

A Multi-Agent and Service-Oriented Architecture for Developing Integrated and Intelligent Web-based Education Systems

Fuhua Lin, Peter Holt, Steve Leung, Mike Hogeboom, Yang Cao

Center for Computing and Information Systems,
1 University Drive, Athabasca, Alberta, Canada, T9S 3A3
{oscarl, holt, stevel, yangc}@athabascau.ca

Abstract. This paper presents an architecture for developing integrated and intelligent Web-based educational systems (WBES) using multi-agent system technology and Web services technology. Intelligent agents are designed for support Web-based education, considering target population characteristics and topic areas, estimating relevant existing skills and knowledge of the learners, formulating objectives and learning outcomes, selecting appropriate learning objects, assembling courses and curricula, and assessment of learner performance. Web Services are designed for the modularization of WBES, and are excellent complimentary partners with intelligent agents, since they are characterized by their standardized communication protocol, interoperability, easy integration and development. To demonstrate the feasibility of the proposed approach, some agents and Web services for Web-based distance education are implemented.

1 Introduction

Web-based learning systems can potentially deliver personalized course material and services, and are, therefore, able to accommodate a larger variety of learners than what can currently be accommodated. To tap this potential, we propose a framework for an innovative WBES.

An innovative Web-based educational system should consider individual learners differences and the profiles of individual learners. It should be an integrated system, due to the dynamic and distributed nature of both resources and applications [1]. An innovative Web-based educational system should not merely respond to requests for information, but should intelligently adapt and actively seek ways to automate tasks to reduce the ever-increasing information workload.

We are working on designing, implementing, and evaluating, with real users, an agent-supported online educational environment able to support learners during the whole cycle of learning, and able to support educators in curriculum planning, course designing and delivering, tutoring, and learner performance evaluation.

The agent-based approach is suitable for supporting Web-based education since relationships among learners, courses, and instructors last for a considerable period of time [2]. Due to the inherent distributed nature of Web-based learning, a Web-based educational environment can be enhanced by a set of software agents [3, 4]. Much of the experimental research has shown that intelligent software agents have great potential for reducing information workload and for automatically performing many knowledge/labour-intensive tasks for both learners and educators [5]. However, people have faced many challenges in developing this technology for commercial applications, mainly because of the lack of an accepted industry-standard method for the development and implementation of agent-based systems [6] and environments where agents can live and run.

Since Web Services technology is characterised by standardized communication protocol, interoperability, easy integration and development, it provides an excellent architecture for developing service-based learning technology systems. For example, the learning services architecture and learning services stack have been proposed and developed by the Learning Systems Architecture Lab at Carnegie Mellon University [7]. However, when compared to agents, Web Services has some

limitations [8]. For instance, Web services are passive until invoked while agents are inherently communicative. Overcoming these limitations appears to require the integration of agents and Web services. In this paper, we propose an architecture in which Web services are used for modularization of WBES and are excellent complimentary partners with software agents in integrated and intelligent WBES.

2 The Proposed Architecture

In the proposed architecture, an integrated and intelligent WBES consists of users' personal agents, task agents running on distributed agent platforms, and a set of discoverable Web services. The Agent platforms are basic facilities for creating, deleting, and locating agents. They also involve inter-agent communication.

2.1 Personal Agents

A personal agent (PA) or user interface agent (UIA) is a GUI-driven interface between a user and an agent-based learning environment. Through a PA, a user can delegate rights to his/her agent, manage task agents within the environment and configure the options provided by the task agents. Users can meet their PA's either by running an application or opening a secure Web page in a desktop/laptop/pocket PC. There are two main kinds of PA's: instructor PA's and learner PA's.

An instructor PA is an assistant to the instructor, helping the instructor generate, deliver, and maintain online courses. These kind of agents interact with and mediate curriculum planning agents, course delivery agents, course update agents, learning object recommendation agents, and notification agents to fulfil the tasks delegated by the instructor or respond to requests from learners or learner PA's.

A learner PA is a simulated instructor that can provide adaptive course material and appropriate instruction according to the learning process of the individual learner. These kind of agents can be viewed as an authoritative representative of the course author, the instructor, or the tutor. Learner personal agents manage and configure program-advising agents, tutoring agents, performance-monitoring agents, and collaboration agents located in the agent platforms of the environment.

2.2 Task Agents

A task agent plays either the role of a client of a Web service or the role of a supporter of a Web service. As a client of Web services, an agent can perform searches of different entries stored in a UDDI, and can contain, reason about the semantics of Web services, and mediate and compose Web services. It then can make message- and RPC-style calls to a Web service. As a supporter of a Web service, a task agent facilitates and enables the service. The Web service benefits from the ability of the agents to perform the task autonomously and intelligently, dynamic creation of agents, and semantic level communication. Most Web services in Web-based learning environments can profit from the flexibility and robustness, autonomy, and intelligence of agents. For example, we need 'spiders' [9] which are Web agents to support course information Web services by monitoring and maintaining Web-course materials [10]. Another example is agents for learning object repositories. Vast educational resources available today and tomorrow simply could not function without being able to delegate to agents the multitude of tasks that would otherwise be left to armies of people to handle [11].

A task agent is required to perform certain specific tasks, such as providing services, knowledge, and information resources, and also providing intermediary functions such as coordinating and communicating with other task agents. Therefore, a task agent's memory contains specific task-related information in greater depth than the personal agent's memory can possess. A task agent usually has a monitoring and learning function, which allows it to update its own information through new updates when necessary. Because a task agent is deemed a "common resource" shared by many users, its processing capability comprises a spooling function, in which requests are queued in accordance to their priority. In performing multiple tasks, the resource allocation function determines how many resources should be provided to each uncompleted task.

2.3 Web Services

Web Services for Web-based education can include knowledge management and information resource management located in different places. Knowledge management Web services manage, locate, analyse, and retrieve knowledge for Web-based learning – for example, domain knowledge and curriculum planning knowledge. Information resource management includes ‘learner information management’, ‘staff information management’, ‘course information management’, and ‘educational resource information management’.

These knowledge and resource management Web services are also responsible for retrieving the knowledge and resource needed. For instance, a domain ontology Web service is designed for providing services about a taxonomy database. It is used for a target language for (1) terms in the prerequisite and post-conditions of learning objects, and (2) the terms in the learner profiles.

2.4 Agent Management and Deployment Service

An agent management and deployment service is implemented through the Web technology. A registered user can login to download his/her favourite personal agent(s). The agent management service assigns unique agent identification to agents and records agent information such as agent types.

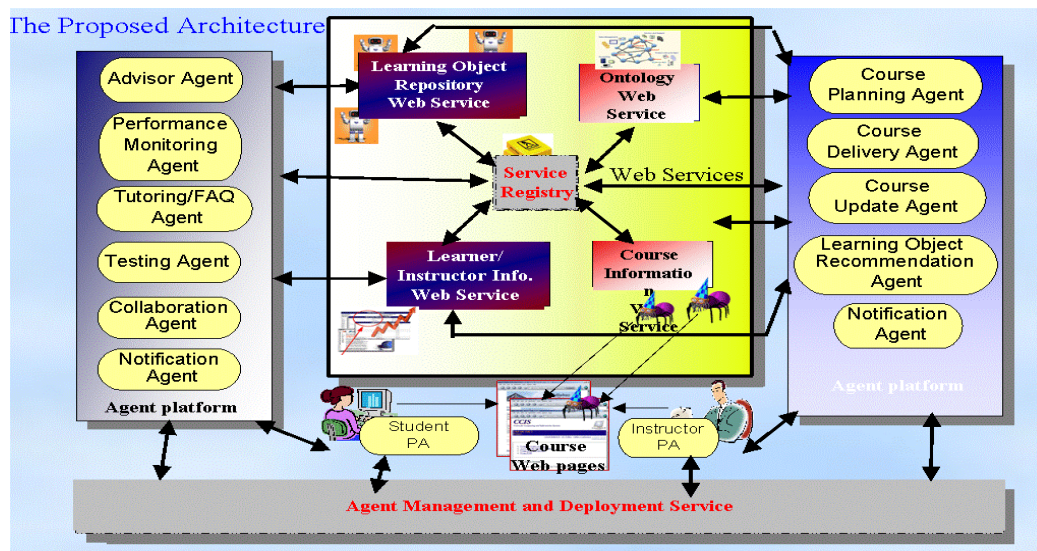


Fig. 1. The proposed architecture.

3 Implementation

3.1 The Agent Platform

We use Apache Axis as the SOAP engine and create a dispatcher that accepts SOAP remote procedure calls as input. We wrote the WSDL describing the method calls that our Web Services will accept by following a combination of the WSDL and schema specifications. We next published information about the Service Provider to a Service Registry, agent UDDI registry. There we entered in data about the Web site, such as the URL of the WSDL for the Web service.

A task agent on the agent platform performs a ‘find’ operation on a service registry. The agent finds the entry for our service and uses the listed URL to download a copy of the WSDL. Using the WSDL, the agent generates a program to serve as the Service Requester to access the service. When this is complete, the agent tests it by requesting that it perform a “bind” operation on the Web service. After the “bind” operation is successful, the agent passes the request and waits for a response.

3.2 Notification Agent

The Notification Agent makes use of JavaMail class to perform the actual sending. The agent gets the sender or recipient information from the Web services through sending XML request messages.

3.3 Learning Object Repository Web Services

The learning object repository Web services are provided by the internal eduSource infrastructure. The initial set of Web services available in the eduSource network are:

- Storing learning objects,
- Storing learning object metadata,
- Tagging tools for creating metadata records,
- Searching learning objects,
- Aggregating learning objects into lessons and
- Handling copyrighted materials.

The backbone of the interoperability is the eduSource Communication Language (ECL) [12] that implemented the core functions defined in the IMS digital repository interoperability reference model.

3.4 Ontology Web Services

We developed Ontology Web services that support Web-based learning in a language and platform-independent manner by using Protégé 2000 (<http://protege.stanford.edu>), a domain-modelling tool from Stanford University [13]. Protégé generates a default form when the domain is created, which can then be further customized to suit the project visual preferences and requirements. Once the domain model and data entry forms have been created, the instance tab, which is a knowledge acquisition tool, can be used to acquire instances of the classes defined in the ontology. Once the model has been populated with information, the Protégé library can be accessed using a Java API to retrieve that information for use in the Java Web Services. The domain models are being populated with IEEE/ACM Software Engineering Body of Knowledge 1.0 (SEBOK) (<http://www.swebok.org>) and ACM Computing Classification Systems (<http://www.acm.org/class/>). In the future, other publications can be added. The service can be invoked and a service side business command can be executed to retrieve the matching content based on the input ID [14].

For example, the “GetTopicsServices” is used to list all the topics (Subject areas) in the Protégé 2000 knowledge base. The service will be invoked and a server side business command can be executed to retrieve the topics. Just like any other WSDL document, the *GetTopics* WSDL is simply a set of definitions. Services are defined using six major elements:

- *Types* - Provides data type definitions used to describe the messages exchanged.
- *Message* - Represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system
- *PortType* - A set of abstract operations. Each operation refers to an input message and output messages
- *Binding* - Specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType
- *Port* - Specifies an address for a binding, thus defining a single communication endpoint
- *Service* - Used to aggregate a set of related ports

The following describes the WSDL file for the *GetTopics* Web Service. This Web service is used to return all the topics (Subject Areas) that are contained within the eLearning knowledgebase.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="GetTopicsService"
targetNamespace="urn:Foo">
```

The following *types* node provides the data type definitions used to describe the messages exchanged between the Web Service and the client. The *GetTopics* Service contains two complexType nodes. The first *complexType* describes the array of topics (*ArrayOfTopic*), while the second describes the actual

topic. The Topic *complexType* maps to the Topic Java object, while the *ArrayOfTopic* maps to both the root array of topics returned from the Java and the subtopics contained within each Java object. Both complexTypes in the following XML reference each other to create the same structure as the Java class as depicted in Figure 1 above.

```

<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:soap11-
enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/" targetNamespace="urn:Foo">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  <complexType name="ArrayOfTopic">
    <complexContent>
      <restriction base="soap11-enc:Array">
        <attribute ref="soap11-enc:arrayType"
wSDL:arrayType="tns:Topic[]" /></restriction></complexContent></complexType>
    <complexType name="Topic">
      <sequence>
        <element name="subtopics" type="tns:ArrayOfTopic" />
        <element name="title" type="string" /></sequence>
      </complexType>
    </schema></types>

```

The message nodes of the *GetTopics* WSDL file provide an abstract definition of the data transmitted. There are two message nodes defining the request and response. The part node of the response message describes the logical abstract content of the response message, being *ArrayOfTopic* in this case.

```

<message name="GetTopicsIF_getTopics" />
<message name="GetTopicsIF_getTopicsResponse">
  <part name="result" type="tns:ArrayOfTopic" />
</message>

```

PortType node describes the set of abstract operations and the abstract messages in the operation.

```

<portType name="GetTopicsIF">
  <operation name="getTopics" parameterOrder="">
    <input message="tns:GetTopicsIF_getTopics" />
    <output message="tns:GetTopicsIF_getTopicsResponse" />
  </operation>
</portType>

```

The binding node defines the message format and protocol of the communication.

```

<binding name="GetTopicsIFBinding" type="tns:GetTopicsIF">
  <operation name="getTopics">
    <input>
      <soap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
use="encoded" namespace="urn:Foo" />
    </input>
    <output>
      <soap:body encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
use="encoded" namespace="urn:Foo" />
    </output>
    <soap:operation soapAction="" /></operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
  </binding>

```

The service node simply groups a set of related ports together.

```

<service name="GetTopicsService">
  <port name="GetTopicsIFPort" binding="tns:GetTopicsIFBinding">
    <soap:address xmlns:wSDL=http://schemas.xmlsoap.org/wSDL/
location="http://localhost:8080/elearning-jaxrpc/GetTopics" /></port></service>
</definitions>

```

3.5 ‘Spider’-like Course Information Monitoring Agent

Currently the Monitoring Agent is designed for monitoring online course material to (1) determine whether or not the links in those pages are broken and (2) determine whether or not the content in those pages have been significantly changed. The meaning of “significantly changed” is based on a couple of pre-defined criteria. For example, the number of hyperlinks or photos increased or decreased, or the content lengths of the Web page by examining its MIME header.

If the monitoring agent discovers such changes, it can trigger a Notification Agent to send out a message to those students who are interested to receive the message. Another way is to store the event of changes in the database. The instructor’s Notification Agent invokes the course information Web Services to notify the instructor of the Updated course information (e.g. newly-broken links in the course materials) or to recommend an alternative link from a learning object repository [10].

Most of the work is done by agentized and multi-threaded class Spider [9]. ‘Spiders’ are programs that can visit Web sites and follow hyperlinks. We have successfully implemented the agent system for Web-based course link maintenance using the architecture above. The agents and the agent platforms were written in Java and deployed at different locations. More than ten courses of CCIS of Athabasca University of Canada have been used for the testing. The agent then checks for broken links (by the spider) and writes the results into MySQL databases. The notification agent can send emails to the course instructors if some broken links have been found. The instructors can configure their agents via running their instructor Personal Agents.

3.6 MARC-IEEE LOM/CanCore Converter

Athabasca University has developed an IEEE LOM/CanCore compliant metadata repository application called ADLiB (<http://ADLiB.athabascau.ca/>), which is a Web application for creating and storing standards compliant metadata records, and for storing their corresponding learning objects in the repository. As part of eduSource Canada project, a MARC-IEEE LOM/CanCore converter (<http://emd.athabascau.ca/courses/crosstalk/converter.html>) has been developed by some of the authors. The tool enables XML-based digital repositories to interact with and harvest metadata from MARC (Machine Readable Cataloguing) records that are used in libraries. MARC records contain hundreds of fields and were developed to describe print materials [15].

3.7 Learning Object Recommendation Agent

Based on Learning object repository Web services, a Learning Object Recommendation Agent has been implemented, which is designed for supporting Learning Objects Repository users in identifying and accessing learning objects according to personalized specifications (preferences) that have been dynamically interpreted. We expect that the agent can work on behalf of the user, monitoring the new arrival of learning objects in the learning object repository and then notifying the user when relevant learning objects are deposited in the repository (<http://ADLiBx.athabascau.ca/lora/jsp/lora/index.jsp>).

3.8 Course Planning Advisor Agent

We have developed a prototype of Advisor Agent to help students and/or student advisors to plan their study. The goal of the agent is to provide a list of reasonable alternate program plans for a student. Although the final decision will be up to the student, his academic advisor and the program coordinator, the Advisor Agent will respond to the latest changes in the environment and promptly advise the involved parties. To facilitate the planning process, the agent will take into consideration course structure information, individualized students’ background, program constraints, and a list of extensible searching heuristics. We create an example to demonstrate the Advisor Agent. In this example we have a program that consists of 5 courses for Program: B.Sc. in Software Development (SD) (see Fig. 2).

We have two students planning to finish the program. One of the students does not have any computer background, while the other has finished a course in another institution that equivalent to SD303. Responding to this request the Advisor agent will advise possible plans (we will call them *program plans* hereafter) to the students and their academic advisor.

Course Structure information and Course Dependency: A program plan is a step-by-step list of courses in a certain sequence that should be taken by a student. The sequence of the courses in a program is built entirely on the idea of “course dependency”. For example, if a course, C1, is said to be dependent on another course, C2, a student will be required to complete C2 before starting C1. However, to establish such a relationship we need a common mapping of all relevant courses to a common course content repository. Though not without controversy, there exist some common course content repositories. For example in the domain of Software Engineering, SWEBOK is a well-known initiative to establish such a standard. The example used below is taken from SWEBOK.

Table 1. Knowledge Areas in SWEBOK
Knowledge Areas 1st level

KA's ID	KA1	KA2	KA3	KA4	KA5	KA6	KA7	KA8	KA9	KA10
KA's Name	Software Requirement	Software design	Software construction	Software testing	Software maintenance	Software configuration	Software engineering management	Software engineering process	Software engineering tools and methods	Software quality

We adopted the “Bloom’s Taxonomy” [16] to represent the intensity of understanding of course content. Combining the Bloom levels and knowledge areas, we can analyse the content of each course as:

ID	Name	Knowledge Areas
SD201	Fundamental of Programming	(KA1,1) (KA2,1) (KA3,1) (KA3,2) (KA3,3) (KA4,1)
SD303	Software Development	(KA2,2) (KA5,1) (KA7,1) (KA8,1) (KA8,2) (KA10,3)
SD324	Human Computer Interaction	(KA1,1) (KA2,2) (KA2,3) (KA9,1) (KA10,1) (KA10,2)
SD389	Project Management I	(KA1,3) (KA4,2) (KA6,1) (KA7,2) (KA7,3) (KA9,2)
SD423	Project Management II	(KA2,4) (KA5,2) (KA5,3) (KA6,2) (KA9,3)

Fig. 2. A course design pattern.

By assuming Knowledge Area dependency we derive the dependency relationship of the courses as shown in Fig. 3. For any two courses C1 and C2, if:

- Independent: Both C1 and C2 can be taken at the same time and at any sequence.
- Parallel: Both C1 and C2 can be taken at the same time and at any sequence.
- C2 is dependent on C1: C1 must be completed before taking C2, which means that C1 and C2 cannot be taken together.
- C2 is parallel dependent on C1: C1 & C2 can be taken together but C2 cannot be taken before C1.

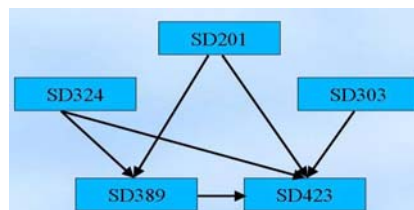


Fig. 3: An example of the dependency relationship of courses.

Assume the starting point of the program is SD201 (see Program Constraints below). Given the dependency relationship in our example we will have 4 alternate paths for students without any program background:

- SD201 → SD324 → SD303 → SD389 → SD423;
- SD201 → SD324 → SD303 → SD423 → SD389;
- SD201 → SD324 → SD389 → SD303 → SD423;
- SD201 → SD303 → SD324 → SD389 → SD423.

To achieve *individualized* course plans, the advisor agent considers the courses the students have taken before. However the presumption of this approach is that the courses can be mapped to the Knowledge Area repository. Since Student A does not have any computing background, his possible program paths will be all 4 of the above. For student B, since he already finished a SD303 equivalent course, he will have only one path: SD201 → SD324 → SD389 → SD423.

A *program constraint* is a factor that affects a program plan from the point of view of the program. In our implementation we have two constraints:

- Starting point: a student must start with a certain course.
- Semester, minimum and maximum courses: courses are grouped into a semester. Course taken in the same semester are considered as taking the courses at the same time. A student can take a minimum number of courses up to a maximum number within a semester. In our example, we prescribed that:

$$1 \leq \text{no. of courses/per semester} \leq 2$$

By applying these constraints to the above course path for student B we will have 4 alternate plans:

(S1: SD201, SD324) (S2: SD389, SD423)
 (S1: SD201, SD324) (S2: SD389) (S3: SD423)
 (S1: SD201) (S2: SD324) (S3: SDP389, SD423)
 (S1: SD201) (S2: SD324) (S3: SD389) (S4: SD423)

Program constraints are not limited to the above two examples. Other conditions, like course availability, number of credits per course and financial constraints, can be added to the consideration.

Heuristics are introduced to achieve more reasonable solutions. Considering all possible program plans and the 2 constraints, there are as many as 20 different plans for student A who does not have any computing background. Heuristics is not a new idea; it introduces rules that are obvious to human but not to computer programs. In our case, heuristics are used to eliminate excessive course paths. We introduce only one heuristic: there are not more than 3 semesters in each plan for the students. Therefore in the above example, the Advisor Agent suggests only plans #1, #2 and #3.

To take full advantage of Web services and agent architecture we did not implement everything into one single application and did not run all of the functions on the same platform. The Advisor Agent itself does not maintain/update any of the information it requires for the advising functionality. Instead, it will try to retrieve the information it requires. The Advisor Agent will keep a list of end-points of the Web services that holds the information. Namely, we have a Web service for student information, including the background of the student, a Web service for Course Repository and a Web service for Knowledge Area Repository. In the future, we would maintain a yellow pages service for the Web services in standard UDDI format so that we can eliminate the need of storing Web services end points in the Advisor Agent. Information, however, must be available on a format that can be interpreted by the Advisor Agent. The logical choice will be XML format. A sample student profile looks like:

```
<Student>
  <demographics>
    <id>1234567</id> <name>Steve Leung</name>
    ..... </demographics>
  <goal> <degree>BSc</degree>
</program>Software Engineering </program> </goal>
  <experience> <course> <source_ID>CSC121</source_ID>
    <source_name>Introduction to Programming</source_name>
    <grade>85%</grade>
  <equivalent_ID>SD201</equivalent_ID>
</course>... </experience>
```

A sample Course Repository looks like:

```
<Course_repository>
  <Course> <id>SD201</id> <description>Fundamental_of_Programming
</description>
  <KA><label>KA1</label><bloom>1</bloom></KA>
```

```

<KA><label>KA2</label><bloom>1</bloom></KA>
<KA><label>KA3</label><bloom>1</bloom></KA>
<KA><label>KA3</label><bloom>2</bloom></KA>
<KA><label>KA3</label><bloom>3</bloom></KA>
<KA><label>KA4</label><bloom>1</bloom></KA>
</Course> ... </CourseRepository>

```

A sample Knowledge Area Repository looks like:

```

<KA_repository>
  <KA> <label>KA1</label>
    <description>Software_Requirement</description>
  <parent/>
</KA>
...
</KA_repository>

```

In the future we will be developing schema for universal validation of the data. The sample of the program plans in this example is shown in Fig. 5.

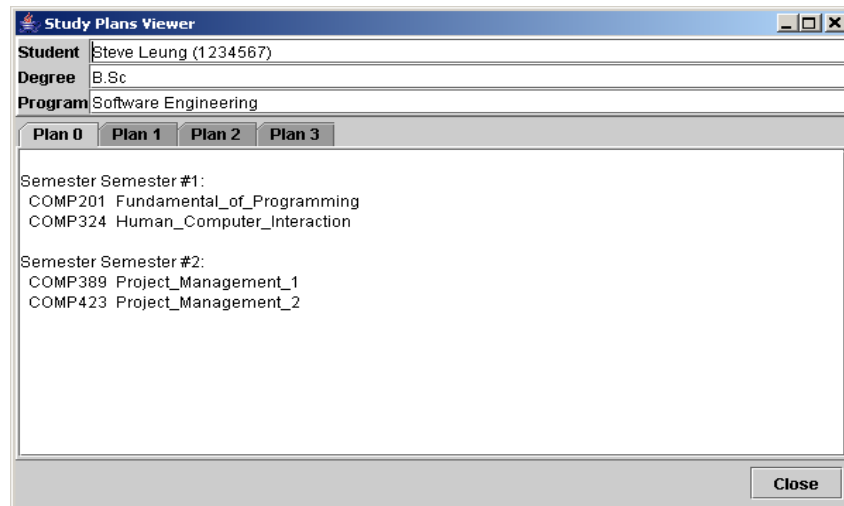


Fig. 5. A sample of the program plans.

Since the above is the client side program, it is easy to transfer the result on a Web server and make it available for enquiry using regular browsers. Moreover, one of the advantages of the agent-based approach over the Web services approach is the autonomous nature of an agent. Once the Advisor Agent starts, it will keep on running until it is instructed to stop. If there is any change to the student profile and course repository, the Advisor Agent will notice the impact on the program plans and inform the student and his academic advisor.

4 Conclusions and Future Work

Educational information standardization, educational resource development, Web technologies, and AI technologies have paved a way for Agent-Enhanced Web-based On-line Learning. To integrate agents into existing legacy learning environments or into heterogeneous learning environments, one may encounter many difficulties. Web Services technology provides a new way to integrate existing systems or applications, and the ability to access data in a heterogeneous environment and to provide interoperability of components and learning content.

We have proposed an approach to designing and developing adaptive Web-based learning environments by integrating agents and Web services. Some agents and Web services have been developed for both real applications and experiments.

We are working on the general issues related to performance monitoring and adaptation mechanism in the architecture. Also we will explore Adaptive E-Learning Based on Distributed Re-usable Learning Activities [17] and how to apply the current standardization efforts related to the Web Services Choreography [18] to the coordination of the agents in the architecture. Through our research into Web-based course generation and delivery, we are creating a distributed intelligent agent system that will allow instructors to generate timely, personalized and adaptive courses best suited for each student.

Acknowledgements

We would like to thank National Science & Engineering Research Council (NSERC) of Canada for sponsoring this research project.

References

1. Holt, P., F. Lin, H. Wang, G. Pu, A Learner-Centered Distributed Learning Environment, WSEAS Transactions on Communications, Issues 2-3, April 2003 and July 2003, ISSN: 1109-2742, pp271-276.
2. Chan, T-W, (1995), Artificial Agents in Distance Learning, International Journal of Educational Telecommunications, 1(2/3), 263-282.
3. Greer J., McCalla G., Vassileva J., Deters R., Bull S., Kettel L., (2001) Lessons Learned in Deploying a Multi-Agent Learning Support System: The I-Help Experience, *Proceedings of AIED'2001*, San Antonio, 410-421.
4. Baylor, A., (1999). Intelligent agents as cognitive tools for education. *Educational Technology*, Volume XXXIX (2), 36-41.
5. Thaiupathump, C., J. Bourne, J. Olin Campbell, Intelligent Agents for Online Learning, JALN Vol.3, Issue 2, November 1999.
6. Sturm, A., Dovi, D., Shehory, O., (2003), Single-Model Method for Specifying Multi-Agent Systems. *Proceedings of AAMAS, 2003, Melbourne*. 121 – 128.
7. Blackmon, W., & Rehak, D. (2003). Customized Learning: A Web Services Approach. *World Conference on Educational Multimedia, Hypermedia and Telecommunications 2003*(1), 6-9.
8. Huhns, M. N., (2002), Agents as Web Services, *IEEE Internet Computing*, July/August, 93-95.
9. Heaton, Jeff, Programming Spiders, Bots, and Aggregators in Java, 2002, Sybex.
10. F. Lin and L. Poon, Integrating Web Services and Agent Technology for E-Learning Course Content Maintenance, *Innovations in Applied Artificial Intelligence*, B. Orchard, C. Yang, M. Ali (eds.), LNAI 3029, Springer, pp. 848-856.
11. F. Lin and L. Esmahi, Integrating Agent Technology and Web Services into Distributed Learning Environments, in book edited by F. Lin, *Designing Distributed Learning Environments with Intelligent Software Agents*, IGP, 2004.
12. Hatala, M., G. Richards, T. Eap, and J. Willms, The eduSource Communication Language: implementing open network for learning repositories and services, Symposium on Applied Computing Proceedings of the 2004 ACM symposium on Applied Computing, Pages: 957-962, 2004, ISBN:1-58113-812-1
13. Hogeboom, M., (Master Thesis), Domain Modelling for E-Learning Using PROTÉGÉ 2000 with ontology Web Services, Athabasca University, 2004.
14. Hogeboom, M. and F. Lin, Developing Domain Model Web Services for Agent-Supported distributed Learning Using PROTÉGÉ 2000, IEEE Learning Technology newsletter, vol. 6 Issue 2, April 2004.
15. Y. Cao, F. Lin, R. McGreal, S. Schafer, N. Friesen, T. Tin, T. Anderson, D. Kariel, B. Powell, and M. Anderson, Facilitating E-Learning with a MARC to IEEE LOM Metadata Crosswalk Application, *Innovations in Applied Artificial Intelligence*, B. Orchard, C. Yang, M. Ali (eds.), LNAI 3029, Springer, pp. 739-748.
16. Anderson, L.W., & Krathwohl (Eds.). (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman.
17. Brusilovsky, P. and Nijhawan, H. (2002) A Framework for Adaptive E-Learning Based on Distributed Re-usable Learning Activities. In: M. Driscoll and T. C. Reeves (eds.) *Proceedings of World Conference on E-Learning, E-Learn 2002*, Montreal, Canada, October 15-19, 2002, AACE, pp. 154-161.
18. Web Services choreography: <http://www.webservices.org/index.php/article/view/1178/>