SOLVING THE PROBLEM OF INTEGRATING A CLIENT-SERVER ARCHITECTURE WITH A MULTI-AGENT SYSTEM IN THE EDUCATIONAL DOMAIN

Adolfo Riera Alvarez, Eduardo Sánchez Vila, Xosé A. Vila Sobrino, Manuel Lama Penín
Dep. Electronics and Computer Science, University of Santiago de Compostela, Santiago de Compostela, Spain

Elke Hinrichs, Rudolf Ruland
Fraunhofer Institute for Applied Information Technology FIT
Birlinghoven Castle, Sankt Augustin, Germany

ABSTRACT
This paper presents the middleware developed under the EUME project to integrate the EUME environment, an Intelligent Learning Management System aimed to support classroom activities, with the BSCW, a 'shared workspace' system that enables collaboration over the Web. This middleware allows the EUME multi-agent system to send information to, and get feedback from, the BSCW client-server system by means of a XML-RPC based communication. The motivation behind this integration stems from the need to improve the learning process initiated in the classroom with an infrastructure supporting follow-on activities based on cooperative work between remote students.

KEYWORDS
Multi Agent System, Client-Server, XML-RPC.

1. INTRODUCTION

Software agents are considered software entities, such as software programs or robots, that present, with different degree, three main attributes: autonomy, cooperation and learning (Nwana 1996). Autonomy refers to the principle that an agent can operate on their own (acting and deciding upon its own representation of the world). Cooperation refers to the ability to interact with other agents via some communication language. Finally, learning is essential to react or interact with the external environment. Teams of intelligent agents build up MultiAgent Systems (MAS). In this type of systems each agent has either incomplete information or not enough capabilities for solving the problem at hand. Other important aspect concerns with the lack of centralized global control; therefore, data is distributed all over the system and computation is asynchronous (Sycara 1998).

On the other hand, client-server architectures (Orfali 1999) are made up of two types of components, clients and servers, communicated through an asymmetrical communication protocol. Clients send requests to servers, and these ones react performing an appropriate action and sending back the reply.

Even though these two architectural styles are used in different types of problems, sometimes both can be required into a single solution. For instance, when dealing with regular tasks, such as database access, and complex tasks, such as reasoning or planning. However, the integration of styles is not trivial, and many problems arise concern with communication as well as information model mismatches. This work presents a middleware aimed to solve these problems and to allow a MAS system to manage a client-server application.
2. MOTIVATION

The general problem posed in the last section was faced in the educational domain. During the last years several efforts to innovate in both distance and in-classroom learning have been made. Some examples are virtual communities, a group of learning participants connected through the Internet (Fion, 2002), and smart classrooms, which include technologies to introduce electronic devices, networking, and multimedia technologies in order to support a myriad of educational scenarios, ranging from collaborative activities (Danesh, 2001) to activity evaluation (Chen, 2000) and recording (Abowd, 1999). Recently, the Department of CSCW of the Fraunhofer Institute of Applied Information Technology (FIT) and the Intelligent System Group (GSI) of the University of Santiago de Compostela have started a joint effort oriented towards the integration of the BSCW, a web-based system supporting virtual communities, and the EUME system, an Intelligent Learning Management System for smart classrooms. The former based on a client-server architecture, the latter based on a MAS.

With the integration of EUME and BSCW, we envision collaborative work in virtual communities as the natural continuation of in-classroom activities, and, on the contrary, these ones as resuming sessions of the work performed in every community.

2.1 EUME system

The EUME system (spanish acronym for Multimedia Ubiquitous Environment for Education) (EUME project, 2004; Riera et al., 2004; Sánchez et al., 2003) is an Intelligent Learning Management System, which covers general services for teaching activity such as learning design, content management and hardware and software resource management. It allows the teachers to design the courses by following the IMS-EML (Koper, 2001) and LOM specifications (Hodgins, 2002). During the design stage of each course, the teacher will specify the course contents, the related learning activities, the required learning materials or resources for each activity, and finally the assignment of activities either to the whole class or to student groups. After that, the teacher carries out the active stage in which the scheduled activities will be performed with the students in different sessions.

The architecture of the system (figure 1) is composed by four different tiers (Riera et al. 2004, Sánchez 2003):

- **Resource Tier**: It contains software components that directly handle learning resources, such as applications and hardware devices available in the classroom.
- **Service Tier**: It is composed by software agents providing high-level learning management services. A quiz generator and an automatic result analyzer could be examples of services in this tier. The services are grouped according to previously defined user profiles.
- **Mediator Tier**: It acts as a security check point between clients and services (but also among the clients), thus preventing clients from requesting unauthorized tasks (such as a student trying to access...
a grading service), and is in charge of keeping updated information about the clients connected to the system.

- **Client Tier:** It contains all the graphical user interfaces (GUIs). The Teacher or Professor is assigned two different types of clients: Professor In Classroom Client (PIC/C), used to interact with the system during the active stage, and the Professor Out Classroom Client (POC/C), used to interact with the system during the design stage.

The system was implemented using the Java Agent Development framework (JADE), a free agent development framework. Compliant with the ‘de facto’ standard in the Software Agent field: the FIPA specifications (FIPA). Following a FIPA recommendation a new educational ontology, the EUME ontology (Amorim et al., 2003), which describes both contents and resources used by the system, was also built as a common language used for the agents during the communication process.

The hardware infrastructure, which will support the system, is constituted by an organizational server, in which the main EUME’s services are running; a database, where the information is stored, a classroom server, which manages the resources on each classroom; general resources (applications, projectors, cameras, printers, etc.); and, finally, the user interface devices. On this regard, PDAs were chosen because of their usability and mobility, but other types of devices (laptops or desktops, for instance) could be used as well.

### 2.2 BSCW system

BSCW — Basic Support for Cooperative Work, (Bentley 1997, BSCW) — is a web-based groupware system around the notion of a *shared workspace*, a joint storage facility that may contain all kinds of objects such as documents, tables, graphics, spreadsheets, links to web pages and more. The members of a workspace, e.g. students of a class and their teacher, cooperatively upload, edit or download documents and other objects to coordinate their work and communicate results.

Besides user and group management, the BSCW system maintains stored documents and applies access control mechanisms to shared workspaces. BSCW supports the concept of roles — workspace managers may define new roles and assign roles to group members. The system keeps the members of a group informed about each other’s relevant activities in the shared workspace by means of event alerting, catch-up functions and daily reports.

BSCW proved a powerful and scalable groupware system in several thousand server installations worldwide; the software runs on all major operating systems such as Windows, Linux, Solaris, etc. On the client side the only prerequisites are a standard Web browser and an email address.

#### 2.2.1 X-BSCW: API to BSCW

Through the X-BSCW API, a BSCW server offers the same end-user actions as via the HTML interface which users normally use in a browser. The API may be used to offer of BSCW services by other end-user programs or in batch scripts, or to customize BSCW and strip down BSCW’s functionality. It is the purpose of X-BSCW to provide access to the core features of BSCW to "anyone" not willing to use the HTML interface of BSCW; no internal knowledge of BSCW is required. It is also well-defined interface to other clients of a BSCW server: various other projects, mobile devices, third-party services etc.

The BSCW Shared Workspace system is an extension of a standard Web server through the server CGI Application Programming Interface: BSCW consists of a set of Python modules (Python). A *BSCW server* manages a number of shared workspaces, i.e. repositories for shared information, accessible to members of a group using a simple user name and password scheme. In general a BSCW server will manage workspaces for different groups, and users may be members of several workspaces (e.g. one workspace corresponding to each project a user is involved in).

BSCW functionality can be accessed either via an ordinary Web browser or via the XML-RPC API using an XML-RPC client (figure 2). The XML-RPC interface basically provides the same functions as the human-operable Web interface. Where the latter returns calls by rendering HTML output, which is eventually
displayed by a Web browser, the first returns with an XML-RPC response. Both the normal HTTP/HTML and the XML-RPC interface to BSCW are accessible via the same port.

3. INTEGRATION TECHNICALS PROBLEMS

In order to integrate these two independent and architecturally different systems, a number of problems have to be solved:

- Two different design paradigms: a traditional client-server architecture in BSCW and a Multi-Agent approach in EUME.
- Two different communication paradigms (figure 3): the EUME agents communicate with each other interchanging messages encoded in the Agent Communication Language (ACL), while the BSCW is a event-driven system that offers an XML-RPC interface for connectivity with other applications.
- Two different information models (figure 4): the EUME system has a educational-oriented information model, but the BSCW, as a more general tool, has a flexible folder-oriented one; therefore, the mapping between both models is far from being trivial.
- Two different programming languages: EUME is implemented using java and BSCW with python.

Figure 3. Communication layers
4. SOLUTION: BSCW-EUME BRIDGE

To achieve our objectives we have designed the EUME-BCSW bridge: a new component that allows the bidirectional information interchange between both systems. Its architecture was inspired by the combination of three classical design patterns (figure 5), and it is composed by four different components (figure 6): the data Model, the Encoder, the Event Dispatcher, and the Translator:

- Both systems were developed using two different programming languages, so we need a communication language that might be understood by them both. The solution was XML-RPC. EUME will use the BSCW’s XML-RPC API to interact with BSCW. Thus, EUME will use the Model to represent the actions that should be carried out, and the Encoder to send those actions as XML-RPC calls. These components follow the Wrapper (Gamma 1995) pattern: they offer a java interface to a BSCW.
- The BSCW’s events will be managed through the Event Dispatcher: it listens for those events and notifies the Translator when one is received. In this way, it implements the Observer pattern.
- And finally, the Translator is an agent that offers the same features than the system (because it could communicate with it) but using the communication protocols of the multi Agent System. It is also in...
charge of the high level task of translating the information model of the systems. So, it follows theFacade pattern providing a unified interface to the whole subsystem.

![Figure 5. EUME-BSCW bridge’s software architecture](image1)

![Figure 6. EUME-BSCW bridge’s design patterns](image2)

The elements involved in the communication process depend on the direction of the information flow. So, we need to analyze at least two different use cases to understand how the components interact each other:

- If an action is carried out in EUME, the Translator will be informed about this action, and it will prepare the action in the BSCW using the classes of the Model. This action will be passed to the Encoder, which code it into a XML-RPC call and send it to the BSCW.
- If an action is performed in the BSCW, the server will throw an event that will be received by the Event Dispatcher, which will code it as a java event and inform the Translator about it. Then, the Translator will communicate with the right EUME’s agents to update the system.

At this point, we provide a more detailed explanation of each one of the elements that compose the EUME-BSCW bridge.

### 4.1 Model

The Model is a set of java classes that model all the different elements used by the BSCW (folder, document, URL, etc.), and all the actions that could be performed using the BSCW’s XML-RPC based API (create a folder, upload a file, define a URL, etc.). This enables a java system to manage the same information structure as the BSCW in a full java format.

### 4.2 Encoder

The Encoder is the element in charge of the XML-RPC communication from a java system to BSCW. Its functionality consists of the following steps:

- First, it receives a BSCW action coded as a Model’s java class.
- Then, this action is translated into a XML-RPC call. It is important to remember that the Model only contains the actions supported by the BSCW’s XML-RPC API. So, this call could be managed by the BSCW server.
- Then, it carries out the XML-RPC call, and processes the response (this processing consists in encoding the response into Model’s java classes).
- Finally, it returns the processed response.

The XML-RPC calls are performed by means of a external library: the Apache XML-RPC java API (Apache XML-RPC). It allows a fast, and easy, development of applications that could manage this remote procedure calling protocol.

The Encoder could be understood as a wrapper: it encapsulates the BSCW system enduing it with a java interface with the same functionality of it’s XML-RPC API.
4.3 Event Dispatcher

The BSCW uses an event-based system to inform about the actions that users carry out on the different workspaces. The system could be setup to allow XML-RPC servers to register as event listeners. If this kind of listeners are present, the BSCW could notify the events thrown by calling a specific XML-RPC method that the listener should implement.

Following the same idea of the Encoder, the Event Dispatcher wraps the BSCW transforming it into a java event producer. For it, the Event Dispatcher must implement two different interfaces: it becomes as a BSCW event listener and a java event producer. To act as a java event producer, it must implement the needed methods to register and unregister java listeners, which will be stored in the Listener Registry and will be notified about the events received from the BSCW.

The BSCW event listener interface is a bit complex, and requires two different elements:

- An Encoder: the dispatcher must register (and deregister) itself as a event listener in the BSCW server. This process is carried out using the XML-RPC API.
- An XML-RPC server: this element implements the XML-RPC method that is called by the BSCW when a event is thrown. This method translates the event into a java event (a Model’s java class too) encapsulating the same information, which is sent to the java listeners registered in the Event Dispatcher. This server was implemented with an extended version of the Apache XML-RPC API. The Apache API is object-oriented, so they only could serve methods that belong to an object. However, the XML-RPC method that the BSCW calls when a event is thrown is a “global method” (a method that belongs to any object). To solve this problem, the XML-RPC server was modified to use an object as default: when a call is received for a “global” method, the same method of the “default object” is called.

4.4 Translator

The three EUME-BSCW bridge’s components introduced so far could be called “general purpose” components because they could be used by any java application to communicate with a BSCW server. Those components solve the technical problems related to the communication process, but what about logical ones? Who knows what to do when a new content is included in a Learning Design or what to do when a BWSC event is received? The answer is: the Translator.

This component is a java Agent, included in the Service Tier, which could perform actions over a BSCW server through a Encoder and which knows all the actions performed by the users because it’s registered as a event listener in a Event Dispatcher connected to the server. Even this component bears a great responsibility on the EUME-BSCW integration (it must adapt the EUME information into a BSCW folder hierarchy, it must keep updated the information on both systems,…), it’s implemented following a quite simple philosophy, too close to “if…then…” rules:

Inside the Translator there are a set of subcomponents called Triggers. They monitor the system and check whether a list of predefined conditions is verified. Those conditions do not follow a common pattern, and currently two kind of them are implemented: one checks whether a specific message arrives from other EUME agent, and the other checks whether a specific event has been thrown. With this two kinds of triggers, the translator could detect changes both in EUME (through messages from the other agents) and in BSCW (thought events thrown by the Event Dispatcher). Associated to each Trigger there is an Action that defines a sequence of “atomic” actions that modify the EUME or BSCW in some way. So, this “atomic” action could consist in send a message to other EUME agents or carry out a call to the BSCW XML-RPC API. As a synopsis we could say that the Actions encapsulates what do and the Triggers when do it.

5. CONCLUSION

This paper describes a solution that allows the integration of a Multi Agent System, such as the EUME system, with a client-server application, the BSCW system. It opens the possibility of reusing applications based on low-level communication in systems with high-level communication requirements. In the
educational field, the integration of EUME and BSCW aims to provide a system that consider both in-classroom and out-of-classroom activities as natural and inter-related elements of the overall learning design rather than separate learning processes.

As future work we have planned to improve the bridge in two different ways: improve the system extensibility creating a easy procedure to add new actions over the two systems, and use an ontology to perform a consistent mapping between the two different information models.

ACKNOWLEDGEMENT

Authors would like to thank the Xunta de Galicia for their financial support in carrying out this work under the project PGIDT02TIC20601PR.

REFERENCES


EUME project, http://www.eume.net/


FIPA, http://www.fipa.org/

Gamma, E., Helm, R., Jhonson, R. And Vlissides, John, 1995. *Design Patterns, elements of reusable object-oriented software*. Addison-Wesley, Reading (Massachusetts, USA).

GSI, http://www-gsi.dec.usc.es/


JADE, http://jade.tilab.com/


