KEEP THE WEB SERVER COOL: A PROPOSAL FOR SERVER-SIDE OBJECT DEVELOPMENT FOR ONLINE COURSES

Gangmeng Ji, Albert Ip, Ric Canale and Paul Fritze

Multimedia Education Unit, University of Melbourne, Australia.
email: g.ji@meu.unimelb.edu.au
http://www.meu.unimelb.edu.au/meu/staff/Ji.shtml

email: a.ip@meu.unimelb.edu.au
http://www.eddy.meu.unimelb.edu.au/albert/

email: r.canale@meu.unimelb.edu.au
http://www.meu.unimelb.edu.au/meu/staff/Canale.html

email: p.fritze@meu.unimelb.edu.au
http://www.meu.unimelb.edu.au/meu/staff/Fritze.html

ABSTRACT

This paper introduces a framework called the COOL framework for developing server-side software components or complete systems for online courses. The framework has been designed to eliminate the limitations facing the integration of commercial web-based course delivery software and in-house pedagogical innovations on the server end. With the adoption of the framework, complete server systems can allow third-party software components to be incorporated into and pedagogical innovations can be re-used to their full potential. The COOL framework provides a middle layer 'wrapper' that exposes the server-side components as URLs, which can be accessible to any web delivery system. The framework is being adopted in a number of projects.

KEY WORDS

Online education, course delivery system, development framework, technology in education, Internet and education, web delivery system, distance learning.

1. BACKGROUND

Along with widespread use of the Internet, particularly the web, the education sector has been quick to adopt this evolving and challenging technology for the purposes of teaching and learning. Some universities, like the University of Melbourne, see Internet technology as a significant part of their strategy for maintaining and extending the quality of their educational programs into the future, as an opportunity for restructuring their curricula, and as a chance for adopting flexible approaches to teaching and learning. The growth of interest in online course delivery has created a demand for online course delivery systems. In fact, both the commercial solution providers and educational institutions have seized on the opportunity to develop online course delivery systems. There are, however, significant differences between these two streams for utilizing the online technology. The former intends to develop generic and cross discipline online systems for delivering courses comprising course material and

resource management, student management, and student tracking utilities, web conferencing, threaded discussion forums, and bulletin boards. Those complete systems are often expensive to develop and maintain. They are beyond the capabilities of the typical in-house developments of all but a few educational institutions. On the other hand, many educators are more interested in using Internet technology to visualize their pedagogical innovations. Their interests are often based on projects, which represent an amalgam of institutional, departmental and individual understandings of educational issues (Fritze & McTigue, 1997). Conversely, this side of development for online courses is often beyond the grasp of commercial solutions as it is closely tied to localized educational needs. Those pedagogical innovations may require rich interactive learning activities. Some are generic objects across disciplines and others are highly customized for localized learning requirements.

The two perspectives of online course development coexist but usually one perspective will dominate in any particular course or project. Full integration of the two perspectives into the one course can be very difficult or impossible. The off-the-shelf commercial products are often self-contained and impenetrable packages. Their components can not be taken apart and do not allow themselves to be replaced or used in other applications. There is understandable commercial interest in packaging products in a way that they are self-contained and impenetrable. However, software of this nature will maximize its benefit to educational institutions only if it allows educationally motivated innovations to be added. On the other hand, the in-house pedagogical innovations are commonly developed in an ad hoc fashion. They often function only within specific systems or specific educational contexts and hence can not effectively be re-used. Ways should be sought to maximize their re-use. Technically, there are two aspects to implementing in-house pedagogical innovations, namely client-side development and server-side development. Effective client-side development of pedagogical solutions has been addressed in the literature (Ip et al, 1997; Fritze & McTigue, 1997). This paper focuses on the server-side implementation of pedagogical innovations though they are eventually materialized on the client-side web pages.

Closed systems, whether 'home grown' or acquired commercially, limit educational creativity since the products do not allow innovative solutions to be added except perhaps by the originators themselves. This observation came out of a systems review project called EdWebEval (http://www2.meu.unimelb.edu.au/EdWebEval). The project evaluates the major current Web delivery systems including Topclass (http://www.wbtsystems.com), WebCT (http://homebrew.cs.ubc.ca/webct), and Virtual U (http://virtual-u.cs.sfu.ca). It has been found that pedagogically innovative server-side tools such as the Text Analysis Object TAO, the Learning Evaluation Online LEO, the html document annotation tool ANN, the Question Plus Qplus (http://www2.meu.unimelb.edu.au/oxygen/tools), and the email forwarder MailTo (http://www2.meu.unimelb.edu.au/le/about3/support/Lecontact.html) are difficult or impossible to incorporate into the server systems being reviewed. Since the evaluation was carried out, two notable new commercial offerings have appeared – WebMentor (released by Avilar in January 1998, http://www.avilar.com), and MelbourneIT Creator (released by MelbourneIT in August, 1998, http://www.melbourneit.com.au). WebMentor is fully compatible with the framework proposed in this paper. At the time of writing, MelbourneIT Creator has not been trialled by the authors, however it appears to be a very capable and promising product.

This paper proposes a solution called COOL (Component-based Object Library). The proposal is a framework for server-side development of online course systems, which will cater for appropriately designed commercial solutions and in-house innovations. The COOL framework fits into a global architecture that includes the frameworks of server and client side development. This architecture is the subject of on-going development at the Multimedia Education Unit. The paper describes the development of server-side components within the global architecture.

2. PRINCIPLES OF THE COOL FRAMEWORK

The COOL is more than just a library of objects developed for the server systems. It is a framework for the development of re-usable server-side components or complete server systems with the aim to:

- adopt a component-based technical design for software;
- design with re-usability in mind;
- use or provide a 'pluggable' architecture;
- provide access to components cross-platform; and
- comply with relevant international standards.

2.1 COMPONENT-BASED

The first principle is that the development of Web systems should be component-based. The idea of component-based design originates from the method of block design developed by Ericsson, a Swedish telecommunications company, in the1960s. It has been widely adopted not only in telecommunications but also in other software development areas (Jacobson et al, 1996). This design principle forms the basis of software re-use. The component is a high-level module or block of a complex system, which contains a public interface. The public interface is formally documented and exposes its functions to the outside world. The public interface makes the software component re-usable. The component-based approach to server-side development has to some extent been ignored because many institutions rely on off-the-shelf products which they believe will provide the required systems for delivering their web courses. This assumption will stop any innovative thinking on how these systems will be developed though it has been argued that component-based courseware development will benefit the institutions tremendously (Ip et al, 1997). Moreover, the component-based approach makes the complex educational issues more manageable and deliverable. It especially suits the research and development practices of educators wishing to develop innovative pedagogies.

2.2 RE-USABLE

This principle defines how advantage can be taken of the component-based principle, which of itself does not guarantee that the components can be re-used. It is often found that software components can only be used for a specific application. The component-based principle is often practiced to reduce the complexity of a system under development. Software components should be developed with a vision to re-use them in a variety of applications. For example, an email forwarder called MailTo was developed initially for an on-line slide ordering system (http://www.meu.unimelb.edu.au/sectionWebsites/photo/pics.html). However, the author chose to implement it in a way that it can be used for systems other than the slide ordering system. It can be used in a course delivery system for students to send queries about certain topics anywhere without using any personal email applications (http://www2.meu.unimelb.edu.au/le/ about8/ Support/LEcontact.html).

2.3 PLUGGABLE

This principle extends the re-usability of server-side software components. Component-based software design and implementation may not result in software modules that are easy to use or that can be used in a number of applications. Being pluggable means the software components are developed with a vision to be generic and to be widely used. Their public interfaces should be implemented using certain standards and be well documented. They are not developed just for one specific system. They should be developed in a way that any system can easily 'hook' them. This requirement creates a challenge for the commercial products, as most products tend to be self-contained. It is generally accepted that the intellectual property of a product or software component can be protected by making the technology proprietary but proprietary technology does not exclude being pluggable. It does require greater care to observe international

standards and perhaps a change in business culture. If software components are pluggable, then the systems they become part of can be easily enhanced because the complexity of building the systems has been reduced.

Being pluggable also means software products or components can allow other software components to be part of them or to interface with them. For instance, a student administration system should allow any course delivery system to send student scores to it and take the results received as part of the student academic record.

2.4 CROSS-PLATFORM ACCESSIBLE

The principle of being cross-platform accessible further extends the re-usability of server-side software components. Often server-side components either freely downloaded via the Internet or developed for some projects are for specific platforms. Duplicating them on different platforms is often totally unnecessary and a waste of effort. Those components either in the form of a Java bean or Microsoft Com object can be wrapped and exposed as a URL which can be used by any course delivery system. The MailTo server-side utility mentioned above is actually a URL, which was built on top of a Java bean called JavaEmailer for the platform of Windows NT.

2.5 STANDARDS COMPLIANT

The last but not least principle defines the communication protocol between the server-side components. The communication protocol describes the formats of messages passed between the components. This requirement will facilitate the global adoption of the proposed framework for the development of server-side software components. Both the XML and the URL encoding mechanism are what is proposed by this framework. With the formats defined, the configurations of a server component and results to be returned can be understood by any participating component.

3. SPECIFICATION OF THE COOL FRAMEWORK

With the five principles, the framework for server-side development is implemented as shown in Figure 1. The COOL framework consists of three layers. On the bottom of the framework are the platform-dependent low-level software components developed with a variety of technologies including Microsoft COM and DCOM, Java, and CORBA. They are often difficult to be re-used by applications of operating systems other than the systems they sit in. They need to be wrapped in a way that they can be used to their full potential, which is what the middle layer is expected to achieve.

The middle layer of the COOL framework packages the platform-dependent software components in a way that are easy to be accessed by server-side web applications or software components. The packaging or wrapping can be implemented using a variety of technologies so far available. The predominant technologies include Microsoft Active Server Pages (ASP), Perl, and CGI. These technologies extend the public interfaces of the software components of the bottom layer and make them easily accessible.

The top layer is what the server-side components present themselves to the outside world. This is a layer that exposes the server-side components as URLs. Exposing server-side components as URLs makes them accessible to any web delivery system. They can easily work with client-side objects (i.e. Shockwave, Applets, and Scriptlets) hosted by web browsers.

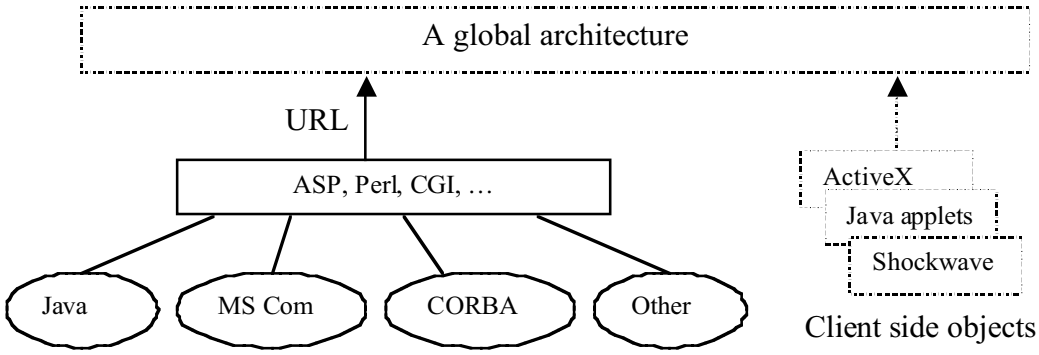

Figure 1: The COOL framework operates within a global architecture that connects both server-side and client-side components

As implied in the fifth principle of the COOL framework, the format of messages is critical for components to communicate with each other. Using URLs alone cannot deliver a total solution. Messaging among components must be in a globally accepted format. The current formats are the XML and the URL encoding mechanism. The exact details for specifying the formats of messages using XML and the URL encoding mechanism are part of the ongoing project at the Unit. They are going to be resolved by the specification of a global architecture.

4. ADOPTION OF THE COOL FRAMEWORK

A number of server-side software components or objects have been developed in the Unit for specific projects. The majority of these COOL components is generic and can be used in a number of scenarios. Examples of these components are ANN, Qplus, Text Analysis Object, MailTo, MCQ Builder, ClozeBuilder, and GlossConverter. Those examples are listed in the following mainly to illustrate that there is an ongoing need for pedagogical innovations and how the COOL framework has been adopted. The technical implementation of each software component is beyond the scope of this paper. It is important to point out the difference between how these COOL components are used with open course delivery systems and how they are used with closed course delivery systems. The COOL components can be used with close systems, but it is often found necessary to replicate the functions and data storage already with the systems. On the other hand, the COOL components can be integrated into open course delivery systems. These systems are built with open architectures. There is no need to replicate any functions and data storage since they are accessible to the COOL components.

4.1 WEBMENTOR

WebMentor is a commercial web delivery system for online courses with an open architecture developed by Avilar (http://www.avilar.com) in early 1998. This course delivery system is so far the only one evaluated by the authors in the market that is open for customization and allows easy addition of third-party software components. This product has been bought for a major on-line course in Chinese described below. We have been able to add all of the pedagogical innovations requiring server-side software components and integrate them with this comprehensive online course delivery system. Examples will be provided in the following section. The user interface, navigation engine, its system functions, and its back-end databases are open for customization. WebMentor itself has been implemented using an XML-like tagging system, which leaves room for future enhancement and extension. A technical example for illustration is the specification of attributes of a course (Figure 2).

```
[COURSE name = "EXAMPLE" title = "Example Course Title"]
This is a sample description of the example course. This course is to
teach …
[COPYRIGHT name = "Your Institution" year = "1998"]
&copy;1998 Your Institution. All Rights Reserved.<br>
[/COPYRIGHT]
[AUTHOR name = "Your Name"
email="YourEmail@YourDomain.com"]
Provide a brief bio about yourself.
[/AUTHOR]
[/COURSE]
```

Figure 2: XML-like tagging system in WebMentor

4.2 BRIDGES TO CHINA.

Bridges to China is a project for NALSAS (National Asian Language Studies in Australian Schools), which is funded by the Commonwealth government. This is a project for developing an online course for teaching the Chinese language to the Australian teachers in the schools. The project is challenging for a number of reasons. First, the course will be delivered cross-platform. It is expected to run on both Macintosh and IBM-compatible PCs. Since this project is language specific, it requires the course delivery system to be adaptive to the client platform. For example, the Chinese character representation varies from platform to platform. Only systems with open architectures can provide a satisfactory solution to this problem. An open architecture has allowed the development of a number of generic pedagogical innovations to come from this project. Finally, streamlining the authoring process of interactive web pages is greatly facilitated by the open architecture. The development of both these pedagogical innovations and page authoring utilities follows the COOL frameworks. Most of the COOL components can either be used by or integrated into other web delivery systems.

4.3 ANN

ANN (Figure 3) is a tool that allows readers to annotate presented html pages. The readers' annotations are stored in a database, which is separate from the annotated documents. The annotations become accessible to any reader of these documents or pages (http://www2.meu.unimelb.edu.au/oxygen/tools). This tool is quite generic and can be used in a number of educational contexts. The tool was developed initially for a high school subject where many poems were put up on line as html pages. With this tool, the students and teachers can comment on the online poems, which turns the web site into a valuable resource for poems.

This annotation tool qualifies as a COOL component (a server-side software component whose implementation follows the COOL framework). It is accessible to any web delivery system although ANN can be best integrated into those course delivery systems with open architectures because open systems like WebMentor are implemented using industry-standard database engines.

Figure 3: ANN – a COOL component that provides online annotation

4.4 QPLUS

Question Plus (Figure 4) extends the functionality of ANN. In QPlus, a question is presented to students. As submissions are made by different students, they are made accessible, but only to others who have already submitted an answer. QPlus is particularly suitable for use in distance education and it is used extensively in the Bridges to China project. It is used in this project to set the scene for discussion activities amongst Chinese language learners. In other words, the students start with the initial submission to QPlus. Following these initial, unprompted submissions, the discussion is then developed using the standard web conference tool with this on-line course.

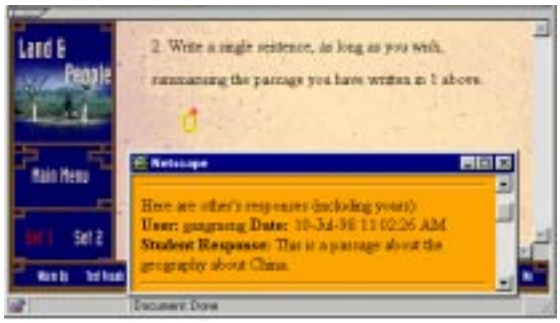

Figure 4: QPlus – a COOL component that is used in the Bridges to China

QPlus demonstrates the importance of openness of course delivery systems. In this case, the desired functionality requires access to student identification information and to student records database of the system.

4.5 TAO

TAO, Text Analysis Object (Figure 5), is a new class of tool to analyze free text responses from learners. The answer to a question consists of concepts and details, which can be scored separately (http://www2.meu.unimelb.edu.au/oxygen/tools). Feedback is provided to help the student get the concepts and details correct. TAO was developed for a medical project where there was a need to analyze free text entries from on-line tutorials. Free text entry has always been a challenge for computer assisted learning packages. The invention of major concepts and minor concepts or details has helped to solve the problem for processing free text entries. With this tool, the lecturer can predefine major concepts and minor concepts for each question. With the major concepts and minor concepts defined, the tutorial system can provide informative feedback to the students by saying what major concepts and details have been included in their answer and by pointing out what they are.

TAO is one of the pedagogical innovations that show what commercial solutions can not provide. Those kinds of innovations have to come from educational institutions. However, tools like this one can be easily incorporated into a course delivery system if it is open and allows room for third-party components.

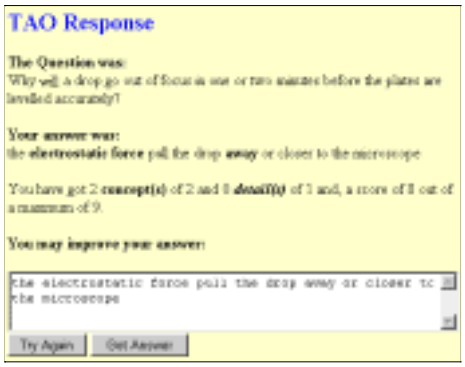

Figure 5: An example of Text Analysis Object in use

4.6 MCQ BUILDER

MCQ Builder is a server-side component implemented using OXYGEN (an online course authoring engine, http://www2.meu.unimelb.edu.au/oxygen), which helps course instructors to compose interactive multiple choice questions. The tool generates dynamic html and Javascript functions to build interactive multiple-choice questions. The tool is easy to use. All it requires is to ask the user to specify the question stem, the options, and the key for each question, and the scoring scheme. Figure 6 is an example of a multiple-choice question generated by MCQ Builder.

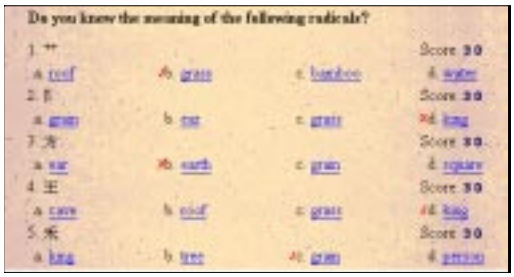

Figure 6: An interactive multiple-choice question generated by MCQ Builder

The interactive multiple-choice question as shown in Figure 6 is implemented using dynamic html. A user clicking on a wrong answer gets a cross. A tick and a score are received if a right answer is clicked (the obtained score depends on the number of attempts). More informative feedback can be implemented in a different version of this interactive multiple-choice question. The facility for authoring this type of questions is not available with any course delivery system. The need for interactive multiple-choice questions in the Bridges to China project has led to the development of MCQ Builder. This tool aims at streamlining the process of generating hundreds of interactive multiple-choice questions for this on-line course. The streamlining process or tool is often needed for large-scale course development projects.

4.7 CLOZEBUILDER

The ClozeBuilder is a server-side software component that converts a color-encoded text into a cloze exercise or test. Cloze tests are commonly needed in a foreign or second language course. This COOL component was specifically developed to fulfil one of the requirements of the Bridges to China project. The ClozeBuilder facilitates the authoring of cloze tests. Any user capable of using a simple html editor like Symantec Visual Page, Claris Home Page, or Microsoft Frontpage can simply highlight the words or phrases to be taken out using a consistent color. The html page is then submitted to the ClozeBuilder for conversion 'on the fly'. ClozeBuilder searches for color-encoded words replacing them with underlined blanks. The application randomizes those words to build a palette on the top of the text or passage. Finally, Javascript functions are incorporated into the page to realize the 'drag and drop' functionality. This functionality is implemented using dHTML, which is an enhanced version of HTML that permits animation of screen objects. Again, this tool has been developed in response to specific educational needs not to be found with any commercial course delivery system.

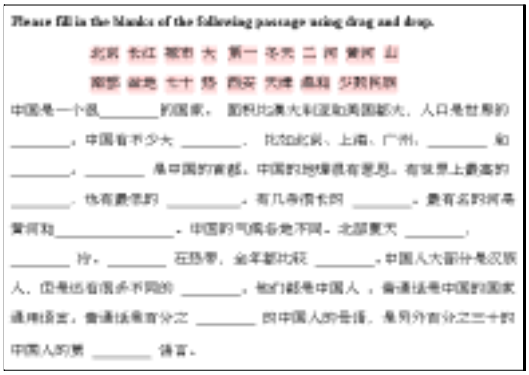

Figure 7: A cloze test generated by ClozeBuilder

4.8 COLLABORATION WITH CLIENT-SIDE OBJECTS

The component-based approach can be used to enhance the operation of independently developed client-side objects. In Figure 8, an 'Itemset' tutorial-style question object is used to provide open-ended question learning activities. The Itemset is developed as a scriptable Director Shockwave movie as a component in the 'Learning Engine' object model (Fritze & Ip, 1998). A question has been set up in the Itemset that will match student typed input against a set of criteria, and in turn respond with appropriate feedback depending on the student entry. A problem common to Shockwave and Java applets is that system-level Copy and Paste functions are disabled for security reasons. This problem is solved here by calling an external 'Text Edit' COOL component. The Itemset creates the text edit window by passing a message in the form of a URL via a simple JavaScript function on the page to the server. Text entered by the student is retrieved from this window and evaluated by the Itemset object. Feedback is provided based on performance against the criteria and is presented to the student in another COOL component, the 'Explanation'. The third COOL component used in this example is the MailTo component, which provides a questionnaire form that is automatically emailed to a given address without the student requiring a direct connection to an email system.

Figure 8: Shockwave tutorial object (top) inter-operating with COOL components

While it is possible to build all these functionalities into a single object, there are obvious advantages in constructing learning activities out of smaller independent objects developed in different software technologies. The ability to inter-operate is made practical by the simple interface wrapper provided by the COOL framework.

5. CONCLUSION

The COOL framework would typically provide benefits to institutions where there are a number of heterogeneous server systems in use. Software components developed natively for those systems can be re-used by implementing a 'wrapper' so that any web system can take advantage of them. The extent of their usability depends on the openness of the course delivery systems. Closed systems can embed the COOL components, but extra functions and data storage are often found to be replicated for them to function. The COOL components tend to work better with open systems as their system variables, functions and backend databases are accessible. Full integration of the COOL components has proven to be possible in open course delivery systems.

The adoption of the COOL framework will greatly enhance project development. Any department or faculty following this approach will benefit by not only delivering required software products but also providing a solid architecture of course development. The library of server-side software components that becomes available from finished projects will be ready for re-use in future projects. This will greatly reduce the cost of on-line course development.

Systems delivered following the COOL framework can be easily reconfigured and extended. Any future enhancement can be added to the existing systems since they are pluggable.

6. REFERENCE

Fritze, P. & McTigue, P. (1997) Learning Engines - a Framework for the Creation of Interactive Learning Components on the Web. http://www.curtin.edu.au/conference/ascilite97/papers/Fritze/ Fritze.html

Fritze, P. & Ip, P. (1998) Learning Engines - a functional object model for developing learning resources for the Web. In T Ottman & I Tomek (Eds.), Proceedings of ED_MEDIA & ED-TELECOM 98 Conference (pp. 342-7). Freiburg: Association for the Advancement of Computing in Education.

Ip, A., Canale, R., Fritze, P. & Ji, G. (1997) Enabling Re-usability of Courseware Components with Web-based Virtual Apparatus. http://www.curtin.edu.au/conference/ascilite97/papers/Ip/ Ip.html

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1996) Object-Oriented Software Engineering: a use case driven approach, Addison-Wesley Longman Limited.