OBJECT-ORIENTED ARCHITECTURE OF A MODERN LEARNING MANAGEMENT SYSTEM

Frank Kappe, Nick Sceerbakov

Institute of Interactive Systems and Data Science, Graz University of Technology (AUSTRIA)

Abstract

Learning Management Systems (LMS) are software systems to implement different e-learning scenarios in practice. Developing LMS that satisfy the needs and desires of teachers is crucial in e-learning. In a simple case, teachers need to provide a set of files to be downloaded by students (library) and participate in an asynchronous discussion (discussion forum). In more complex cases, students are provided with synchronous communication and collaboration tools, implement practical assignments, use virtual reality landscapes, etc. Teachers see a particular LMS from the perspective whether it is capable of implementing an e-learning scenario needed by the teacher, or not. Thus, a modern LMS should demonstrate the ability to implement a variety of different e-learning scenarios, and should support an evolutionary development paradigm to adopt the system functionality to permanent changes of user requirements.

This paper proposes a novel object-oriented architecture for implementing learning management systems suitable for evolutionary development of variety of e-learning scenarios on a single software platform.

Keywords: WEB application, Learning Management System, Architecture.

1 INTRODUCTION

Learning Management Systems (LMS) are software systems to implement different e-learning scenarios in practice. Developing LMS that satisfy the needs and desires of teachers [1, 2, 3] is crucial in e-learning. In a simple case, teachers need to provide a set of files to be downloaded by students (library) and participate in an asynchronous discussion (discussion forum) [4]. In more complex cases, students are provided with synchronous communication and collaboration tools, implement practical assignments, use virtual reality landscapes, etc. [5, 6, 7]. Teachers see a particular LMS from the perspective whether it is capable of implementing an e-learning scenario needed by the teacher, or not. Thus, a modern LMS should demonstrate the ability to implement a variety of different e-learning scenarios, and should support an evolutionary development paradigm to adopt the system functionality to permanent changes of user requirements [8, 9].

In accordance with recent trends in developing modern internet applications we distinguish between client-side (front-end) and server-side (back-end) software components [9]. Communication between such components is carried out by means of HTTP requests generated on the client-side, and HTTP responses from the server-site.

2 BACK-END COMPONENTS

A back-end layer consists of addressable Abstract Data Objects (ADO). ADOs roughly corresponding to main components of an e-learning course such as: course announcements, course curriculum, course library, course forum, course chat, etc. ADO does nothing but encapsulates private memory together with some methods (i.e. software components). Thus, ADO may be seen as a combination of persistent private memory and methods that can be applied to elements of the private memory. ADO has a unique identifier and, hence, is addressable on the cloud. There may be so-called basic ADO that represents basic data types and applications that are used for a particular training course. Typical examples of such basic ADOs are "HTML document", "PDF File", "Movie", "Question", etc. There is also composite ADO that encapsulates other ADOs as so-called members. Often, members are combined into a navigable structure (see below). This structure is in fact a link structure expressing the relationships or associations between members. A member is either a basic or composite ADO.
Therefore, a particular e-learning course can be seen as an ADO encapsulating all the necessary course components as other ADOs. For example, a simple e-learning course (see Fig. 2) may encapsulate an announcements object, library object and forum object. The methods applicable to ADOs provide all the functionality needed from the teacher and student perspectives. For example, announcement objects can be edited, created, deleted and accessed.

3 DATA CLASSES

A particular ADO is an instance of one of predefined types of ADO such as "Course", "Announcements", "Curriculum", "Library", "Quiz", "Forum", etc. Each type or Data Class (DC) defines a particular functionality of instances of this Data Class (ADO).

Link topology defines a navigable structure that is used to browse the members. For example, members of an "announcements" object are sorted by the date; members of "library" objects are structured as folders and sub-folders, etc. The navigable structure is visualized on the user screen as a particular ADO is accessed. For example, members of ADO can be visualized as tabs (see Fig. 3), drop-down list, folders and sub-folders (See Fig. 4), clickable or scrollable lists, etc.

All Data Classes share a common interface (methods):

- the "Create" method is used to create a new instance of this class;
- the "Insert" method allows defining certain ADO as a member of another ADO. Please note that a single ADO may be a member of a number of different ADO. For example, one and the same "Announcement" object may be a member of two or more different courses.
- the "Remove" method allows canceling participation of a member ADO in private memory of another ADO.
- the "Delete" method simply deletes a selected ADO. Note the members of such deleted ADO are not affected.
- the "Access" method simply visualizes a particular ADO, i.e. members are visualized using a predefined template.
For example, if the course shown on the Fig.2 is accessed, the first member "Announcements" is accessed automatically, and the user screen looks as on Fig.3

Note that members of the "course" ADO are visualized as tabs on the top. This way of visualization is predefined for all instances of this class, i.e. for all training courses. Obviously number of course members can be easily modified using insert/remove methods. The number of tabs will be modified accordingly.

It should be especially noted that there are a number of "Course" Data Classes that provide different paradigms for visualization and access to the course content. For example, "Mobile Course" Data Class may encapsulate the same ADOs as an instance of ordinary "Course" ADO, but provide entirely different access paradigm (See Fig.4).

Basically, we define the following main Data Classes:

- the "Course" object is a container for other ADOs that implement a particular training scenario for the training course.
- the "Announcements" object is a list of members; normally members are ordinary HTML or text documents. All the announcements are visualized as a scrollable list of members.
- the "Curriculum" object is a set of HTML or other documents that is visualized as a pull-down menu of members. Normally, these documents describe schedule, goals, prerequisites and requirements for the course.
- the "Library" object is a collection of other documents that are visualized as a tree-like structure consisting of folders and sub-folders.
- the "Teaching Aids" object is an object similar to a "Library" object, but members of such "Teaching Aids" instance are other applications. For example, "Quiz", "User Uploads", etc. are data objects. Such "Application" objects can be inserted into a "Teaching Aids" ADO.
- additionally to the previously mentioned data classes, the system supports a big number of purpose oriented applications such that "Quiz", "User Uploads", "Chat", "Discussion Forum", "E-Book", etc.
For example, a user may click on the Tab "Course Library" (see Fig.3) to access another member of the course (see Fig.2). The selected ADO gets the message "Access", and the object is visualized as defined by its data class ("Library" in this particular case).
4 CREATING AND MODIFYING A TRAINING COURSE

Building an online training course typically follows a top-down approach.

- First, a particular course is created and gets an id, title and general properties (open date, close date, access restrictions, etc.). Thus, the course is an "empty" container that can host a number of functional components.

- Second, the teacher defines a set of the course components needed to implement a desired training scenario. In a simplest case, an "Announcements", "Curriculum" and "Library" are sufficient. Obviously, teachers may use more data objects such as "Discussion Forum", "Chat", "Quiz", "Opinion Poll", etc. to implement a desired training scenario. All the objects that are needed for the course are created as instances of corresponding data classes and inserted into the "Course" object. For example, the teacher can create an empty "Announcements", an empty "Curriculum" and empty "Library" objects thus creating a template for a new training course.

- Finally, multimedia page, textual documents and/or movies are created. They then become members of previously defined ADOs as above.

Once a training course has been created, members can be inserted, modified or removed in a very flexible way. Insertion of a new member into the "Announcement" object, for example, will automatically update the list of all course announcements. Conversely, removing a member will remove an entry from the list. Similar, adding and deletion of files in the course library will modify content of folders from the user perspective. With the course library, of course, we must also specify the insertion position (target folder) in the hierarchy.

Insertion/Removing basic objects into/from predefined functional objects of the course provides a so-called vertical scaling of the course, that is, content of the course is modified without affecting predefined training scenario. In other words, vertical scaling deals with adding/modifying individual announcements, files, curriculum entries, etc. Horizontal scaling deals with adding/modifying structural course element such as user uploading room, forum, chat, etc. Horizontal scaling obviously affect the training scenario as such, thus, students may be additionally requested to upload their home assignments - adding a new ADO "User Uploads", or students can be obliged to make an online quiz - adding a new ADO "Quiz", etc.

Please note that the navigable structure is separated from rather than embedded in the members. This is of course essential since a member can be re-used in another ADO with a different navigational paradigm.

5 CONCLUSIONS

It should be apparent that the proposed architecture is highly modular. Documents and objects can be created independently but at the same time can be re-used through a flexible nesting or containment mechanism. This greatly facilitates multi-author development of training courses, involving mainly:

- creating basic documents (using some appropriate HTML/text editor application or any other multimedia authoring facilities)
- creating ADO of appropriate Data Class (determining therefore their functionality)
- re-using previously created documents and ADOs to define the contents of an object being created

Any ADO can be modified at any time and, as mentioned above; internal navigational structure will be automatically updated as members are added/removed. Note, however, that removing a member does not actually delete it from the system. To delete an object, we must do so explicitly. All objects containing the one being deleted will be updated accordingly.

The proposed architecture essentially replaces the "spaghetti" view of the internet course with more structured, independent but fully compatible functional modules called Abstract Data Objects.

In summary, the following features of the proposed architecture distinguish it from other existing LMS:

- Navigational structure neither belongs to individual nodes/documents. Instead, the structure is encapsulated in containers called Abstract Data Objects. By definition, navigation is carried out only between members of ADO, i.e. links cannot be created to destination nodes that are outside the ADO. ADOs therefore represent well-defined chunks of information with a particular
functionality. ADOs may be re-used in different contexts without concern for superfluous links and functionality.

- **Containment of complex Abstract Data Objects within another**, referred to as "re-use" rather than "reference", is in line with object-oriented views and closer to the intended concept of sharing existing resources, particular in different contexts. The term "reference" too strongly suggests "jumping to another location" (with a fixed given context). "Re-use" on the other hand implies a (logical) embedding of the external object into the current context (switching between embedding contexts is via "Access" method available for all Data Classes).

- **Authoring** is effected by memberwise inclusion of object, as opposed to "spaghetti" linking of nodes in other architectures. Any ADO can be included in any other. Recursive membership relations remain a possibility and will allow the modeling of arbitrarily complex training courses.

- **All operations** are addressed to a particular ADO and do not affect the navigational structure of other objects. This object-oriented character of the model presents new ways of supporting the logical integrity of training courses.

There are three aspects of the architecture that should be especially pointed out:

**Flexible adaptation of course functionality**: a particular course may encapsulate an arbitrary number of different ADOs providing a wide range of functionality. At any time, certain ADO may be added or removed to adjust the course functionality to particular needs.

**Evolutionary development of the whole system**: the system may be further developed by adding new Abstract Data Classes implementing desired e-learning concepts. For example, special Data Classes implementing such innovative training scenarios as below were recently added to the system:

- **The Group Lockers and Project** are usage scenarios that allow uploading data and making it available for a teacher, by a group of students.
- **The Authoring Room** scenario provides an environment for online collaborative authoring of textual documents.
- **The File Exchange** scenario deals with a structured internet repository that can be accessed and edited by a big number of students.
- **The Course diary** is a list of tasks that are supposed to be accomplished by students throughout the course. The definition of tasks may significantly vary, but generally each task requires performing a certain element of a database application designs, for example, developing a database schema, proving that the database schema is in particular normal form, implementing a database transaction, etc.

**Reuse of course components**: the same ADO may be reused in context of different courses allowing sharing functionality of some common components. As a consequence, different courses may use a single shared library, shared forum, chat, etc. Moreover, courses that are supposed to be used by means of different client hardware and software can be implemented as different ADOs encapsulating the same or similar members.

The approach is implemented as an LMS called TeachCenter [10]. TeachCenter is used at TU Graz, Austria for a number of years. The system is under permanent development and is very suitable for so-called "open source" or "crowd source" development philosophy. Programmers simply implement new Data Classes that are subsequently reused by teachers. Teachers, in turn, define new requirements to the ADOs functionality that form a request for developing new Data Classes.

Further development of the concept leads us to implementing a so-called "Personal Learning Environment" that can be seen as a new Abstract Data Object encapsulating a number of courses of personal interest, or in more complex case, such personal learning environment may encapsulate just selected objects from training courses. For example, a student may simply select a number of "Announcement" objects to be aware about recent developments in the flow of the courses. Obviously, there may be Personal Environments for a personal usage, or environments created for student groups providing collaborative work and flexible communication within such groups.
REFERENCES


