preprint. International Journal of Continuous Engineering Education and Life-Long Learning, Vol. 12, Nos. 1-4, pp. 299-317, 2002 (23% acceptance rate). Special issue on Intelligent Agents for Education and Training Systems.

Building A Next-Generation Infrastructure for Agent-based Distance Learning

Junichi Suzuki

Department of Information and Computer Science
University of California , Irvine
Irvine, CA 92697-3425, U.S.A.
949-824-3097

jsuzuki@ics.uci.edu

Yoshikazu Yamamoto

Department of Computer and Information Science,
Keio University
Yokohama City, 223-8522, Japan
+81-45-563-3925

yama@cs.keio.ac.jp

Abstract

The emergence of the Internet has radically changed the way in which we learn, teach and train. This paper proposes an integrated and extensible architecture for agent-based distance learning. The architecture provides component-based extensibility, allowing emerging technologies to be plugged-in, so that they can produce synergy. It provides HTTP and IIOP connections for maintaining and delivering courseware to students. Via the HTTP connection, our Persona system provides the personalization service to each set of courseware, allowing it to customize its content and/or presentation context-sensitively. This service facilitates the effective delivery of courseware. Via the IIOP connection, our SoftDock system provides the foundation facility allowing users to work collaboratively in teams. Our current educational domain is software modeling. Participants learn the basic concepts and principles of software modeling, and then leverage their newly-acquired modeling skills. We believe our work provides a blue print for showing how emerging technologies can be applied to practical distance learning applications.

Keywords

Distance learning, Internet-based learning, Web-based learning, Collaborative training, Intelligent user interface, Agent computation, Personalization, XML, DOM, CORBA

Biographical Notes

Junichi Suzuki received his B.S., M.S. and Ph.D. degrees in computer science from Keio University, Japan. He is currently a post-doctoral scholar at the Univeristy of California, Irvine. His research interests include highly distributed systems, adaptive communication systems, autonomous decentralized agent computing, and biologically-inspired software architecutres. He is a member of ACM, IEEE Computer Society, IEEE Communications Society, IPSJ and JSSST.

Yoshikazu Yamamoto received his B.S., M.S. and Ph.D. degrees in administration engineering from Keio University in Tokyo. He is currently an associate professor of the Department of Computer and Information Science at Keio University. He worked at Linkoping University, Sweden as a visiting professor from 1981 to 1983. His current research interests include distributed discrete event simulation and modeling, OOP, agent programming, intelligent interface and documentation. He is a member of ACM, IEEE-CS, IPSJ and also director of the board of JSSST.

1. Introduction

1.1 Background

The emergence of the Internet has great significance for distance learning and training, as it is an effective and economical medium for making information available to dispersed individuals. It has radically changed the way in which we learn, teach and train. It has also altered the way in which learning resources are developed. Distance learning at “Internet speed” requires less time for preparing courses and more frequent updates of courseware, and allows just-in-time delivery of this courseware to students anywhere, at any time, while maintaining high levels of functionality and quality.

This paper proposes a flexible architecture for Internet-based distance learning using an agent-mediated mechanism to create and deliver educational materials effectively. Our work makes advances in the areas of:

- Effective delivery of courseware to students
- Consistent remote editing of courseware by educators and/or training participants
- Hardware and software heterogeneity

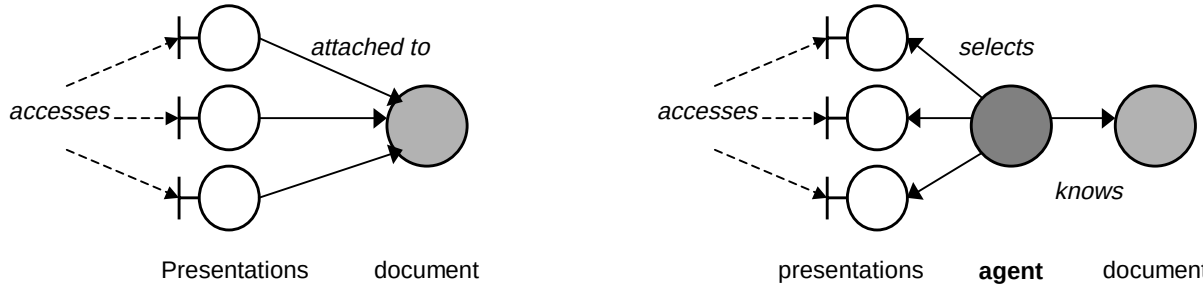

Figure 1: In the traditional manner, multiple presentations are attached to a single document. Users access the document through a selected presentation (left). In Persona, an interface agent intermediates between a document and its presentations (right). It adapts the presentation for a specific user.

1.2 Our approach

Our system addresses the above challenges by using interface agents and open standard technologies. To address the first issue, it applies the document personalization service to all courseware, customizing it according to the context in which it is accessed by students. This facility is provided by our Persona toolkit, an agent-based engine that allows web documents (i.e. HTML and XML documents) to be mined for their appropriate content and/or presentation in a given context [1].

To address the second issue, we developed an agent management framework to track and fetch remote documents on demand. It is used for distributed synchronous or asynchronous collaborative work. The framework is layered on top of our SoftDock system, which manages XML documents in a distributed environment [2].

To address the third issue, our system uses CORBA (Common Object Request Broker Architecture) [3], a middleware specification which allows distributed applications developed with different languages to interoperate with each other across different platforms. It allows educators and learners to participate in a course without forcing them to use any specific hardware or software.

The system's focus is online self-paced learning and group-paced training which are conducted by universities providing online courses for their students or distance learners, and private organizations that wish to keep their workforces well trained. The target educational domain is software modeling. The courseware consists of online courses for teaching the concepts and principles of software modeling, and exercises designed to developed modeling skills. The self-paced learning courses are intended to teach general software modeling techniques for novice learners. The group-paced training courses are intended to be used in a team setting, where participants collaboratively model software structure and behavior. They can be used by universities for student's semester projects or by organizations for development team training. Team members can share their knowledge, practical techniques and heuristics gained through experience, so members continue to benefit even though they are already high-performing experts. This course is intended to help the development teams gain skills in distributed software development.

Our courses use the Unified Modeling Language (UML) [4] as the modeling language. UML is a standard object-oriented modeling method that provides most of the semantics and their notations required for representing software constructs, and has been widely accepted by academic and commercial developers. Our courses also use an XML-based format for describing and exchanging UML models, called UXF (UML eXchange Format) [5, 6]. Note that due to space limitations, the basics and benefits of UML, UXF and XML are not covered here. Please see [5, 6] for more depth discussion.

In this paper, we describe how new technologies including agent technologies and emerging standards can be applied to distance teaching and learning. Leveraging these technologies to structure the courseware and learning environment can yield significant benefits.

The remainder of this paper is organized as follows: Section2 overviews the foundation technologies in our system. Section 3 proposes our architecture for distance learning and describes our system. We conclude with a note on the current project status and future work in Section 4 and 5.

2. Foundation Technologies

This section presents an overview of the technologies that provides the foundation of our system.

2.1 Document Personalization

One of the current issues in managing web documents is context-sensitive customization or personalization of documents for different users or users' behavior. Up until now, every document has had only one presentation and

content across its entire readership. Obviously, a better solution is to transfer documents such that their presentations and/or contents are tuned for the reader. Such a capability is called *personalization* [7]. Personalization is considered the next logical step in the evolution of the Web. Our system provides this capability for educational materials.

2.2 Persona and Agent Technology

Our vehicle for researching personalization is the Persona toolkit [1, 7, 8]. Persona is a personalization engine for HTML and XML documents. It inspects the contextual information and dynamically generates an appropriate document best-tuned to the context by re-authoring the content and/or applying a stylesheet. Persona allows web documents to be dynamic and active without inconsistency. Their contents and presentations are determined at runtime. This contrasts with static traditional documents where a document has a single content/presentation (Figure 1). Using Persona, educators need only prepare a single source document. Then, defining policies for re-authoring content and applying stylesheets. This increases the educator's productivity.

The process of selecting an appropriate content and presentation for a given document is performed by a *document agent* (right of Figure 1). A document agent exists for each document. It knows the metadata (or document profile) of a corresponding document. Section 2.3 describes document metadata, and Section 3 presents how the agent works. A *user agent* asks document agents to personalize courseware on its behalf. It represents each user and maintains user metadata (or user profile). It decides when and how to personalize courseware. Section 2.3 describes user metadata, and Section 3 presents how the document and user agents cooperate.

The personalization service requires intelligent user interface and content management. We have used agent technology to provide this intelligence. Conceptually, a software agent is a computational entity that:

- acts on behalf of its users or other entities in autonomous fashion (agency or autonomy).
- behaves with some degree of proactivity and/or reactivity.
- exhibits some degree of learning, cooperation and mobility.

Agents are expected to know users' interests and to act autonomously to meet their goals without their supervision. Users can delegate a task to an agent rather than explicitly ordering the agent to perform it [9]. Recent research has identified three promising application domains: *intelligent interface agents*, *distributed agents* and *mobile agents*

```
<RDF
 xmlns="http://www.yy.cs.keio.ac.jp/~suzuki/RDF"
 xmlns:SoftDock="http://www.yy.cs.keio.ac.jp/~suzuki/SoftDock"
 xmlns:IMS="http://www.yy.cs.keio.ac.jp/~suzuki/IMS"
 xmlns:Persona="http://www.yy.cs.keio.ac.jp/~suzuki/Persona">
 <!-- Description about a course -- >
 <Description about="http://www.yy.cs.keio.ac.jp/cources/uml001">
  <SoftDock:Educators>
   <Bag>
    <li resource="softdock://people/yy"/>
   </Bag>
  </SoftDock:Educators>
  <SoftDock:Learners>
   <Bag>
    <li resource="softdock://people/jsuzuki"/>
    <li resource="..."/>
   </Bag>
  </SoftDock:Learners>
  <SoftDock:Materials>
   <Sequence>
    <li resource="softdock://cources/uml001/intro.xml"/>
    <li resource="..."/>
   </Sequence>
  </SoftDock:Materials>
  ...
 </Description>
 <!-- Description about a user -- >
 <Description about="softdock://people/jsuzuki">
  <Persona:Name>jun</Persona:Name>
  <Persona:Preference>
   ...
  </Persona:Preference>
  ...
 </Description>
 <!-- Description about a material -- >
 <Description about="softdock://cources/uml001/intro.xml">
  <IMS>
   <IMS:Category>
   ...
   </IMS:Category>
  </IMS>
  <Persona:Persona>
   <Persona:Presentation format="XSL"
                         src="table.xsl">
    <Persona:Name>table</Persona:Name>
   </Persona:Presentation>
   ...
  </Persona:Persona>
 </Description>
</RDF>
```

Figure 2: Sample metadata description with RDF

[10]. Persona provides interface agents, i.e. user and document agents, for personalizing web documents.

An interface agent operates in an autonomous fashion. It may observe user interface actions and make changes to interface objects displayed on the screen [11]. Interface agent is required to be aware of user metadata, as well as its application domains or environment in which it operates [12].

2.3 Metadata Management

Metadata is data about data, or specifically descriptive information about learning resources, e.g. educational materials, educators, learners, etc., in the context of our system. It is used to index, retrieve, manage, interchange or automate learning resources. The concept of metadata is becoming important in the web community. Resource Description Framework (RDF) [13] has been proposed as a recommendation by the World Wide Web Consortium

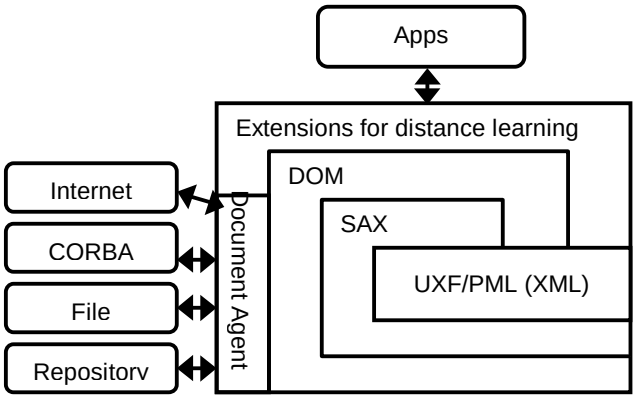

Figure 3: Our system architecture

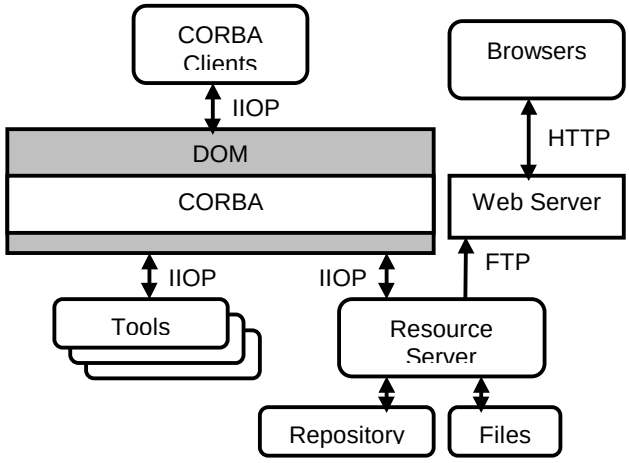

Figure 4: Current system organization

(W3C). RDF allows us to define metadata of arbitrary web resources including an entire web document, a part of a document, a collection of documents and even an object that is not directly accessible via the web.

In the area of distance learning, the IMS project [14] is in the process of standardizing the IMS Meta-Data specification [14, 15]. The goal of IMS Meta-Data is almost the same as that of RDF, but IMS Meta-Data is more specific for describing metadata of educational resources. RDF is designed as a XML-based format, while IMS Meta-Data is not constrained to XML.

In our system, we use RDF to define metadata of the overall structure of learning courses. Figure 2 shows a sample RDF description. It defines metadata of a course, a person and an educational material. The course description specifies the course name, educators, learners, materials used, etc. The section of user metadata defines every user including his/her name, selected courses, progress of courses, access history, preferences and so on. The section of material description defines document metadata based on the IMS Meta-Data specification and document's available presentations. Our system uses about the half elements defined in the IMS Meta-Data: some elements of the General and Characteristics sections, and the most elements of Life Cycle, Technical and Educational Use Dependent sections.

2.4 SoftDock and Resource Management

The foundation of our system is constructed with the SoftDock system that we have provided [2, 16]. SoftDock is a distributed model management system for UML. It allows developers to share UML model information using UXF and PML (Pattern Markup Language). PML is an XML-based description language for software patterns [17]. A software pattern represents a recurring solution to a software development problem within a particular context [18]. A pattern identifies the static and dynamic collaborations and interactions between software components. It leverages design reuse by documenting design heuristics including typical decision-makings and trade-offs. In general, applying patterns to complex applications can significantly improve software quality, increase software maintainability and support broad reuse of components and architectural designs [17].

SoftDock interchanges model information with UXF and PML through the Document Object Model (DOM) interface [19] implemented on CORBA (Common Object Request Broker Architecture). DOM defines a general-purpose interface to manipulate parsed tree structures of XML documents. It provides a set of APIs for the following capabilities:

- Structure navigation, which is the navigation of document structures such as accessing and searching elements or attributes
- Structure manipulation, which is the manipulation of document structures such as adding, changing and removing elements or attributes
- Content manipulation, which is the manipulation of document contents such as putting or getting values to elements and attributes

CORBA is a standard for object middleware in the heterogeneous environment. It provides a standard way to interoperate distributed objects. CORBA defines a series of interfaces and components that organize an Object Request Broker (ORB). An application that needs to access the services of a remote object uses an ORB to send messages and receive results. The CORBA allows to distribute objects on multiple platforms in a seamless and transparent manner to applications. One of the key components in CORBA is the OMG Interface Definition Language (IDL), a language to define the interface of a remote CORBA object. It is programming language neutral by providing mappings from IDL to various languages. Another important component is the Internet Inter-ORB Protocol

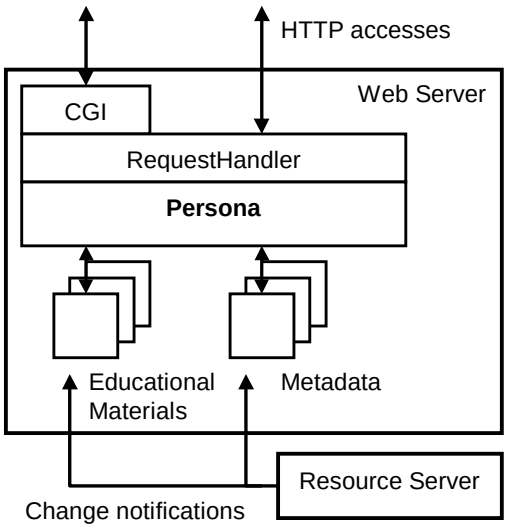

Figure 5: Educational materials are delivered using Persona via the HTTP connection

(IIOP), which is a standard on-the-wire protocol based on TCP. IIOP leverages the interoperability between different ORBs as well as across a single ORB. CORBA provides a rich set of services including naming, event notification, transaction, security, life cycle, A/V streaming.

Our distance learning system is developed by extending the SoftDock. system and integrating Persona with its architecture. Section 3 describes the architecture and system organization based on Persona and SoftDock.

2.5 Benefits of Using New Promising Technologies

As described above, our system uses XML for describing various information associated with online courses, and DOM/CORBA for sharing and interchanging them. Currently, several hundreds of XML compliant tools exist as free and commercial products. The DOM compliant tools are now about 30. There are over 70 CORBA complaint ORBs, which support about 15 languages on over 30 platforms from handheld computers such as Windows CE and PalmPilot to mainframes. Using these widely-accepted standards ensures the interoperability between tools that educators and students use. They can choose and replace their tools that are compliant to these standards, without relearning particular tools when they change from one environment to another. This increases the interchangeability of courseware and productivity of both educators and students by reusing materials between tools and between courses. In addition, open standards reduce the strategic and political risks to develop and maintain the distance learning system.

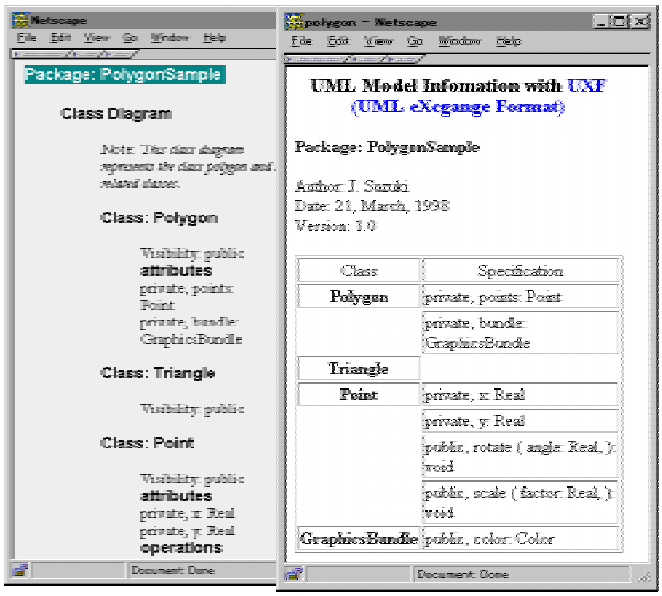

Figure 6: Sample textual (left) and table (right) presentations

3. System Overview

This section presents our extensible architecture and its deployment.

3.1 Architecture

Figure 3 shows our system architecture, which describes the relationships between different APIs to access educational materials. The UXF and PML descriptions are manipulated through either DOM or SAX (Simple API for XML) [20]. SAX is a de-facto interface between a XML parser and its applications, which has been developed in the XML community. The SAX is an event-based parser interface, while DOM is a tree-based [21]. Some DOM compliant parsers encapsulate SAX-based parsers. Any parsers supporting either DOM or SAX can be plugged into our framework without affecting other components in the system. The extension part in the architecture provides a series of utility objects that are useful for building various distance learning applications (see Figure 3). One of the most important objects in this part is `DocumentAgent`. Its responsibilities are:

- Fetching the content of a document.
- Personalizing the presentations and/or contents of a document.
- Connecting the DOM interface with external environments such as the Internet, CORBA, file systems and repositories.

The design of `DocumentAgent` is described in Section 3.3.

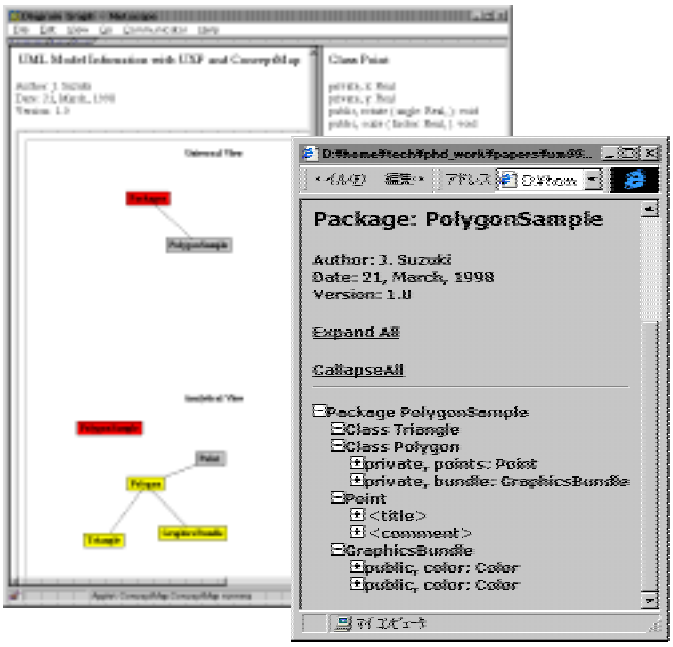

Figure 7: Sample presentations using a Java applet (left) and DHTML (right)

Figure 4 depicts the current system organization. Course documents including UXF and PML descriptions are stored in a resource server. The server is accessed from a web server and CORBA environment. The HTTP connection aims to allow client applications such as web browsers to refer the educational materials stored in a web server. Whenever a material is updated, the resource server pushes it to an appropriate web server. The IIOP connection aims to allow learners to refer, create and modify arbitrary materials. Client applications include web browsers, program editors, CASE tools, graphical profiling tools, documentation tools, metrics tools, etc. The HTTP connection is basically used for self-paced learning, while the IIOP connection is for group-paced collaborative training. In the following sections, Section 3.2 describes the former, and Section 3.3 describes the latter.

3.2 Self-paced Learning via HTTP

The self-paced learning is provided by broadcasting and navigating educational materials via the HTTP connection. As depicted in Figure 5, Persona is deployed in the back-end of a web server. We are currently using it on our reflective web server, OpenWebServer, which is extensible and configurable for various requirements [22, 23, 24]. Persona recognizes the user and client-side environment information using an incoming HTTP request, and then delivers an appropriate material. It provides two levels of personalization services. The first personalization involves generating a HTML document from a XML-formatted courseware by applying the best-suited XSL stylesheet in a given context. The second one is performed by rearranging the content and/or presentation of the generated HTML document.

When a user accesses a Persona-enabled web server, the user information is passed to the web server with cookie [25], explicit URL or CGI. With a cookie, user information is embedded in a HTTP header and passed from a web browser to a web server. A sample header is like:

```
Cookie: PersonaUserName=jun
```

The user information can be also transferred with an explicit URL:

```
http://.../test.htm?PersonaUserName=jun
```

Or, it may be passed through CGI:

```
http://.../persona.cgi?PersonaMaterial=
test.htm?PersonaUserName=jun
```

Persona finds the user agent corresponding to the user name passed from a web browser. Then, the user agent retrieves the user's metadata to inspect his/her progress of courses, preferences, etc. It chooses the courseware that follows the one the learners used last time. The target courseware is retrieved from a set of plain documents or external document repositories (see Figure 3 and 4). The user metadata is written in RDF, as described in Section 2.3, and maintained as a parsed tree structure in memory. User agents access the RDF tree structure via the DOM interface.

The user metadata is created by educators and learners explicitly. Educators create the metadata entry for every learner with RDF (see also Figure 2), and learners input the information about themselves through a HTML form in the process of course registration. After the registration, a web server creates a cookie about the user by passing a HTTP header to the user's browser like:

```
Set-Cookie:        PersonaUserName=jun;
path=/; expires=DATE
```

By default, Persona sets the one month later as an expiration date.

As for initializing the user metadata, there are some ways in general (based on [26]):

- Explicit installation
- Initialization using user characteristics
- Collaborative filtering initialization with the user's participation
- Initialization using examples
- Initialization by observing the user's behavior

Our current system basically creates the user metadata in the first way. User agents can refine its metadata, for example access history or preferences, by watching their behavior.

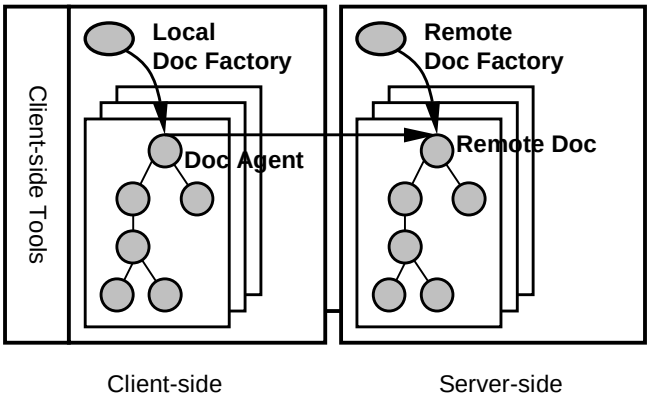

Figure 8: System organization for the IIOP connection

When a user agent forwards courseware to a client, it obtains the corresponding document agent to personalize its presentation. It asks the document agent to choose an appropriate presentation by applying a XSL stylesheet to a target document. A set of applicable XSL stylesheets for a document is defined in its document metadata, as described in Section 2.3 (see also Figure 2). A document agent chooses the most appropriate XSL definition to produce a HTML formatted document, by referring the user's preference and history of XML selection. Figure 6 shows sample textual and table presentations that are generated from a single material (i.e. UXF description) with different XSL stylesheets.

As described above, a document agent can customize the generated HTML document further in order to insert additional contents to the document, and to display the document best-tuning the client-side environment. When learners use a mailing list to discuss with an educator or each other, emails in the list are retrieved and inserted in a browser's additional frame. Also, the results from previous online quizzes or reports may be inserted.

In order to customize documents so that they are displayed best-tuned to the client environment, document agents inspect the client-side environment information such as the resolution, color/grayscale of display, extent of the current window, Dynamic HTML enable/disable, Java enable/disable and operating system. This information is obtained by JavaScript running on a browser. When Persona accepts a request for courseware through a web server (see Figure 5), it sends back a dummy HTML document containing the JavaScript script to inspect the client-side information. Then, the script automatically requests the target document again using the URL that embeds the environment information. A sample URL description is like:

```
http://.../index.htm?resolution=800*600
```

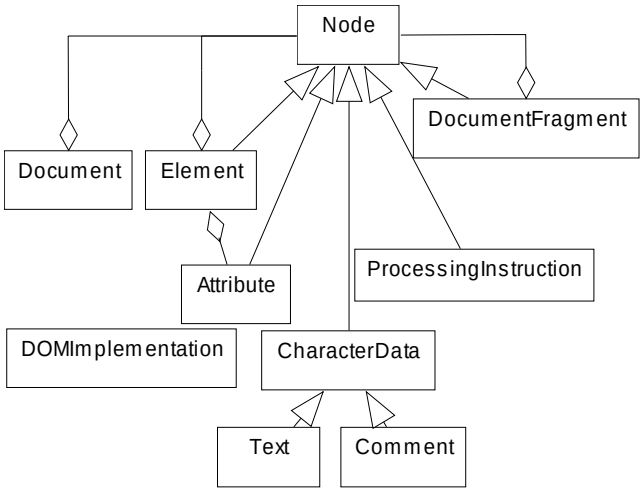

Figure 9: The essential objects in the DOM Core specification

Users can explicitly specify the environment information within a URL instead of the above automatic manner using JavaScript. This approach can be taken in the situation where a leaner uses JavaScript-disabled browser.

Figure 7 shows sample presentations that Persona transforms a material using the above capability. A Java applet presentation (left of Figure 7) helps learners to browse the software model in the graphical and intuitive manner. The DHTML-based presentation (right of Figure 7) allows the content to stretch and shrink when learners point headers with mouse. These presentations are generated when the client browser is Java-enable and DHTML-enable respectively. The preference of these presentations is recorded in every user metadata.

Persona can also re-author HTML documents according to the client-side resolution. It provides the following re-authoring policies by default [8]:

- Outlining
- Only first sentence
- Image reduction and elision
- Device-specific

The outlining policy transforms documents so that the content of each section is elided and every section header is converted into a hyperlink to the content. The converted document seems to be a table of contents. This strategy can be applied well for structured documents such as technical papers and manuals. Persona extracts HTML header tags, from H1 to H4 tags, and outlines the document.

The only first sentence policy transforms documents so that the content of each section is elided except the first sentence. This can be co-used with the outlining policy.

```
#include <dom.idl>
module SoftDockExtention {
  interface UXFDescription
  :dom::Document,
  :CosEventComm::ConsumerAdmin,
  :CosEventComm::SupplierAdmin {
    readonly attribute float revision;
    void externalize();
    sequence<string> content();
    boolean isLocked();
    oneway void lockNode(in dom::Document doc,
                         in CorbaDocAgent agent);
    oneway void releaseNode(in dom::Document doc);
  };
  interface CorbaDocAgent
  :dom::Document,
  :CosEventComm::ConsumerAdmin,
  :CosEventComm::SupplierAdmin {
    readonly attribute dom::Document remoteDoc;
    oneway metadata(in sequence<string> mdata);
  };
  interface CorbaDocFactory {
    dom::Document createDocument();
    dom::Document cloneDocument(
                              dom::Document doc);
    oneway void releaseNode(in dom::Node node);
    void destroyDocument(in dom::Document doc);
  };
};
```

Figure 10: IDL extension

The image reduction and elision policy transforms images in a document using pre-defined scaling factors, and then makes a hyperlink from the reduced image to original one. Persona can reduce images with 75%, 50%, 25% and 0% scaling factors. 0% means the image elision. An elided image is replaced with the text of its ALT attribute. The above three strategies are used to display the courseware on a hand-held PC that provides the pen-based input method.

The device-specific policy transforms documents so that they are displayed on a certain device. We are using this policy to display the course information on a cellular phone, which has a very limited memory buffer. Learners can browse the syllabus, schedule and progress of their courses.

In our system, educators and learners have to specify when and how the re-authoring policies are used in advance. Learners may specify a policy within a URL explicitly like:

`http://.../index.htm?policy=outlining`

Persona allows educators and learners to define a new re-authoring policy.

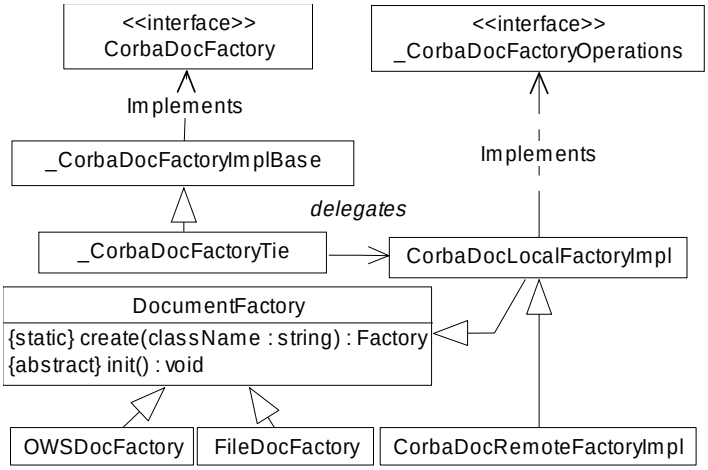

Figure 11: DocumentFactory and its subclasses

3.3 Group-paced Training via IIOP

The group-paced training is basically provided with the IIOP connection (see Figure 3 and 4). It allows the complete two-way communication between clients and servers. Course participants collaboratively create and refine software models of their deliverable. They can work in either synchronous and asynchronous mode.

Our IIOP-based system is organized as shown in Figure 8. A client-side application accesses a remote courseware using a *document agent*, which is the client-side equivalent of a server-side document. It serves as a local proxy or cache of a remote document. It is different entity from a document agent in the Persona system. The local and remote factory objects are responsible for managing the lifecycle of local and remote documents respectively.

The essential objects that consist of the DOM Core specification are depicted in Figure 9. The DOM objects are defined with CORBA IDL (see Section 2..4), and represent common structure of XML documents. The `Document` object has a recursive aggregation relationship to `Node`. This means an instance of `Document` is the root of instances of its any subclasses. Our system uses only the DOM Core interface, instead of the DOM HTML interface, because our system processes only XML documents.

Our system defines an extension IDL module so that it can handle the XML materials, i.e. UXF/PML descriptions, on CORBA, because DOM is not designed to run on CORBA originally. Figure 10 shows three extended interfaces: `UXFDescription`, `CorbaDocAgent` and `CorbaDocFactory`. `UXFDescription` is a remote document shown in Figure 8, and represents the server-side root element of an entire UXF document. It is derived from `dom::Document`, which provides the primary access to the document's data and defines factory methods needed to

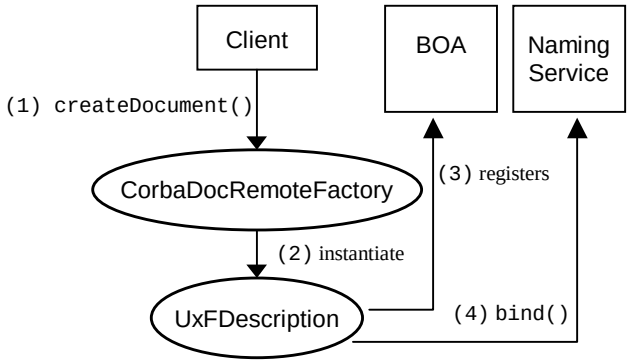

Figure 12: Server-side typical behavior in creating remote document

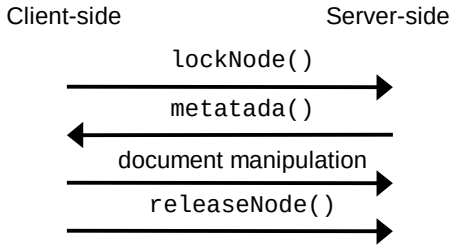

Figure 13: Client-server Interactions in the synchronous editing mode

create objects inside its context, e.g. `Element`, `TextNode`, `Comment`, `ProcessingInstruction`, etc (see Figure 9). It is also derived from `CosEvent::ConsumerAdmin` and `CosEvent::ConsumerAdmin`, which are defined in the CORBA event service. These interfaces are used to notify the change events between CORBA objects. A document change notification is transmitted between `UXFDescription` and `CorbaDocAgent`. `UXFDescription` has an attribute `revision` to track its revision number and exposes four methods. `externalize()` externalizes a document's tree structure into a file, and `content()` returns its representation as a sequence of string data. `lockNode()` and `releaseNode()` are used to require and release a lock of the document passed with the argument. The `CorbaDocAgent` interface is a client-side document agent shown in Figure 8. It has a reference to a remote document, and a method to obtain metadata about a certain remote document. The `CorbaDocFactory` interface represents local and remote factory objects described above.

The DOM specification does not define the way of creating a document instance, and therefore `CorbaDocFactory` provides factory methods for creating, cloning and destroying every document. The typical time a client should use these factory methods is when it creates an

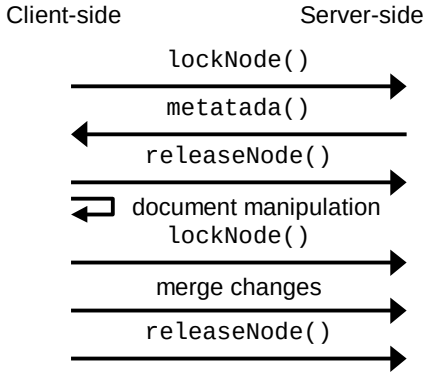

Figure 14: Client-server Interactions in the asynchronous editing mode

initial document. The `CorbaDocFactory` interface is translated to a series of objects by an IDL-Java compiler. Currently we are using JavaIDL included in Java2 and ORBacus for Java [27]. Figure 11 shows the objects generated by the IDL compiler of JavaIDL. The compiler automatically generates `CorbaDocFactory` and objects whose name has the prefix "_". `CorbaDocFactory` is an interface class that is a Java-side equivalent of the `CorbaDocFactory` IDL interface. `_CorbaDocFactoryImplBase` and `_CorbaDocFactoryTie` are the stub and skeleton class for `CorbaDocFactory` respectively. Our system employs the *tie approach* that a skeleton delegates the method invocations to its implementation class, instead that an implementation class derives from its skeleton. `CorbaDocLocalFactoryImpl` and `CorbaDocRemoteFactoryImpl` are implementation classes for `CorbaDocFactory`. `CorbaDocLocalFactoryImpl` is the factory object for the client-side copies of remote documents (see also Figure 8). `CorbaDocRemoteFactoryImpl` manages the server-side documents.

Figure 12 shows a typical server-side behavior in creating a remote document. When `CorbaDocRemoteFactoryImpl`'s `createDocument()` is called, it builds an in-memory tree structure of the target UXF description and then registers each nodes into the Basic Object Adapter (BOA), which is a server-side component in CORBA. After that, newly created document is accessible remotely.

As described above, our system supports both synchronous and asynchronous manipulation of courseware. The synchronous editing is performed as shown in Figure 13. A client-side document agent locks the remote document using its method `lockNode()` to prevent modifications by other applications. Once the document is locked, its metadata is transferred to the document agent. Then, the document agent accesses and manipulates the remote

document using the obtained metadata. After editing remote documents, the document agent releases the lock with its method `releaseNode()`. The synchronous editing is a simple model and ensures the document's consistency by preventing overwrites. However, it does not scale well in the situation where many learners are accessing a remote document frequently because every access requires a lock even if the document is not changed.

In contrast to the exclusive lock, our system provides an alternative lock, shared lock, which allows a group of participants to work together on a single resource. This lock enables the asynchronous editing. Participants can continue to work even when users are offline, e.g. using mobile computers, and merge changes later. It can also reduce the number of remote method invocations. Figure 14 shows the interactions between client and server in the asynchronous mode. This mode works best in environments in which participants know each other's activities, or just refer the remote document.

4. Current Project Status and Future Work

We are now developing some applications for group-paced training via the IIOP connection. We have developed a command-line document reference/revision tool, and connectivity glues for a commercial CASE tools named Rational Rose and MagicDraw. These glue tools fetch UXF description from the resource server using document agents, and converts from UXF to proprietary formats used in Rose and MagicDraw. We are developing glues for some Java source code editors and source code documentation tools.

As for the metadata handling, we work for supporting a full set of the IMS Meta-Data specification. We plan to develop a LDAP (Lightweight Directory Access Protocol) compatible repository to manage metadata in the centralized manner. It would allow metadata to be exchanged among a variety of distance learning systems.

As for the Persona system, we extend it so that both document and user agents are more intelligent and proactive. We are analyzing those agents' behaviors, and investigating more effective behaviors for learners. Also, we are adding additional HTML customization policies. The HTTP connection is extended using WebDAV [28] to extend its one-way communication for the collaborative training.

As for the IIOP-based system, we are investigating the different locking scopes, e.g. a lock for a collection of documents. We are also analyzing the system behavior in the asynchronous editing mode, and investigating the consistent distributed document management.

We have finished an experiment use of our system, and are now in the process of applying it for some real student courses. We plan to improve our system functionality and usability by analyzing the student attitude to courses and their effectiveness using questionnaires for students and educators.

5. Conclusion

This paper proposes an integrated system architecture for agent-based distance learning, and describes our current applications for self-paced learning and group-paced training. Our system delivers courseware effectively to students, allows educators and learners to work for educational materials collaboratively, and provides the application-interoperability in heterogeneous computing environments. We believe our work provides a blue print for showing how emerging technologies can be applied to practical distance learning applications.

6. Acknowledgements

We sincerely thank Robb Keayes for improving this paper with his careful reading and invaluable comments. We are grateful to Yutaka Abe and Kei Fuji for their significant contributions to the Persona project. We also thank Nozomu Matsui, Kumiko Nakano, Gaku Tashiro, Shogo Tsuji for their supports.

7. References

[1] J. Suzuki and Y. Yamamoto. Document Brokering with Agents: Persona approach. In *Proceedings of the 6th Workshop on Interactive Systems and Software (JSSST WISS '98)*, Miyazaki, Japan, December 1998.

[2] J. Suzuki and Y. Yamamoto. Toward the Interoperable Software Design Models: Quartet of UML, XML, DOM and CORBA. In *Proceedings of the 4th IEEE International Software Engineering Standards Symposium (ISESS'99)*, Curitiba, Brazil, May 1999.

[3] Object Management Group, *Common Object Request Broker Architecture version 2.2*, 1998, at http://www.omg.org/library/c2indx.html.

[4] Object Management Group, Unified Modeling Language Specification version 1.3 beta R1, 1999, at http://uml.systemhouse.mci.com/artifacts.htm.

[5] J. Suzuki and Y. Yamamoto. Making UML Models Exchangeable with XML over the Internet: UXF approach. In *Proceedings of UML'98*, Mulhouse, France, June 1998.

[6] J. Suzuki and Y. Yamamoto. Managing the Software Design Documents with XML. In *Proceedings of ACM SIGDOC'98*, Quebec City, Canada, September 1998.

[7] J. Suzuki and Y. Yamamoto. Metadata Management in Personalizing Web Presentations. Poster position paper of *7th International Conference on User Modeling*, Banff, Canada, June 1999.

[8] J. Suzuki, G. Tashiro, Y. Abe and Y. Yamamoto. Persona: A Framework to provide Adaptive

Presentation for Web Documents. In *Proceedings of the IPSJ Summer Programming Symposium*, Tateshina, Japan, September 1998.

[9] P. Maes. Agents that reduce work and information overhead. In *Communications of the ACM*, July 1994.

[10] S. Green et. al. *Software Agents: A review.* at http://www.cs.tcd.ie/research_groups/aig/iag/toplevel2.html.

[11] H. Lieberman. Autonomous Interface Agents. In *Proceedings of the ACM Conference on Computers and Human Interface (CHI '97)*, March 1997.

[12] B. Laurel. Interface Agents: Metaphors with character. In *Software Agents* (M. Bradshaw, ed.), MIT Press, 1997.

[13] O. Lassila and R. R. Swick (eds.). *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Proposed Recommendation, February 1999, at http://www.w3.org/Press/1999/RDF-REC.

[14] M. Resmer. Internet Architectures for Learning. In *IEEE Computer*, vol. 31, No. 9, September 1998.

[15] IMS Project. *IMS Meta-Data Specification, version 1.02*. at http://www.imsproject.org/.

[16] J. Suzuki and Y. Yamamoto. SoftDock: a Distributed Collaborative Platform for Model-based Software Development. In *the Proceedings of the 2nd International Workshop on Network-Based Information Systems (NBIS'99)*, Florence, Italy, October 1999.

[17] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal. *A System of Patterns: Pattern-oriented software architecture*. WILEY, 1996.

[18] E. Gamma, R. Helm, R. Johnson and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[19] M. Champion et. al. (eds.). Document Object Model Level 1 Specification. W3C Recommendation, 1998.

[20] SAX 1.0: The Simple API for XML. at http://www.megginson.com/SAX/index.html.

[21] D. Chang and D. Harkey. *Client/Server Data Access with Java and XML*. Wiley, 1998.

[22] J. Suzuki and Y. Yamamoto. OpenWebServer: an Adaptive Web Server using Software Patterns. In *IEEE Communications Magazine*, Vol.37, No.4, April 1999.

[23] J. Suzuki and Y. Yamamoto. Building an Adaptive Web Server with a Meta-architecture: AISF approach. In *Proceedings of SPA'98*, March 1998.

[24] J. Suzuki and Y. Yamamoto. Dynamic Adaptation in the Web Server Design Space using OpenWebServer. In *Proceedings of SPA'99*, to be appeared.

[25] Netscape Corporation. Persistent Client State: HTTP Cookies. at http://www.netscape.com/newsref/std/cookie_spec.html

[26] W. Brenner, R. Zarnekow and H. Wittig. *Intelligent Software Agents: Foundations and Applications*. Springer, 1998

[27] ORBucus ORB. http://www.ooc.com/ob/.

[28] J. Whitehead, Jr. and M. Wiggins. WEBDAV: IETF Standard for Collaborative Authoring on the Web. In *IEEE Internet Computing*, Vol2. No. 5, Sept/Oct, 1998.