An Agent- and Service-oriented e-Learning Platform

Ivan Madjarov[1], Omar Boucelma[2], Abdelkader Betari[3]

[1] IUT d'Aix-en-Provence, Département GTR,
Université de la Méditerranée, Marseille, France
ivmad@iut-gtr.univ-mrs.fr
[2] LSIS - CNRS et Université de Provence, Marseille, France
omar@cmi.univ-mrs.fr
[3] Laboratoire d'Informatique Fondamentale de Marseille - CNRS
Université de la Méditerranée, Marseille, France
betari@lif.univ-mrs.fr

Abstract. This paper presents an e-Learning Web-reachable hypermedia system as the foundation of a course content development toolset. Course content, developed in XML, is stored in native XML databases and propagated via Web services. A helper agent delivers the learning objects that compose a course based on a pedagogical strategy pre-defined by the course author. The agent dynamically establishes the learning objects delivery order. The sequencing of Web pages in the proposed system relies on a Petri Net analysis of incoming events such as student responses to exercises.

1 Introduction

This paper introduces an effective agent-oriented approach to drive the implementation of a Web Services-based e-Learning system.

Metadata, the primary building block of the emerging semantic web, will enable computers to understand the nature of information available over the Web hereby driving the coordination of complex Web service assemblies. The e-Learning domain was one of the first to benefit from the definition, among other standards, of LOM (*Learning Object Metadata*) [4]. From another perspective, the distributed nature of the Web suggests that agent technologies will play a key role in leveraging the use of metadata.

For the purpose of this paper, a "*service*" is a component, which handles transactional requests and directs their execution. An "*agent*" is a component, which acts like a mediator between a user and a service. Various interaction configurations are possible including agent-agent, service-service, agent-service, etc. Typical agent architectures have many of the features found in Web Service platforms.

Web Services are self-contained, modular applications that provide a set of functionalities to anyone that requests them. The main characteristic of Web Services is that they interact with the applications that invoke them, using Web standards such as WSDL (*Web Service Definition Language*), SOAP (*Simple Object Access Protocol*) and UDDI (*Universal Description, Discovery and Integration*). Leveraging

W. Liu et al. (Eds.): ICWL 2004, LNCS 3143, pp. 27-34, 2004

Web standards in an e-Learning environment facilitates the dynamic integration of applications distributed over the Internet regardless of their underlying platforms. Web Services follow the SOAP Protocol to handle communication. The strong reliance of Web Services on XML standards guarantees interoperability so that data manipulated in the proposed learning model becomes readable from any computer. Web Services rely on HTTP (*HyperText Transfer Protocol*) for message transfer and thus have the advantage of being able to flow messages securely through most system firewalls, proxy servers, etc.. Finally, Web Services follow the WSDL protocol to provide the descriptive metadata required to use the services, and UDDI to publicize services on UDDI servers. These combined functionalities facilitate the dynamic integration of applications distributed over the Internet regardless of their underlying platforms.

The organization, classification, encoding and distribution of educational content on the Internet is subject to international standards set forth by the IMS Global Learning Consortium, the Advanced Distributed Initiative (ADL) and the IEEE [7]. A synergy between all these standards is the proposal to organize educational content in *learning objects*, conceived as relatively small chunks of educational material. These objects can be shared among users, reused in different contexts, recombined in different ways to build larger blocks of educational content and create personalized learning paths.

Nevertheless, Web-based educational applications today are still mostly static and based on a generic approach to tutoring that does not take into account individual student needs and scores. An important limitation to consider is that most e-Learning systems available today do not provide personalized and intelligent assistance. A helper agent approach sets forth a major contribution as it comes to taking into account differences among students.

The next section of this paper presents the architecture and implementation of our XESOP system. In the last section, we discuss issues related to the dynamic choosing and sequencing of learning objects that compose a course and how they are delivered to students based on their responses to tests.

2 The XESOP system and the course design

Our XESOP system [5, 6] is based on XML technologies and may be used for course development in an e-Learning environment. The structure of a course is subject to a XML Schema (i.e., a grammar) developed according to the most recent standards adopted by the e-Learning community [7]. The system consists of a semantic editor for XML documents with plug-ins that support the description of mathematical equations (MathML) and vector graphics (SVG). Data integrity is preserved via validation rules and by keeping the data structures independent from their presentation. Learning objects created using this system are stored in native type model XML databases [8, 9]. Learning objects include pages of type lesson, examples, exercises or test forms. Presentations are created via targeted selection of relevant document parts. This selection leads to complementary tagging that facilitates automatic content extraction via parsers.

The learning objects representing exercises have a major importance in e-Learning platforms as they make it possible to keep students challenged as they take a course. The exercise concept is therefore an essential ingredient in the assimilation of knowledge. For this reason, we have designed a service and a tool to facilitate the development and execution of remote exercises. The RDS (*Remote Development Service*) can be programmed by course authors to make exercises accessible and actionable at the time and place of delivery. The use of exercises in this environment relies on the creation of a Web Service that provides the necessary functions for the compilation, execution and/or interpretation of these exercises. To facilitate seamless access to the RDS functionality, related development tools are available [6] on a Web-Application Server, where the Web Service resides. This technology is open, leverages the standard Internet protocols (SOAP, TCP/IP), and facilitates access to a native XML database [9] that is responsible for course material integrity.

The system currently runs on the Linux and Windows operating systems. The software, implemented in Java, is an extension of functionalities provided by free and *open source* products. The learning objects structure and design conforms to the LOM specification [4]. Compatibility with IMS [7] standards has been thought through as well to facilitate integration with third party *Learning Management Systems* (LMSs). Our main motivation in implementing the system was to come up with a cost effective portable alternative to expensive commercial LMSs.

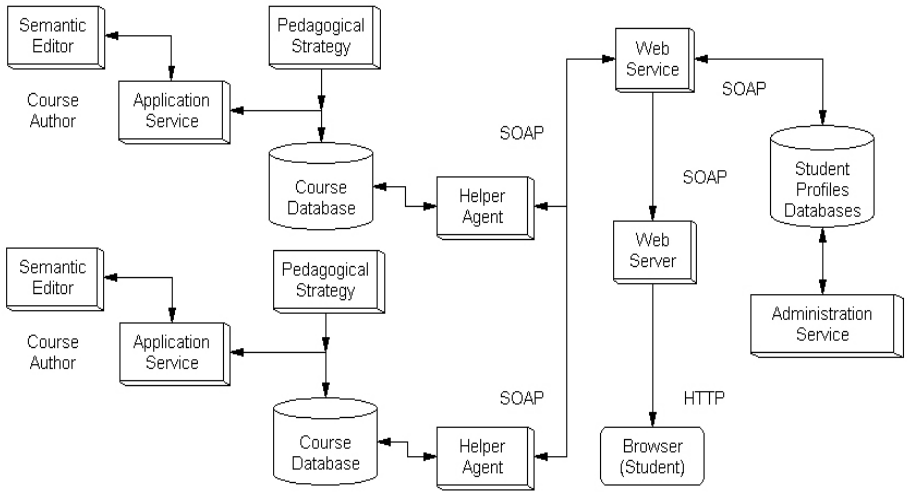

Figure 1: XESOP system architecture

The realization of a pedagogical strategy relies on facilities for delivering and presenting course learning objects from both *traditional* and *adaptive* ordering viewpoints. Traditional sequencing is most straightforward as the system delivers learning objects according to the order defined by the course author. Adaptive sequencing manages learning objects distribution according to students' answers to

exercises. In this case, learning objects' sequencing results from the nature of student answers rather than being based on a pre-established order selected by a course author. In the adaptive approach, sequencing of course pages is never static and is always driven by student capabilities. A *helper agent* is responsible for the guided generation and updating of the content sequence.

3 Helper agent for Web page distribution

We are interested in devising a helper agent responsible for the distribution of course-related Web pages to students according to their tests results. Management of page sequencing is based on an analysis of incoming events via a Petri Net.

Carl Adam Petri introduced Petri Nets in 1962. Petri Nets provide a well-known process modeling technique with formal semantics. Petri Nets have been used to model and analyze several types of systems (i.e.: distributed, asynchronous, synchronous, deterministic or non-deterministic) and processes (i.e. protocols, manufacturing, business) [3].

A Petri Net is a directed, connected and bipartite graph in which each node is either a *place* or a *transition*. The *edges* of this graph connect the *transitions* to the *places* or the *places* to the *transitions*. The places contain *tokens* or marks, which go from place to place by crossing the *transitions* according to *crossing rules*.

Formally [2], a Petri net is defined as a flow relation $< P; T; W >$ where P is a finite set of places, T is a finite set of transitions and W is a set of directed *edges* (flow relation). The flow relation may also be quantified in terms of an evaluation function $W : (P \times T) \cup (T \times P) \rightarrow \{0,1\}$ [3].

When the number of entities modeled in the system is large, the size of the Petri Net quickly becomes enormous and if the entities present similar behaviors, the use of a colored net makes it possible to condense the model. The colored Petri Nets are nets in which the tokens carry colors. This kind of information makes possible to distinguish tokens from each other [1].

A synchronous and colored Petri Net is represented as:

$$R = (P, T, Col, \pi, \lambda) \tag{1}$$

where:

- P is a finite and non empty set of *places*;
- T is a finite and non empty set of *transitions*;
- *Col* is a non empty set of *color* properties;
- $\pi \subseteq PxT$,
- $\lambda \subseteq TxP$,
- And we have the property: $P \cap T = \varnothing$.

To address the problem and the solution set forth in this article, we propose to study the management of a Web-based course. As an example, let us assume that a course is composed of three pages and nine exercises. One main exercise is associated to each

course page. Upon initial connection, the student receives the content of the first page (Page 1). After receiving this page, the student goes on to the first exercise (Ex. 1), which corresponds to an event of type "*suite*" which means "next". The student's answer to the exercise represents an event of type "*validate*". The answer is analyzed and results in a constant from the set {*True, False*}. If the answer is correct, the next course page is sent to the student (Page 2). If the answer is not correct another exercise of the same type as the previous one is sent to the student (Ex. 1-2). If the student's response to the second exercise is correct, the second page of the course is sent to the student, if not the student receives a third exercise of the same type as the two previous ones (Ex. 1-3). The same process applies to all course pages and their corresponding exercises.

The Petri Net associated with this example is represented on Figure 2. Crossing orders are indicated on the corresponding transitions. If the student fails all the exercises corresponding to a course page, a start page is sent back. Similarly, once a student successfully completes the third main exercise, a final page is sent back. For simplification purpose, these ending conditions are not represented in the figure.

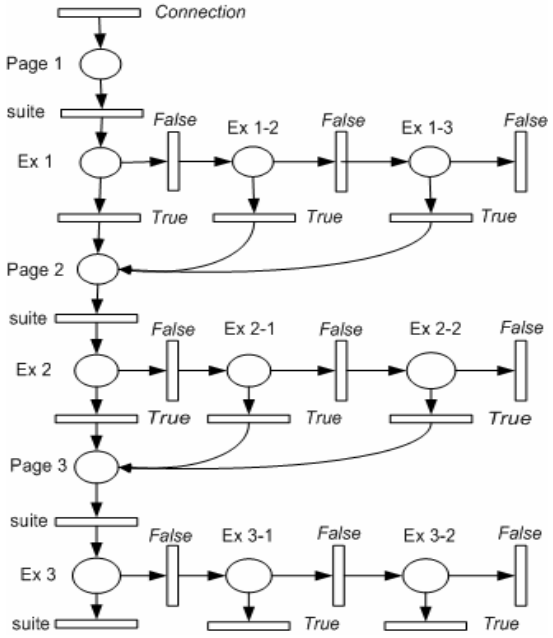

Figure 2: The course management presented by a Petri Net

According to the Petri Net formalism, a course may be represented as:

$$C = (NC, DC, URI, PN) \quad (2)$$

where:

– *NC* is the name of the course,

- *DC* is the description of the course,
- *URI* is the Internet address of the course,
- *PN* is the Petri Net, which models the course management.

The relationship between the basic Petri Net formalism (1) and the course elements is as follows:

- The *P* set represent the set of course pages,
- The *T* set represent the set of events corresponding to the answers,
- The color represents the identity of each student.

In other words, we build the Petri Net by associating a place for each page and a transition to any page changes. To manage the page changes, we analyze the data coming from the student. To carry out the analysis of the answer, we introduce two complementary concepts:

- A *RepEx(a)* function used to defines the type of the data coming from the student machine ("*validation*" or "*suite*"),
- A $\mu(a)$ function used to analyze a response of type "*validation*". This function translates the student answer to a constant {*True, False, Suite*}.

The analysis function compares the returned answer with a standard answer. By evaluating their differences, a grade ("note") is generated as an integer value between 0 and 10 ($0 <= note <= 10$):

- $\mu(x) = True \{ (\text{RepEx}(x) = True) \cap (note(x) \geq 5) \}$,
- $\mu(x) = False \{ (\text{RepEx}(x) = True) \cap (note(x) < 5) \}$,
- $\mu(x) = suite \{ \text{RepEx}(x) = False \}$.

Any transition in the Petri Nets under consideration is ordered according to one single place. This justifies its encoding using a Hash table. The advantage of a Hash table is that the time of insertion and search is *O(1)*. The construction of the Hash table is subject to a collisions control.

The course configuration is described by the author in an XML document using the XML semantic editor [6] provided by the e-Learning platform [5]. The content of this document conforms to the following schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xsd:element name="valid">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="PlaceE" type="xsd:string"/>
      <xsd:element name="Transition" type="xsd:string"/>
      <xsd:element name="PlaceS" type="xsd:string"/>
      <xsd:element name="URL" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

where, *PlaceE* and *PlaceS* ∈ course pages, *Transition* ∈ {*True*, *False*, *Suite*}. The key in the Hash table is obtained by concatenating "*PlaceE* + *Transition*"; the corresponding value is obtained by concatenating "*Places* + *&* + *URL*".

The place "*PlaceE*" is the entry point of the transition "*Transition*". For the validation of "*Transition* ", a mark must be present in the place "*PlaceE* ", and the value of the "*Transition*" must correspond to the value of $\mu(x)$. In other words, when a student is in the page corresponding to "*PlaceE*" and "*Transition*" is validated by the event corresponding to the answer, the following page becomes "*PlaceS*".

For student identification and grade attribution, we associate a color property to the mark. This makes it possible to follow the behavior of several students simultaneously, and to memorize the state of users who have visited the course so far. The pair (*id*, *note*) represents this mark, where:

- *id* is the name of the user,
- *note* is the grade of the student at a given time.

The *color* and *id* fields allow the management of several student sessions at the same time. A Hash table "*Gestion*" (which means "Management") carries out the next stages of several sessions, and is organized as follows:

- The keys of the table correspond to the pages of the course,
- The values represent the marks.

The values of the table "*Gestion*" are modified in the following way: when the user goes from one page to another, which corresponds to going from "*PlaceE*" to "*PlaceS*" by crossing a "*Transition*", its mark is removed from the list "*PlaceE*" and it is added to the list "*PlaceS*". The "*note*" (i.e., grade) of the student is modified as necessary during crossing using the answer analysis function.

At user connection time, two alternatives are possible: upon the first user visit, the user identity must be added to the table "*Gestion*", if the user already visited this course, its identity is already present in the table. If the student visits the course for the first time a registration procedure is activated (3) and a mark is allotted to him with its identity.

$$(\forall y)\left\{\begin{array}{l}[Conx(y) \supset [AddG(z) \cap (z.id = y) \cap (z.note = 0)]] \\ \Leftrightarrow \\ (\forall x)[\neg InG(x) \cup (x.id \neq y)]\end{array}\right\} \quad (3)$$

where:

- *y* is a variable that represents the identity of the student who just connected,
- *z* is a variable that represents the mark (*id*, *note*) allotted to the student,
- *x* is a variable that represents a mark,
- the function *AddG* inserts the mark passed as argument in the table "*Gestion*"
- the function *InG* returns "*True*" if the corresponding mark is already present in the table "*Gestion*", else it returns "*False*".

The session continuation condition represents the case where the student is already registered in the table "*Gestion*" (4):

$$(\forall x)\left\{\begin{array}{l}[Conx(y) \supset [Cont(z) \cap (z.id = y)]] \\ \Leftrightarrow \\ (\exists x) / [InG(x) \cap (x.id = y)]\end{array}\right\} \quad (4)$$

The function *Cont* indicates the place of continuation.

4 Conclusion

The suggested model, with its helper agent, gives the possibility of managing a course in a centralized fashion. It also facilitates the study of various users' behaviors and the sending of different pages according to user tests' results. This model presents a general structure, which can be adapted to other applications or can be generalized. We can easily introduce the concept of time and that of transforming the Petri Net into a time lag net. This would be interesting in cases where time becomes a factor and plays the role of an event in the management of a course. For example, part of the grade ("*note*") of a student could be allotted based on the time elapsed between the sending of a page to a student and the receipt of the student answer.

References

1. Kurt J. Kurt. Coloured Petri nets: Basic concepts, analysis methods and practical use, volume 1. Springer, (1997).
2. Naquet V., Geniet C. Réseaux de Petri et systèmes parallèles. Armand Colin, (1992).
3. Hamadi R., Benatallah B. A Petri Net-based Model for Web Service Composition. Fourteenth Australian Database Conference, Adelaide, Australia (ADC2003).
4. IEEE. Learning Object Metadata. http://ltsc.ieee.org/wg12. (2003).
5. Madjarov I., Shishedjiev B., Betari A. Remote Development Environment in an e-Learning System, (accepted for publishing in SAER 2004).
6. Madjarov I., Betari A. Un éditeur XML sémantique pour objets pédagogiques stockés dans une base de données XML native, (accepted for publishing in NOTERE 2004).
7. Learning Technology Standards Committee of the IEEE Computer Society, IEEE P1484.1/D8, 2001- 04-06 Draft Standard for Learning Technology — Learning Technology Systems Architecture (LTSA), 2001.
8. Bourret R. XML and Databases. (2003) http://www.rpbourret.com.
9. XML:DB Initiative. Enterprise Technologies for XML Databases. http://www.xmldb.org.