facilitators actively construct knowledge rather than are being taught centrally by a static core material. Several ubiquitous multimedia learning systems, such as *BSCW,* the *Notebook University* project, the *Courseware Watchdog, and* the *VEL* tried to solve the above issues using scalable media based on multimedia standards such as MPEG-4, MPEG-21 for animated type and or JPEG2000 for still images [3]. The use of scalable media allows retrieve different versions of contents from a single file and hence saves the burden of generating and handling one file for each required version. However, their use requires both the instructor and learner to be professional multimedia programmers in order to change any of these objects.

In the context of eLearning, content refers to items such as blocks of text, pictures, maps, diagrams, animations, video and audio recordings, software programs, XML, tests and answers, learner information, resource information, collaborative tools, etc. Content needs to be packaged because many eLearning contents are now being created in the form of learning objects, which need to be assembled in order to form a coherent eLearning course/concept. The aim of this section is to propose a general framework for multimedia document production through the specification of a learning object model using SceneBeans. SceneBeans can be adopted as a model for packaging dynamic multimedia objects in a form of Java Beans where its semantics metadata is based on textual XML format [4].

3. Defining Multimedia Objects using SceneBeans

Originally SceneBeans are introduced as a java component-based multimedia animation framework by Pryce Magee during 2002 [6]. SceneBean *animation* encapsulates a scene-graph and the behaviors that animate the nodes of that graph. It acts as the manager for the behaviors encapsulated within it, routing commands and events. Most importantly, a SceneBean animation is also a scene-graph node, since this means we can compose animations, applying transformation and further animation as required. SceneBeans conform to industrial standards: Java and XML, and because of its component nature, allows extensibility of the framework within its “domain-specific visual and behavioural components” [7]. Hence, the SceneBeans framework provides programmers with a convenient programming model for creating and controlling interactive multimedia objects, and a useful set of components that can be plugged together within that framework. Indeed, it is not practical to expect learners to write Java programs in order to define animations for example for use in applications, and even for experienced programmers the edit/compile/debug cycle is slow and frustrating when fine-tuning animation parameters are involved. To simplify the authoring/description process XML format has been used by SceneBeans which requires a parser to translate the XML document into interactive multimedia or Animation objects. The XML document type definition (DTD) used by the SceneBeans parser is relatively minimal compared to DTDs for similar applications, such as the W3C’s Scalable Vector Graphics (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.

A SceneBean object is described using XML. The top-level <animation> element 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 behaviours 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. Figure 1 shows the XML definition of a simple scene describing an animated traffic light. The animated example will turn the lights into red, then turn green, then turn yellow and return back to red.

```
<?xml version="1.0"?>
<!DOCTYPE animation SYSTEM "scenebeans.dtd">
<animation width="135" height="271">
        <behaviour id="show_red" algorithm="track" event="green_on">
        <param name="pointCount" value="3" />
        <param name="point" index="0" value="(32, 9)" />
        <param name="duration" index="0" value="5" />
        <param name="point" index="1" value="(32, 9)" />
        <param name="duration" index="1" value="0.01" />
        <param name="point" index="2" value="(-80, 9)" />
        </behaviour>
                <define id="redlight">
                <transform type="translate">
                <param name="translation" value="(-80, 9)" />
                <animate param="translation" behaviour="show_red" />
                <primitive type="sprite">
                <param name="src" value="images/red_light.gif" />
                </primitive>
                </transform>
                </define>
                <define id="yellowlight">
                <transform type="translate">
                <param name="translation" value="(-80, 9)" />
                <animate param="translation" behaviour="show_yellow" />
                <primitive type="sprite">
                <param name="src" value="images/yellow_light.gif" />
                </primitive>
                </transform>
                </define>
                <define id="greenlight">
                <transform type="translate">
                <param name="translation" value="(-80, 9)" />
                <animate param="translation" behaviour="show_green" />
                <primitive type="sprite">
                <param name="src" value="images/green_light.gif" />
                </primitive>
                </transform>
                </define>
        <draw>
        <paste object="redlight" />
        <paste object="yellowlight" />
        <paste object="greenlight" />
        <primitive type="sprite">
```

```
        <param name="src" value="images/traffic_light.gif" />
        </primitive>
        </draw>
                <command name="turn_red">
                <announce event="~red_light" />
                <announce event="~green_light" />
                <announce event="~yellow_light" />
                <reset behaviour="show_red" />
                <start behaviour="show_red" />
                </command>
                <command name="turn_yellow">
                <announce event="~red_light" />
                <announce event="~green_light" />
                <announce event="~yellow_light" />
                <reset behaviour="show_red" />
                <start behaviour="show_red" />
                </command>
                <command name="turn_green">
                <announce event="~red_light" />
                <announce event="~green_light" />
                <announce event="~yellow_light" />
                <reset behaviour="show_red" />
                <start behaviour="show_red" />
                </command>
        <event object="show_red" event="green_on">
        <announce event="green_light" />
        <stop behaviour="show_red" />
        </event>
        <event object="show_red" event="yellow_on">
        <announce event="green_light" />
        <stop behaviour="show_red" />
        </event>
        <event object="show_red" event="green_on">
        <announce event="green_light" />
        <stop behaviour="show_red" />
        </event>
</animation>
```

Figure 1: The XML Definition of the Traffic Light Animation.

The SceneBeans application interface loads and runs scene objects, allows the user to invoke commands on the animation and displays events announced by the animation (see Figure 2).

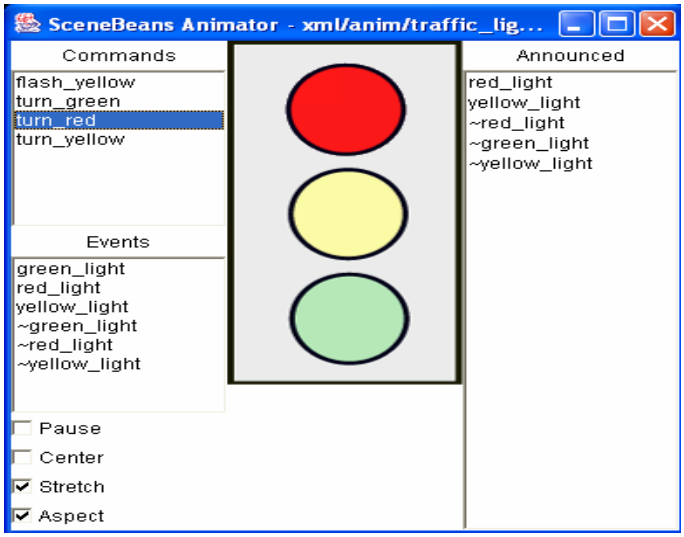


Figure 2: The SceneBeans Interface.

Once you create a scene object/graph, you apply an action to the root node to trigger the events prescribed in the scene graph nodes. Most nodes include an overwritten apply() method that triggers an appropriate response when a specific action is applied to the node. Actions include rendering the scene (draw action) and playing sound files (sound action). A top-down traversal can be use to traverse the sequence of actions used within any scene object Indeed, the Scene Object Hierarchy can be enhanced to include relations that associate animation activity commands and conditions with action labels as defined by the LTS method[8].

4. Developing a Collaborative Environment for SceneBeans:

Collaborative systems are becoming more and more as ubiquitous media for information exchange. Consequently, there is great demand for, and much research on information coordination and integration mechanisms among heterogeneous, distributed and dynamic information sources. In addition, researchers are beginning to focus on mechanisms that allow end users to participate in distributed information networks without much technical support and sophisticated computing platforms. Especially in the distributed educational system involving students and instructors who work autonomously, but regularly need to collaborate and exchange learning information. Such situation is served well by using a peer to-peer computing environment. This environment, popularized by systems such as Napster and Gnutella, views a distributed system as an open, dynamic network of *peers*. Each peer is *acquainted* with a small number of other peers with whom it can exchange information and services. Because of the ubiquitous nature of such collaborative architecture, student/instructor communication requires two different channels for exchanging information on dynamic multimedia objects (e.g. SceneBeans): the XML channel that can be used to display the definition of the multimedia objects as well as to convey the instant messages between the student and the instructor, and the SceneBean channel that transfers the Java Bean Jar file via the TCP/IP Channel which is used by most of the ubiquitous devices

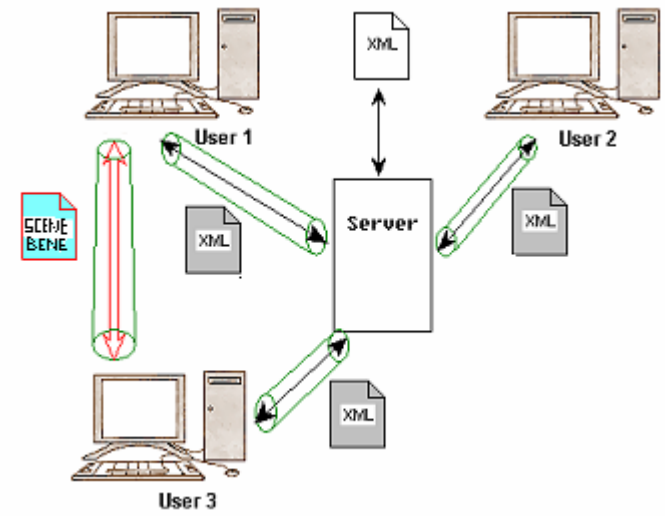
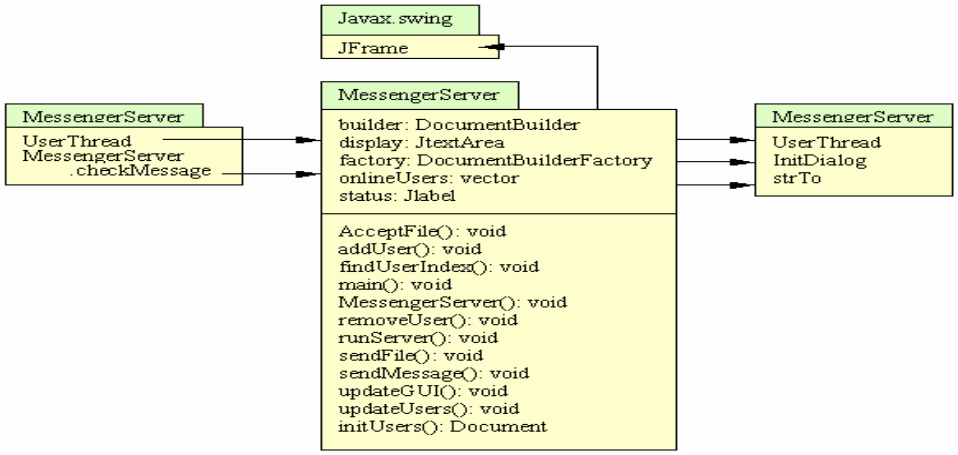


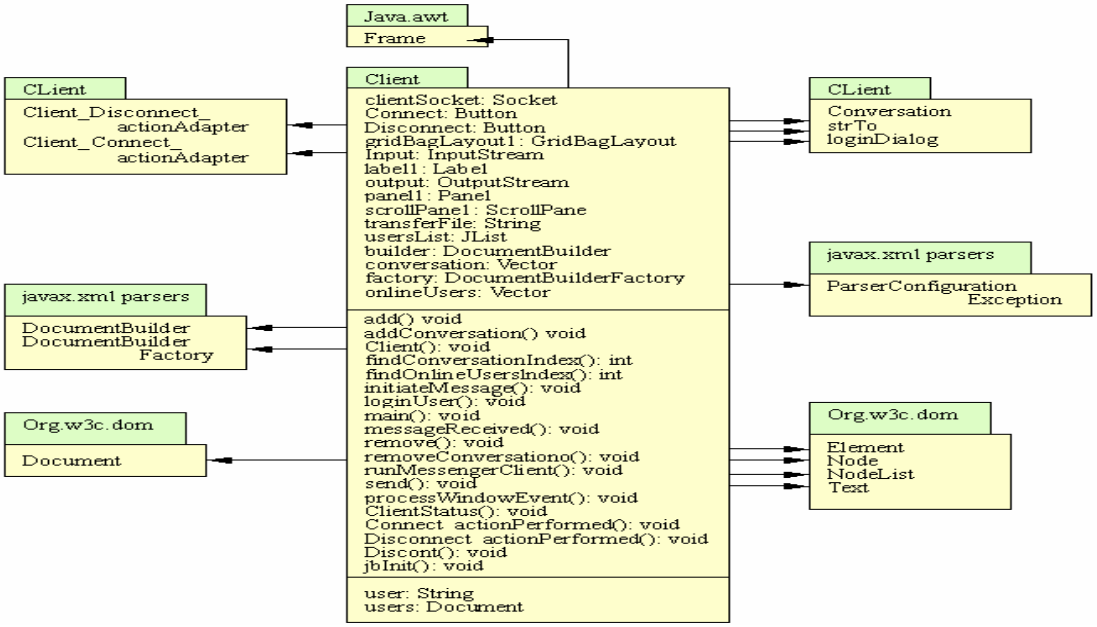
Figure 3: The General Architecture of the SceneBean Collaborative Environment.

The XML channel is based on Client-Server messenger mode [9]. Clients, or user of the system, are the people who need to communicate with other users in the system. Consider a case where user 1 needs to communicate with user 2. For communication to be established, both users need to be logged on to the server. The messenger displays all the users who are currently log on to the server. User 1 can then choose to send message to user 2. When user 1 types in a message for user 2, the message is tagged with XML and sent to the server. The XML message also contains the destination (User 2) of the message and its source (user 1). The server then reroutes the message to the respective user based on the destination information provided in the message. The XML server (MessengerServer) uses a ServerSocket, object to wait for clients to connect. When a client connects, the server creates a new UserThread object to manage the client's socket and streams. The MessengerServer object uses a vector to store all UserThread instances. The MessengerServer also uses a Document object users, consisting of the names of on-line users stored in individual user elements. The XML client has two main functions. First, it registers the user with the server by sending an XML document that

contains the user's name and ID. It then updates its current list of logged-on users with the new information it receives from the server. During the session, that is, the period during which the user is logged on, it has to update this list whenever a new user logs in. All such information is exchanged in the form of XML. The second function of the client is to convert the text typed in by the user into XML-based messages, tagging them appropriately to identify the source and destination of each message, and to send them to the server. The client also has to parse the XML messages received from the server and display them to the user. Figure 4 illustrates the UML Diagrams for the XML communication channel.



(a) The Server Classes of the XML Messenger



(b) The Client Classes of the XML Messenger

Figure 4: The UML Diagrams of the XML Messenger.

The SceneBean channel is based on Java Socket programming. However, there are two Java communication protocols that utilize socket programming: datagram communication and stream communication. Our Image transfer channel is built upon a stream socket programming. The stream socket is protocol that is based on TCP (transfer control protocol). Unlike UDP, TCP is a connection-oriented protocol. In order to do communication over the TCP protocol, a connection must first be established between the pair of sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once two sockets have been connected, they can be used to transmit data in both (or either one of the) directions. Creating a socket like Socket MyClient can be done simply using:

MyClient = new Socket ("Machine name", PortNumber);

Where Machine name is the machine you are trying to open a connection to, and PortNumber is the port (a number) on which the server you are trying to connect to is running. When selecting a port number, you should note that port numbers between 0 and 1,023 are reserved for privileged users (that is, super user or root). These port numbers are reserved for standard services, such as email, FTP, and HTTP. When selecting a port number for your server, select one that is greater than 1,023! The programming techniques for this type of messenger are well-known and we refer the reader to [10]. In case of user1 wants to send an Image File to user2, user1 must use the collaborative infrastructure GUI and press "Send SceneBean" to the user name of User2. Before sending the actual image file, the user needs to use the XML messenger to notify the other user.

<TransferRequir to = "receiver" from = "sender">
Waiting for file transfer: (filename) </TransferRequir>

If user2 accept the file transfer, an accept message will be created and sent back to user1:

<Accept to = "receiver" from = "sender"> (IP address of user2) </Accept>

At the same time user2 create a SeverSocket and waiting for incoming connection. User1 can get the IP address of user2 from the above message. Then user1 create a Socket and connect to user2 by the IP got from message. After all above success, file transfer will start. Each side will be acknowledged when file transfer finished. When server redirects it to user2, the button "receive file" will be enabled. User2 can choose to receive file or ignore it. Figure 5 shows the SceneBeans collaborative environment main screen.

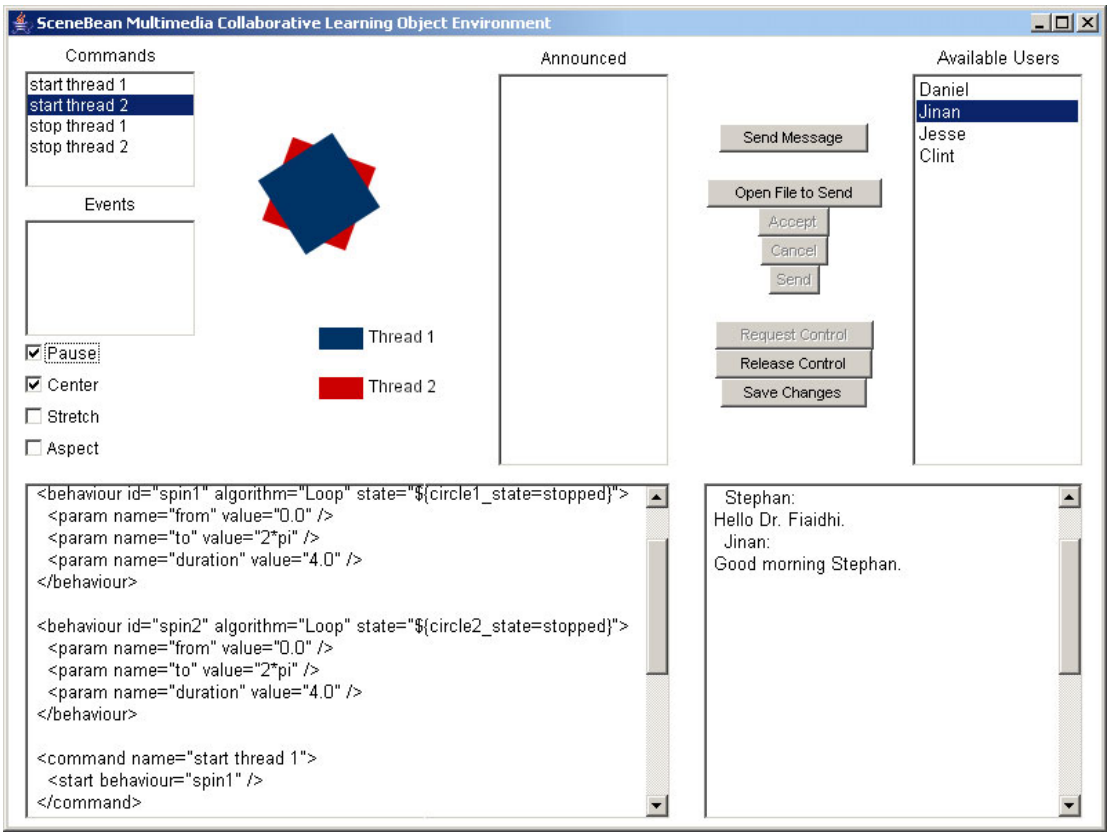


Figure 5: The SceneBeans Collaborative Environment.

5. Conclusions:

In this article we developed a collaborative tool to exchange dynamic multimedia learning objects based on the notion of SceneBeans. SceneBeans describe multimedia objects metadata in XML. The SceneBean metadata does not directly comply with the accepted standards. We are currently involved in translating the SceneBean description into the Canadian Core Learning Resource Metadata Protocol (CanCore). CanCore was developed in Phase I of the POOL project by the collaboration of Canadian researchers searching for a level of sufficient specificity to enable the efficient search of learning objects. CanCore is based on IMS implementation of IEEE. CanCore has sufficient flexibility in its protocol that not all fields need be completed, thus developers can ignore many fields that may be inappropriate for their purposes. The POOL protocol expands the JXTA P2P communication protocol by building in more control for distributed searches and provides for flexibility in metadata schemas used for queries and responses [11]. For this purpose we are proposing to use the Sun Microsystem JXTA to link to the POOL project and to extend our collaborative environment to utilize the CanCore protocol as well as to have access to its other repositories. The proposed meta structure for such extension is illustrated in figure 6.

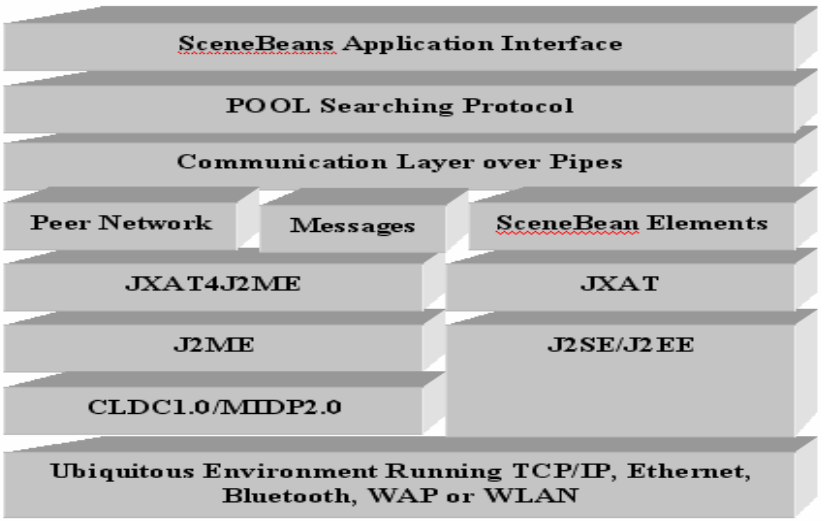


Figure 6: The Proposed Middleware Architecture.

References

[1] W. Richard and D. Oblinger, The Next Generation Student, Higher Education Leaders Symposium, Redmond, Washington, USA, June 17-18, 2003.

[2] H. Ogata and Y. Yano, Knowledge Awareness Map for Computer-Supported Ubiquitous Language-Learning, IEEE WMTE2004, Taiwan, March 23-25, 2003.

[3] IEEE Learning Technology Standards Committee (LTSC) IEEE P1484.12 *Learning Objects Metadata* Working Group http://ltsc.ieee.org/wg12

[4] Abdulmotaleb **El Saddik**, Amir Ghavam, Stephan Fischer, and Ralf Steinmetz. "Metadata for Smart Multimedia Learning Objects". In Proceedings of the fourth Australasian Computing Education Conference. ACM-CSE, Melbourne, Australia, December **2000**.

[5] Abdulmotaleb **El Saddik**, Dongsheng Yang, and Nicolas D. Georganas. "A Lightweight Multi-session Synchronous Multimedia Collaborative Environment". In Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications. Tunisia, Tunis, July 2003.

[6] N. Pryce and J. Magee, SceneBeans: A Component-Based Animation Framework for Java, Technical Report, Department of Computing, Imperial College, June 2001.

[7] J. Magee, J. Kramer, B.Nuseibeh, D.Bush and J. Sonander, Hybrid Model Visualization in Requirements and Design: A Preliminary Investigation , Proceedings of 10th International Workshop on Software Specification and Design (IWSSD-10), 5-7 November 2000, San Diego, USA.

[8] J. Magee and J. Kramer, Concurrency-State Models and Java Programs, John Wiley & Sons, March 1999.

[9] H.M. Deitel, P.J. Deitel, T.R. Nieto, T.M. Lin, P.Sadhu, *"XML how to program"*, Deitel & Associats Inc, 2001.

[10] Qusay H. Mahmoud , Sockets programming in Java: A tutorial, JavaWorld Online Journal, December 1996, http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets_p.html
[11] M. Hatala and G. Richards, POOL, POND and SPLASH: A Canadian Infrastructure for Learning Object Repositories. 5th IASTED Int. Conference on Computers and Advanced Technology in Education (CATE 2002), Cancun, Mexico, May 2002.
Acknowledgment: The authors would like to acknowledge the help of Dan Gaudette on SceneBeans.

Dr. Jinan FIAIDHI, Associate Professor,
Dr. Sabah Mohammed, Associate Professor,
Mr. Stephan SISKO, Graduate Student,
Department of Computer Science, Lakehead University, 955 Oliver Rd., Thunder Bay, ON P7B 5E1
Tel: 1 807 3438224 Fax: 1 807 3438023 Emails:{jinan.fiaidhi,sabah.mohammed, sjsisko}@lakeheadu.ca

---