A Web-Based OO Platform for the Development of Didactic Multimedia Collaborative Applications

David A. Fuller, Luis A. Guerrero , Jenny Zegarra
{dfuller, luguerre, jzegarra}@ing.puc.cl
Computer Science Department
Pontificia Universidad Católica de Chile

ABSTRACT

In this paper, we examine the design of a software platform to support objects to build collaborative applications on top of a Web browser. Our collaborative applications are specifically meant to support the teaching/learning process. In our platform, we recognize two different kinds of objects: those supporting didactic multimedia information and those supporting collaboration among users. The platform is divided in three layers: the data layer, the query layer and the behavior's layer. The data layer implements the objects' persistence, and consists of structured and semi-structured data. This is what we call the course memory. The query layer allows applications to query the structured and semi-structured objects with a common language. Finally, the behavior's layer encapsulates the behavior of both didactic and collaborative objects. Aspects such as objects' visualization and functionality of collaborative objects are treated here. An example of using the platform to build and to use the platform is given.

Keywords: Collaborative applications, teaching/learning process, distributed objects, multimedia.

1. Introduction

Nowadays it is not uncommon to find Web sites with university courses information, so that students can obtain not only administrative information but also a copy of the courses lessons. However, this approach has many pitfalls. First, the concept of roles is not included, and therefore, every user has exactly the same access to all the information. It is clear that the lecturers, teaching assistants, and students, may use the system differently, and therefore, may need a different behavior when accessing the same data.

Second, the system does not have flexibility to produce different information when a user is in a different stage of the teaching/learning process. For example, a student may require studying a lesson with many details if he is starting to study. Later he may require just the examples, later the exercises, and at the end, an abstract of the lesson. The current Web documents do not provide this flexibility.

Third, the current systems do not support collaboration of users between themselves. For example, the new system could support the collaborative writing of a term paper. Or else, it could provide asynchronous and structured and persistent discussions on the lessons. Or an interface to allow students, teaching assistants and the lecturer, who are geographically distributed in the city, to get together virtually at a specified time, to discuss the lesson. The system should handle collaborative tools to make this possible, as well as audio and video through a telephone line whenever possible.

In this paper, we present an object-oriented platform to support the construction and execution of collaborative applications for the teaching/learning process at university level. Collaborative applications built on top of our platform will easily be programmed, and may easily contain powerful features such as the mentioned before. Figure 1.1 shows the architecture of our platform.

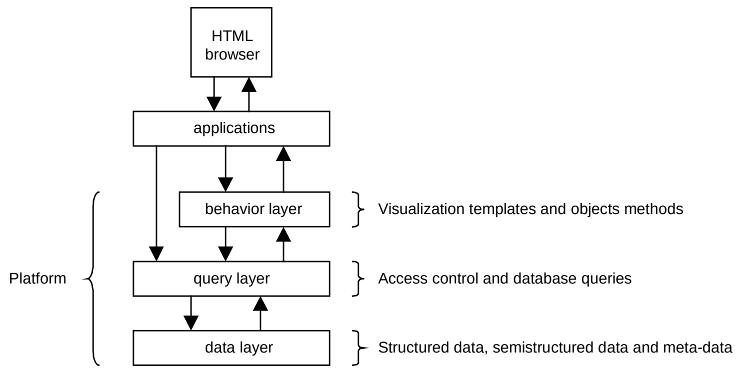

HTML browser
applications
behavior layer
query layer
data layer
Platform
Visualization templates and objects methods
Access control and database queries
Structured data, semistructured data and meta-data

Figure 1.1. *Architecture of the OO Platform*.

The platform is divided into three layers: data, query and behavioral layers. Each layer encapsulates its functionality. To build an application, the developer will require using the functionality of the query and behavioral layers. The next three sections give more details of these layers.

2. Data Architecture

In our platform, there are three types of objects that need to be stored: the course memory objects, meta-objects, and collaborative objects. In this section we will present the structure of these objects.

2.1 Course Memory

Figure 2.1 shows an OMT diagram [Rumbaugh91] with the class structure of a course and the main objects of the *course memory*.

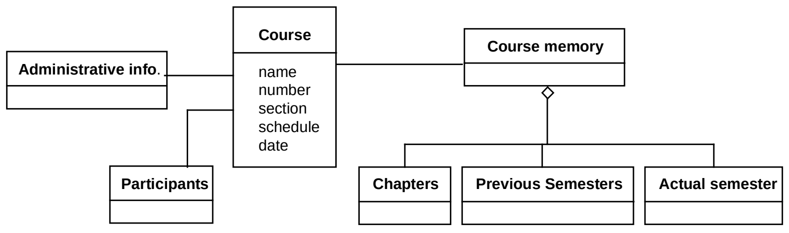


Figure 2.1 *Course Structure.*

As shown in figure 2.2, chapters can be composed of titles, text paragraphs, examples, exercises, algorithms, tables, graphics, formulas, equations, links, audio, and video. The structure of the chapters is irregular, incomplete, and does not necessarily obey to a fixed schema. Also, they can frequently change. This kind of structures is called semi-structured data [Abiteboul97]. Most of the HTML documents (e.g. HTML courses documents) have these kinds of structure.

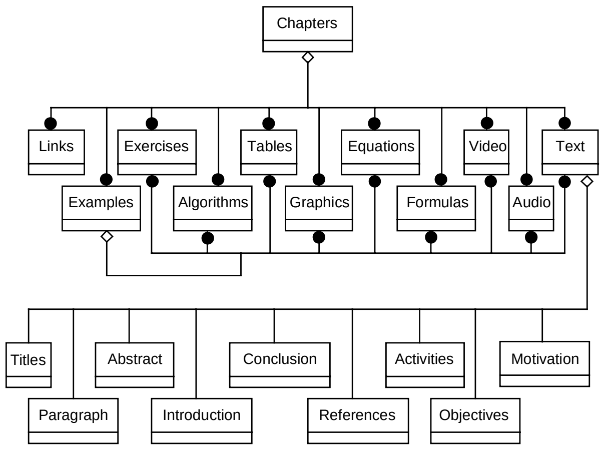


Figure 2.2 *OMT structure of a chapter.*

The above OMT diagrams present the structure of the course memory. However, due to the semi-structured feature of our chapter objects, a different data model for these objects is required. For that, we use a model called Object Exchange Model (OEM) [Goldam96]. In this model, the user is not required to previously know about the structure of the information in order to do a query. Thus, we translated the part of the OMT diagram that we recognize as semi-structured, to an OEM diagram, so that objects can now be grouped and relate themselves in descending order, and sometimes, with cyclic ways.

OEM is a flexible model with a simple heterogeneous representation for semi-structured data, based on graphs [Papakonstantinou94]. In the model, arcs represent the type of each relation. Each OEM object contains an object identifier, a descriptive textual label, a type and a value. We use two types of nodes, those that have descendants and those that have not. Those with descendants are called *atomic objects*, and can be integers, reals, strings, images, video clips, sound clips, programs and any other data value we need to store. The others are called *complex objects*, and contain arcs to the father and sons nodes.

Part of the course memory is of the structured type. Tables with a fixed structure, defined in advance represent it, and it is possible to access the data objects using simple SQL operations.

2.2 Meta-objects

We understand by meta-objects those objects that have information about objects in the data layer. For example, if we want to visualize an object in the Web browser, it is necessary to add several HTML tags to it, since they cannot be shown as they are stored. Each object has different tags with different options (such as the font size, colors, etc.).

Meta-objects have a fixed structure, and therefore, they are stored using a relational database system and accessed via SQL sentences. These objects are only used by the application, and its use is an added value to the final user.

2.3 Collaborative Objects

Collaborative objects are those objects that provide collaboration facilities to the applications, and are structured objects. Some of the collaborative objects provided by the platform are post-it, message, chat, vote, brainstorm, and the information selection matrix.

The platform also provides objects to support the process of building collaborative applications. These objects are the *box*, *canvas* and *session*. The box object is an information repository to store other type of objects. The canvas object defines the way and order in which objects in a box are visualized. The session object has a starting and ending date and time, and a floor control mechanism. The functionality of these objects is defined in the behavioral layer.

3. Queries

The queries layer provides a uniform language to the objects base, which we developed and we call C_SQL, so that the upper layers of the platform can interact with the objects stored in the objects layer. The provided query language has to hide the fact that objects can be structured, semi-structured or meta-objects. The user's role needs also be considered.

This layer receives a query in our C_SQL language, and processes it using the SQL language if it is about structured objects, or the LORE language [Abiteboul96] if it is about semi-structured objects. However, usually the queries consider both structured and semi-structured objects. In many cases, it will first be necessary to obtain structured meta-objects, and only then, do the query to the semi-structured objects, producing as result the union of objects.

Let us consider the following example: "The user JZ requests the selection of all the exercises of lesson 8 of the course ICC-2112." This information is requested by the application using C_SQL in the following way:

```
        C_SELECT lesson_8.examples AND lesson_8.exercises FROM icc-2112 WHERE login = "JZ"
```

This query is selecting objects from the ICC-2112 course, containing semi-structured information. However, due to access rights, it is first necessary to obtain her role information from a structured data table. Once the role of the user “JZ” is obtained, her access rights are added as part of the query to the semi-structured objects.

From the implementation’s point of view, semi-structured objects are stored in a repository composed of objects related by labels (paths). These objects are built according to the structure of the chapters, and are accessed by queries based on those labels. For that, automata and graph techniques are used to handle them. In our case, we use the LORE language [Abiteboul96], developed by the Stanford Database group, to handle semi-structure objects. LORE is a new database system designed to support storage and queries of semistructured data.

The C_SQL language provides instructions to create, delete, insert, eliminate and update objects. The result of C_SQL instructions is data that does not differentiate between structured or semi-structured objects. Some of the C_SQL instructions are:

```
C_SELECT field/object [,field/object] FROM File1 [,File2] WHERE condition
[AND condition] [OR condition]

C_INSERT File field/object [,field/object] VALUES value1 [,value2]

C_UPDATE File SET field/object = value [AND field/object = value]
```

4. Objects’ Behavior

The behavioral layer defines the methods that can be applied to the objects of the data layer. For the case of the objects defining the course memory, this layer defines the methods to visualize them through a Web browser. For the collaborative objects, besides their visualization, this layer also defines other methods that can be applied to them.

4.1 Visualization

The behavioral layer is in charge of adding the tags needed to visualize the objects through a Web browser. These tags are part of the object’s meta-information and are stored as templates in the relational database with the following structure:

```
create table templates (
      object_type  char(20),
      role         char(20),
      tag_def      char(200),
)
```

New objects can be visualized by adding new records in this table. Using this information, a function named `add_tags(object, obj_type, role)` adds the tags needed to visualize each object. For example, in order to add the visualization tags to an image object requested by a lecturer, the application can execute the following statement:

```
st = add_tags("image1.gif","gif","lecturer")
```

This instruction will return in `st` the string `<IMG SRC="image1.gif">.` Other objects such as titles, text, examples, exercises, algorithms, formulas and links require simple tags, similar to this.

A more complex example would be `st = add_tags("sound1.au","au","student")`. This will return the string with the following code in order to listen to an audio file:

```
<SCRIPT LANGUAGE = "JavaScript">
function PlaySound(){
   window.location = "sound1.au"
}
</SCRIPT>
<FORM>
<INPUT TYPE=’button’ NAME=’sound’ VALUE=’ Sound ’ onClick=’PlaySound()’>
</FORM>
```

When an application asks for information from a browser, a CGI takes the query and asks for the appropriate objects in the data layer. This query returns a set of objects to the behavioral layer, which calls the function `add_tags()` for each of these objects. The behavioral layer uses the templates to add the visualization tags. Finally, the CGI adds the tags so that the final page can be returned to the browser that asked for it. In this way, the Web pages are dynamically built according to user requirements.

It is possible that the same object can be visualized differently for different roles. For example, it is possible that while teaching a lesson, the lecturer wants the text with a bigger font type than a student who is studying the same lesson. To do this, the templates have a field with the role type that asks for the visualization.

The collaborative objects also have visualization tags, which most of the time are more complex than the course memory objects. For example, the *mail* object has to display a field to fill out the name of the end user, another field for the message subject, another field for the message and their respective labels. Besides this, it should show a button to send the message once it is written.

4.2 Collaborative Objects Methods

Currently we have a small list of collaborative objects, but we can add as many as we need by simply defining their data structure and their methods. Some of the collaborative objects we have are: post-it, message, chat, brainstorming, voting systems and selection matrix. Each object has its own methods defining its behavior in the application. For example, the selection matrix gives to the user the possibility to select the type of information that he will request to the data layer. This matrix has a visualization method, which is called after a previous query to the meta-information about the components of the course chapters. After showing this information, a button sends the query to the corresponding layer.

Besides these collaborative objects, there are other objects for the process of building collaborative applications such as the *box*, *canvas* and *session*. These objects also have their own methods. The box object is a recipient that allows storing and relating other objects. It has methods to create, destroy, insert and extract objects, visualizes objects and assigns users and right access for them. It also allows setting expiration dates to objects, assigns owners and defines the kind of objects that can be stored in that box.

The canvas is linked to a box and defines the order and way of visualization of each of the objects contained in that box. The main method is used to define the order of visualization of the objects contained in the box.

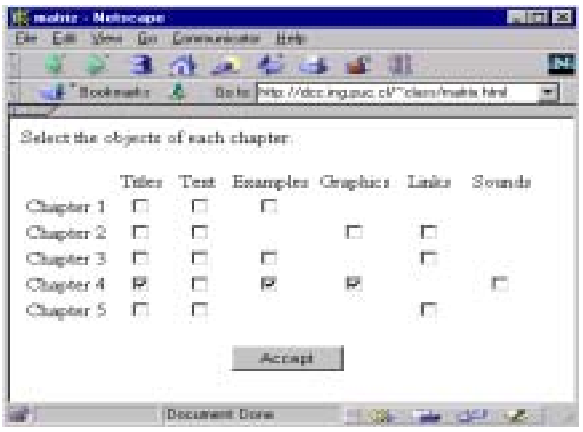
It is possible to build a mailbox using the box object. With this mailbox, users can send messages and other objects to other users. When a user uses an application built on top of CLASS, he can check his mailbox. This is one example of asynchronous communication.

New objects can be created as well as new methods for the objects that already exist. These tasks must be done in this behavior layer. Note that the functionality of objects needs to be encapsulated in this layer to allow the communication with the other layers of the platform.

To avoid concurrency problems, we use the "one writer-many readers" approach. In general, the lecturers are associated to the writers and the others are associated to readers. There is just one lecturer for each course.

5. Building Collaborative Applications

The process of building collaborative applications turns easier when we have the query language, the collaborative objects and the course memory objects already defined and built. For example, let us assume that we want to build a simple lessons browser for a lecturer to present his lessons using a Web browser. First of all we should allow the lecturer to select the subject covered in the lesson. In the platform there is an object called *didactic objects selection matrix* which looks in the data layer for the number of available chapters in the course and for the kind of objects on each chapter. The visualization of that is shown in the figure 5.1.

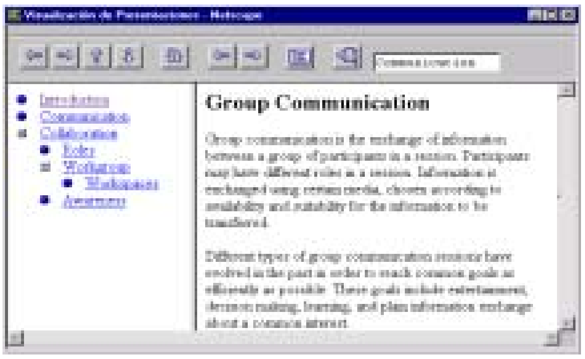

matriz - Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://dcc.ing.puc.cl/~clases/matriz.html
Accept
Document: Done

Select the objects of each chapter:

	Titles	Text	Examples	Graphics	Links	Sounds
Chapter 1	☐	☐	☐			
Chapter 2	☐	☐		☐	☐	
Chapter 3	☐	☐	☐		☐	
Chapter 4	☑	☐	☑	☑		☐
Chapter 5	☐	☐			☐	

Figure 5.1. *Selection matrix of objects by chapter*

In this case, the lecturer is selecting for his lesson the titles, examples and graphics that are present in chapter 4. Once he selects the needed information, he should be provided with the navigation tools that allow him to present the lesson. In this case, the application developer can use an object called *index*. This object can skip directly to a predetermined place or to another Web document. The visualization of this object plus some buttons for the navigation is shown in figure 5.2 for the generated lesson of chapter 4.



Group Communication

Group communication is the exchange of information between a group of participants in a session. Participants may have different roles in a session. Information is exchanged using certain media, chosen according to availability and suitability for the information to be transferred.

Different types of group communication sessions have evolved in the past in order to reach common goals as efficiently as possible. These goals include entertainment, decision making, learning, and plain information exchange about a common interest.

Figure 5.2 *Lessons browser.*

This browser allows the lecturer to go forward in a predetermined sequence or skip directly to an index sheet. A button allows hiding the index in order to expand the space of visualized text. Thus, the collaborative applications developer uses the objects defined in the platform as building blocks, together with other objects defined by him.

The platform is composed by a set of scripts (JavaScript), applets (Java) and CGIs (Perl and C). The programmer must combine all of them to create an application.

6. Conclusions and Further Work

In this paper, we presented an object-oriented platform to support the construction of collaborative multimedia applications for the teaching/learning process, running on top of a Web browser. As an example, we showed a simple lessons browser with dynamic behavior, i.e. the application will produce a lesson according to the user's needs.

Most parts of the platform are currently implemented, and some applications are already being used. However, the platform will be more stable only after having more experience with the construction and use of these kind of collaborative applications.

We are currently starting to experiment with many applications that we have built on top of the platform. One of them is a collaborative multimedia editor to support asynchronous and distributed course work of students. A second system is an application to support synchronous distributed discussions of the lessons, with the participation of the lecturer, teaching assistants and students. This application supports text conversations between participants, lessons using part of the Web browser, voting tools, and the transmission of audio and video whenever the bandwidth permits. Many different lessons browsers are also implemented, in order to experiment the best ways to present them.

For us, it is extremely important to experiment with the applications, measuring their impacts on the teaching/learning processes. This provides us with the required feedback to improve our hypotheses.

We still have to work on the understanding of our collaborative applications generator. For that, we included in the platform objects such as the box, canvas, and session. Specifically, it is our intention to provide an application on top of our platform that supports the construction of collaborative applications on top of the platform by using visual programming.

ACKNOLEDGMENTS

This work was partially supported by the Chilean Science and Technology Fund (FONDECYT), grant 195-0880. Special acknowledge is granted to Prof. Jennifer Widom and Mr. Jason McHugh at the Stanford University Database Group for allowing us to use their LORE system.

REFERENCES

[Abiteboul96] Abiteboul S., Quass D., McHugh J., Widom J. and Wiener J. *The Lorel Query Language for Semi-structured Data*. http://www-db.stanford.edu/pub/papers/ lorel96.ps, Department of Computer Science. Stanford University. California, USA, 1996.

[Abiteboul97] Abiteboul, R. Goldman, J. McHugh, V. Vassalos, Y. Zhuge, *Views for Semi-structured Data*, http://www-db.stanford.edu/pub/papers/oemview97.ps**,** Department of Computer Science. Stanford University. Stanford. California, USA. 1997.

[Berners-Lee95] Berners-Lee T. and Conolly D. *Hypertext Markup Language – 2.0*. Request for Comments 1866, November 1995. http://info.internet.isi.edu/in-notes/rfc/files/ rfc1866.txt

[Dobson95] Dobson S. and Burrill V. *Lightweight databases*. Computer Networks and ISDN Systems 27(6), pp. 1009-1015. April, 1995.

[Goldman96] Goldman R., Chawathe S., Crespo A. and McHugh J. *A Standard Textual Interchange Format for the Object Exchange Model (OEM)*. Department of Computer Science, Stanford University. California, USA, 1996.

[Hadjiefthymiades97] Hadjiefthymiades S. and Martakos D. *Improving the Performance of CGI Compliant Database Gateways*. Hyper Proceedings of the Sixth International World Wide Web Conference. Stanford, California, USA, 1997. http://www6.nttlabs.com/HyperNews/get/PAPER155.html

[Harlan96] Harlan D. et al. *Using Perl 5 for Web Programming*. QUE. USA, 1996.

[Ingham97] Ingham D., Caughey S. and Little M. *Supporting Highly Manageable Web Services*. Hyper Proceedings of the Sixth International World Wide Web Conference. Stanford, California, USA, 1997. http://www6.nttlabs.com/HyperNews/get/PAPER27.html

[Papakonstantinou94] Papakonstantinou Y., Garcia-Molina H, Widom J. *Object Exchange Across Heterogeneous Information Sources*. http://www-db.stanford.edu/pub/papers/ oem.ps. Department of Computer Science. Stanford University. California, USA, 1994.

[Rumbaugh91] Rumbaugh J. et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[Vitali97] Vitali F., Chiu C. and Bieber M. *Extending HTML in a Principled Way with Displets*. Hyper Proceedings of the Sixth International World Wide Web Conference. Stanford, California, USA, 1997. http://www6.nttlabs.com/HyperNews/get/PAPER44.html