An Architecture for Open Learning Management Systems

Avgeriou Paris[1], Retalis Simos[2], Skordalakis Manolis[1]

[1] National Technical University of Athens, Department of Electrical and Computer Engineering, Software Engineering Laboratory, Zografou, Athens, 15780, GREECE
{pavger, skordala}@softlab.ntua.gr,
[2] Department of Computer Science, University of Cyprus, 75 Kallipoleos St., P.O. Box 20537, CY-1678 Nicosia, CYPRUS,
retal@softlab.ntua.gr

Abstract. There exists an urgent demand on defining architectures for Learning Management Systems, so that high-level frameworks for understanding these systems can be discovered, and quality attributes like portability, interoperability, reusability and modifiability can be achieved. In this paper we propose a prototype architecture aimed to engineer Open Learning Management Systems, that professes state-of-the-art software engineering techniques such as layered structure and component-based nature. Our work is based upon standards and practices from international standardization bodies, on the empirical results of designing, developing and evaluating Learning Management Systems and on the practices of well-established software engineering techniques.

Keywords: software architecture, Learning Management Systems, Open Systems, component-based paradigm, quality attributes, Unified Modeling Language, Unified Process, Web-based Instructional Systems, Learning Technology Systems, Learning Technology Standards.

1 Introduction

Governments, authorities and organizations comprehend the potential of the Internet to transform the educational experience and envisage a knowledge-based future where acquiring and acting on knowledge is the primary operation of all life-long learners. In order to realize this vision, the use of Learning Technology Systems (LTS) is being exponentially augmented and broadened to cover all fields of the new economy demands. *Learning Technology Systems (LTS)* are learning, education and training systems that are supported by the Information Technology [1]. Examples of such systems are computer-based training systems, intelligent tutoring systems, Web-based Instructional Systems and so on.

Web-based Instructional Systems (WbISs) are LTSs that are based on the state-of-the-art Internet and WWW technologies in order to provide education and training following the open and distance learning paradigm. WbISs are comprised of three parts: *human resources* (students, professors, tutors, administrators etc.), *learning resources* (e-book, course notes etc.), and *technological infrastructure* (hardware, soft-

ware, networks). A major part of the technological infrastructure of WbISs is the *Learning Management System (LMS)*. LMSs are software systems that synthesize the functionality of computer-mediated communications software (e-mail, bulletin boards, newsgroups etc.) and on-line methods of delivering courseware (e.g. the WWW) [2]. An LMS is a middleware that acts and interfaces between the low-level infrastructure of the Internet and the WWW from the one side and the customized domain-specific learning education and training systems on the other side.

LMSs have been established as the basic infrastructure for supporting the technology-based, open and distance-learning process in an easy-to-use, pedagogically correct and cost-efficient manner. LMSs have been used for educational and training purposes, not only because they have been advertised as the state of the art learning technology, but also because they have substantial benefits to offer. In specific, they alleviate the constraints of time and place of learning; they grant multiple media delivery methods through hypermedia; they allow several synchronous and asynchronous communication facilities; they provide an excellent degree of flexibility concerning the way of learning; they support advanced interactivity between tutors and learners and they grant one-stop maintenance and reusability of resources [3, 4].

LMSs that are in use today are either commercial products (e.g. WebCT, Blackboard, Intralearn), or customized software systems that serve the instructional purposes of particular organizations. The design and development of LMSs though, is largely focused on satisfying certain *functional* requirements, such as the creation and distribution of on-line learning material, the communication and collaboration between the various actors, the management of institutional information systems and so on. On the contrary, the *quality* requirements of LMSs are usually overlooked and underestimated. This naturally results in inefficient systems of poor software, pedagogical and business quality. Problems that typically occur in these cases are: bad performance which is usually frustrating for the users; poor usability, that adds a cognitive overload to the user; increased cost for purchasing and maintaining the systems; poor customizability and modifiability; limited portability and reusability of learning resources and components; restricted interoperability between LMSs.

The question that arises is how can these deficiencies be remedied, how can the quality attributes be incorporated into the LMSs being engineered? Quality attributes in a software system depend profoundly on its architecture and are an immediate outcome of it [5, 6, 7, 8]. Therefore the support for qualities should be designed *into the architecture* of the system [7, 8, 9]. These principles have only recently been widely accepted and adopted and have lead to a research trend into defining software architectures that support quality attributes. Furthermore some of this effort is focused not only in developing but in standardizing software architectures LMSs, in order to provide a more systematic development process for these systems and achieve the aforementioned goals. At present there is an increasing interest in defining such architectures, from academic research teams (e.g. the Open Knowledge Initiative project http://web.mit.edu/oki/), from the corporate world (e.g. Sun Microsystems, see [9] and [10]), and from standardization bodies (e.g. the IEEE LTSC Learning Technology Systems Architecture, [http://ltsc.ieee.org/wg1/]). This paper describes a similar effort of defining a layered component-based architecture for LMSs and primarily aims at the incorporation of quality attributes into the LMS under construction. The ultimate goal is to build truly *Open* Learning Management Systems, that conform to the defini-

tion of *Open Systems* given by the Reference Model of Open Distributed Processing (RM-ODP) [12]: "Open systems are systems that are designed to enable portability of the software, and to allow other software entities to interoperate with it across dissimilar software and systems, whether or not the software entities are on the same computer or reside on different computers."

The structure of the paper is as follows: In section 2 we provide the theoretical background of the proposed architecture in terms of the context of LMSs, i.e. Web-based Instructional Systems and Learning Technology Systems. Section 3 deals with the description of the architecture per se. Section 4 contains conclusions about the added value of our approach and future plans.

2 Business Systems are supported by LMSs

As aforementioned we consider Learning Management Systems to be a part of one of the three components of Web-based Instructional Systems, and in particular the technological infrastructure. In order to comprehend the nature and characteristics of LMSs, we need to put things into perspective and take into account the context of LMSs, i.e. the WbIS and the LTS. Learning Technology Systems, and their specializations, like WbIS, can be considered as *business systems* that are supported by special *software systems*, like LMSs, which automate some of the business processes [13]. The reason for studying the generic category of LTSs is that there is a lot of work being done on the standardization of LTS architectures, and the development of LMSs can benefit from basing its foundations on such a strong and commonly accepted background.

We thus adopt a three-fold approach: we see LMSs as part of WbISs and the latter as children of LTSs, as illustrated in Figure 1. The profit of this approach is that the LTS refined into a WbIS can provide the business case for the LMS under development and can act as the business model in the architecture-centric approach of an LMS engineering process. This, in turn, provides the following advantages [7]:

1. The LMS become an integrated part of the overall business supporting the business and enhancing the work and the results.
2. The LMS and the business systems integrate easily with each other and can share and exchange information.
3. The LMS are easier to update and modify as dictated by changes in the business model. This in turn reduces the cost of maintaining the LMS and of continuously updating the business processes.
4. Business logic can be reused in several systems.

2.1 The LTS and WbIS business systems

The largest effort on developing an LTS architecture has been carried out in the IEEE P1484.1 Learning Technology Systems Architecture (LTSA) working group, which has developed a tentative and rather stable working standard. The LTSA describes a high-level system architecture and layering for learning technology systems, and identifies the objectives of human activities and computer processes and their involved

categories of knowledge. These are all encompassed into 5 layers, where each layer is a refinement of the concepts in the above layer.

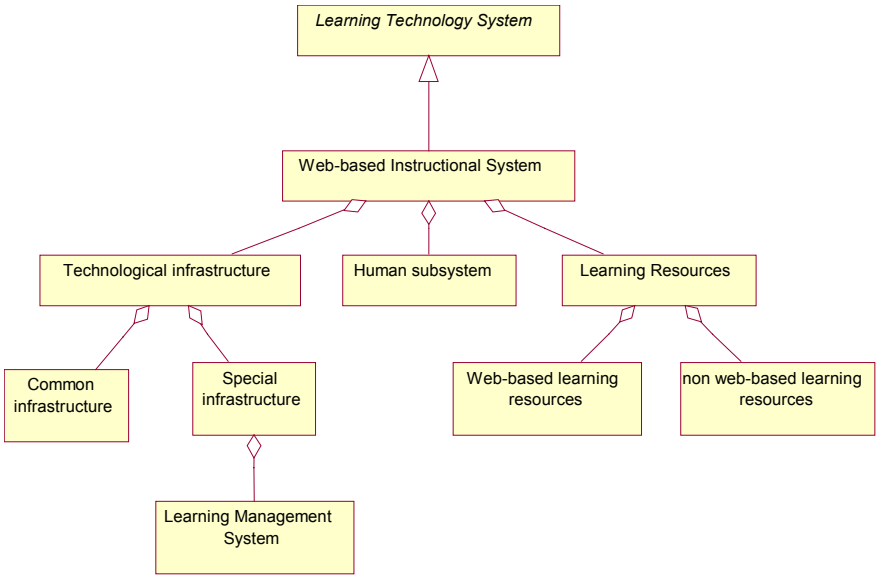

Fig. 1. The decomposition of a WbIS into components

Out of the five refinement layers of architecture specified in the LTSA, only layer 3 (system components) is normative in this Standard. Layer 1, *"Learner and Environment Interactions"* addresses the learner's acquisition, transfer, exchange, formulation, discovery, etc. of knowledge and/or information through interaction with the environment. Layer 2, *"Human-Centered and Pervasive Features"* addresses the human aspects of learning technology systems in terms of human-specific strengths and weaknesses. Layer 3, *"System Components"* describes the component-based architecture, as identified in human-centered and pervasive features. Layer 4, *"Stakeholder Perspectives and Priorities"* describes learning technology systems from a variety of perspectives by reference to subsets of the system components layer. Layer 5, *"Operational Components and Interoperability — codings, APIs, protocols"* describes the generic "plug-n-play" (interoperable) components and interfaces of an information technology-based learning technology architecture, as identified in the stakeholder perspectives. The added value derived from the abstraction-implementation layers, is that the five layers represent five independent areas of technical analysis, which makes it easier to discuss each layer independently of the others.

LTSs are applied in a plethora of domains for learning education and training purposes. A very popular domain of LTS application is web-based open and distance learning. There are currently no standards for architecting and building systems in this particular domain, so we will present a prototype architecture of *Web-based Instructional Systems (WbISs)* that has derived from experience on instructional design and

has been mostly influenced by the LTSA. According to this architecture, WbISs are comprised of:

- *The human subsystem*, which describes the roles, in as much detail as possible, for each kind of human agent involved in the instructional process [14]
- *The learning resources subsystem,* which is divided into web-based learning resources and non web-based learning resources. The former is perceived as a mosaic of online learning resources. Such learning resources can be course notes, slideware, study guides, self-assessment questionnaires, communication archives, learning material used for communication purposes, etc. The latter is comprised of digital or non-digital learning resources that are not deployed on the WWW like textbooks, papers, audio/video cassettes, CDs, DVDs, etc.
- *The technological infrastructure subsystem,* which is divided into common and special. An instructional system basically makes use of services from *common infrastructure*, which is a set of *learning places*, that support student learning in general (e.g. laboratories, networking facilities, etc.). However, in order to best support the instructional process, *special infrastructure* should be created (e.g. multimedia conferencing systems, state of the art hardware and software components etc.), which will provide services unique to a particular instructional problem. [13]. A most significant part of the special infrastructure is the LMS.

The decomposition of a WbIS using the UML notation is depicted in Figure 1 shown above.

2.2 Overview of LMS

Systems exist and have certain meaning and purpose within certain business contexts. Now that we have identified LTSs and WbISs, we can define LMSs, so that the latter will make sense in the bounds of the former.

A vast number of Learning Management Systems (e.g. WebCT, Blackboard, LearningSpace, Centra, TopClass) that provide integrated services, exist nowadays [2, 10]. Such systems offer different services and capabilities regarding organization and distribution of learning content, course management, student assessment, communication and collaboration tools, administration of instructional institutions and so forth. They offer different features and address different needs and concerns as far as pedagogy, open learning and instructional design are concerned. Consequently instructional designers that are called upon to solve a specific instructional problem with explicit needs and requirements must choose a specific LMS that fits closer to the above problem. In particular, the people involved in the decision-making process concerning instructional design and organization of educational institutions would use a Learning Management System in order to:

- Create, operate and administrate an on-line course.
- Support the collaboration between students and provide motivation and resources for team building [15].
- Create and deliver questions and tests for student assessment
- Organize educational, financial and human resources.
- Administer virtual, distributed classes where the students are geographically scattered and communicate via the Internet.

These diverse usage scenarios of LMS, correspond to different categories of Learning Technology Systems, which are respectively the following:

- **General Systems,** which have a number of tools for creating and managing courses and do not give emphasis to any particular set of features. We call these systems 'general' and not, for example 'Course Management', because they provide a plethora of features that span many assorted areas, in order to provide fully functional on-line courses, such as communication tools, administration tools, etc. These systems are also called **Learning Portals** and **Course Management Systems**.
- **Learning Content Management Systems**, which deal with creating, storing, assembling, managing and delivering hypermedia learning content. Often these systems provide metadata management tools so that learning material is accompanied by appropriate metadata [16].
- **Collaborative Learning Support Systems,** which emphasize on team building, student group management and providing the synchronous and asynchronous collaboration tools to support the aforementioned activities.
- **Question and Test Authoring and Management Systems,** which facilitate the design and authoring of quizzes and tests, which are published on the WWW and taken on-line. They provide tools for test creation and their on-line delivery, automatic grading, results manipulation and report generation.
- **People and Institute Resource Management Systems,** which deal with human resources and financial management. These systems are also called **Student Administration Systems**.
- **Virtual Classrooms,** which establish a virtual space for live interaction between all the participants in the learning process, i.e. instructors, tutors and students.

The LMS that can be classified in each one of the above categories support a number of **features**, or tools or capabilities in order to carry out certain tasks. These features do not discretely belong to only one LMS category but can be shared by several categories. These features can be classified into certain groups, namely [17]:

- **Course Management,** which contains features for the creation, customisation, administration and monitoring of courses.
- **Class Management,** which contains features for user management, team building, projects assignments etc.
- **Communication Tools,** which contains features for synchronous and asynchronous communication such as e-mail, chat, discussion fora, audio/video-conferencing, announcements and synchronous collaborative facilities (desktop, file and application sharing, whiteboard).
- **Student Tools,** which provide features to support students into managing and studying the learning resources, such as private & public annotations, highlights, bookmarks, off-line studying, log of personal history, search engines through metadata etc.
- **Content Management,** which provide features for content storing, authoring and delivery, file management, import and export of content chunks etc.
- **Assessment Tools,** which provides features for managing on-line quizzes and tests, project deliverables, self-assessment exercises, status of student participation in active learning and so on.

- **School-Management**, which provide features for managing records, absences, grades, student registrations, personal data of students, financial administration etc.

All these groups of features will be supported in the architecture, presented in the next section. This architecture is meant to be generic enough to embrace the different categories of LMS, and therefore it does not delve into specific details of single LMS.

3 The Architecture

The proposed architecture is a result of a prototype **architecting process** that is characterized of five important key aspects: it is founded on the higher-level architecture of IEEE P1484.1 Learning Technology Systems Architecture [http://ltsc.ieee.org/]; it uses a prototype architecture of a Web-based Instructional System [18] to build a complete business model and refine and constrain the requirements for the LMS; it adopts and customizes a big part of the well-established, software engineering process, the Rational Unified Process (RUP) [9, 19]; it uses the widely-adopted Unified Modeling Language [20, 21] to describe the architecture; and it is fundamentally and inherently component-based. The latter is justified by the fact that great emphasis has been put, not only in providing a pure component-based process, that generates solely components and connectors, but also in identifying the appropriate binding technologies for implementing and integrating the various components. Further study of the architecting process can be found at [22].

In order to describe the architecture for an LMS we need to base our work on a commonly accepted definition of the concept of software architecture. Unfortunately the software architecture community has not reached consensus on a common definition for the term of software architecture, given that the whole discipline is still considered very immature [7]. A rather broadly-used academic definition is the one given in [23]: "Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns". A similar definition from the IEEE Recommended Practice for Architectural Description of Software-Intensive Systems [24] is: "The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." We will adopt these definitions and attempt to refine them so as to *describe* the LMS architecture in terms of the RUP, which has been used as the basis for the architecting process. Therefore in compliance to the above definitions and according to [9, 19, 25], the *architectural description* should contain the following:

- The views (i.e. the most important or architecturally significant modeling elements) of the 5 models described in the RUP (use case model, analysis model, design model, deployment model, implementation model). This set of views corresponds with the classic "*4+1 views"* described in [26].
- The quality requirements that are desirable for the system and must be supported by the architecture. The requirements might or might not be described by use cases.
- A brief description of the platform, the legacy systems, the commercial software, the architecture patterns to be used.

For reasons of clarity and completeness, it is noted that the above list is not exhaustive, meaning that the RUP mentions other items that can also be included in the architectural description. On the other hand, the process is flexible enough to allow the architect to choose what he or she wants to take account of the particular system under development. For the purposes of this paper and for the final goal, i.e. the definition of the LMS architecture we will suffice to say that the above description is comprehensive enough.

3.1 The Architectural Description

The first and most sizeable part of the architectural description is the views of the 5 models dictated by the RUP. It is obvious that it is neither meaningful nor practical to illustrate even a small representative sample of the numerous diagrams produced in the 5 models. Instead, we will emphasize certain points, that will provide a minimum basis for demonstrating the LMS architecture, such as: a first level decomposition of the system; an exemplar second-level decomposition of one subsystem; how component interfaces are specified; platform and implementation decisions; the architectural patterns and the commercial software used.

The first-level decomposition of the Learning Management System is performed by specifying the very coarse-grained discrete subsystems in the design model, as they have derived from the use case and analysis model. It is noted that throughout the paper, the words 'component' and 'subsystem' are used interchangeably to denote pieces of the system that comply with the definition given in [6]: "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." The decomposition is combined with the enforcement of the "Layered Systems" architecture pattern [23, 27, 28], which helps organize the subsystems hierarchically into layers, in the sense that subsystems in one layer can only reference subsystems on the same level or below. The communication between subsystems that reside in different layers is achieved through clearly defined interfaces and the set of subsystems in each layer can be conceptualized as implementing a virtual machine [28]. The most widely known examples of this kind of architectural style are layered communication protocols such as the ISO/OSI, or operating systems such as some of the X Window System protocols.

The RUP utilizes the aforementioned architectural pattern by defining four layers in order to organize the subsystems in the design model. According to the RUP, a *layer* is a set of subsystems that share the same degree of generality and interface volatility. The four layers used to describe the architectural structure of a software system are [9]:

- *Application-specific:* A layer enclosing the subsystems that are application-specific and are not meant to be reused in different applications. This is the top layer, so its subsystems are not shared by other subsystems.
- *Application-general:* A layer comprised of the subsystems that are not specific to a single application but can be re-used for many different applications within the same domain or business.

- *Middleware:* A layer offering reusable building blocks (packages or subsystems) for utility frameworks and platform-independent services for things like distributed object computing and interoperability in heterogeneous environments, e.g. Object Request Brokers, platform-neutral frameworks for creating GUIs.
- *System software:* A layer containing the software for the computing and networking infrastructure, such as operating systems, DBMS, interface to specific hardware, e.g. TCP/IP.

The proposed layered architecture for an LMS is depicted in Figure 2, which is a first-level decomposition in the design model. This diagram, besides identifying all first-level subsystems and organizing them into layers, also defines dependencies between them, which are realized through well-specified interfaces. The list of subsystems contained in this diagram, although not exhaustive, highlights the most important of these subsystems.

The *application-specific* sub-systems of the layered architecture, which are the top-level components of the application, are:

1. Main subsystem (master component that initializes and launches everything else)
2. User management (registration in system, in courses and in groups, groups creation, authentication, access control with different views, student tracking, student profile management)
3. Courseware authoring (web page editing, design templates)
4. Courseware delivery (WWW server and client, delivery of hypermedia pages concerning e-book, glossary, index, calendar, course description etc., personalization per user)
5. Assessment (on-line quiz or exam, project deliverables, self-assessment exercises)
6. Searching (applies to all learning objects through metadata)
7. Course management (creation, customization, administration and monitoring of courses)
8. Study toolkit (private & public annotations, highlights, bookmarks, print out, off-line studying, notepad, log of personal history, adaptive navigation and presentation, intelligent tutoring systems)
9. System Administration (new course, back up, security, systems operation check, resource monitoring etc.)
10. School Administration (absences records, grades records, student registrations)
11. Help desk (on-line help, user support)

The *application-general* subsystems, which can be re-used in different applications, are:

1. Communication management (E-mail, Chat, Discussion fora, Audio/video-conferencing, Announcements, Synchronous collaborative facilities such as white-board, desktop, file and application sharing)
2. File management (FTP server and client)
3. Content packaging
4. Business objects management (connection with database, persistent object factory)
5. Metadata management
6. Raw data management
7. Database client
8. Calendar
9. Web Delivery (WWW client and server)

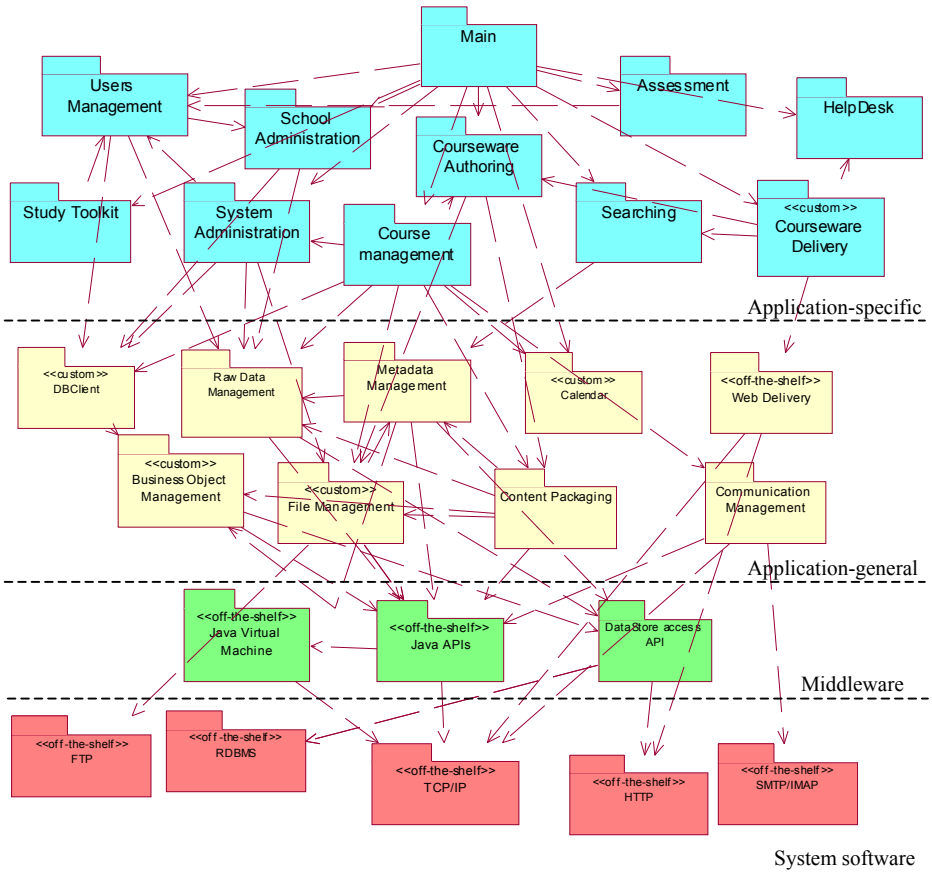

Fig. 2. The layered architecture of the component-based LMS

The *middleware* subsystems, which offer reusable building blocks for utility frameworks and platform-independent services, are:

1. Java Virtual Machine
2. Java APIs (RMI, JFC/Swing, JDBC, JMF etc.)
3. Data Store Access API (JDBC API, JDBC driver, DB access through RMI, Connection pooling)

The *system-software layer* subsystems, which contains the software for the computing and networking infrastructure, are the TCP/IP, HTTP, FTP, SMTP/IMAP protocols and an RDBMS.

These subsystems are further elaborated by identifying their contents, which are design classes, use-case realizations, interfaces and other design subsystems (recursively). For example the decomposition of the "Data Store Access API" into its design sub-systems is depicted in Figure 3. This specific subsystem is comprised of the JDBC API, which is the Java API for Open Database Connectivity, the JDBC driver

of the database used, a component that performs connection pooling to the database, and a component that offers access to the database through the Java Remote Method Invocation API. This decomposition must continue hierarchically until we reach the 'tree leaves', i.e. the design classes. It is noted that the relationships between the subsystems are UML *dependencies* and are simply meant to denote that one subsystem *uses* another.

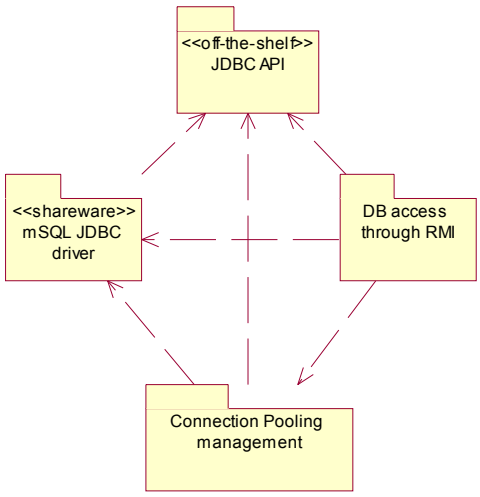

Fig. 3. The decomposition of the Data Store Access API into its design sub-systems

Furthermore, for each subsystem an interface must be specified so that the provided and the required operations are well defined. The *provided* operations are the ones that a specific subsystem offers to the other subsystems, the way a subsystem is used. The *required* operations state the functions that a subsystem expects from other subsystems, so that it can execute its functionality. A very simple example of an interface, with provided operations only, is depicted in Figure 4, where the aforementioned "Data Base access through RMI" subsystem's provided operations are shown. The SQLConnector class implements the RemoteSQLConnector interface, in order to connect to the database through RMI and perform SQL queries and updates, and handles ResultSetLite objects that contain the queries results. The signatures of the operations of the two classes and the interface are visible, and thus can be utilized in the design of other subsystems that interoperate with this one. As it will be shown later (Section 3.2), this formal form of interface specification is of paramount importance to the component nature of the architecture and yields significant advantages for the quality attributes of the system.

After the five views of the system have been completed, the core of the architecture is complete and is comprised of:

- the most significant functional requirements;
- nonfunctional requirements that are specific to architecturally significant use-cases;

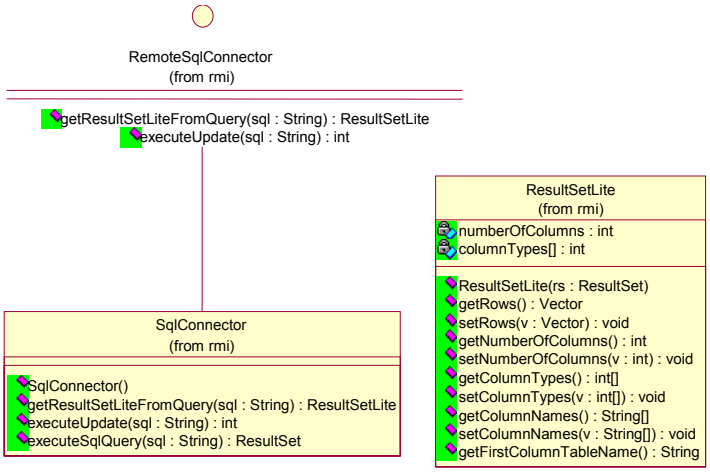

Fig. 4. The interface of the "DB Access through RMI" subsystem specified in UML

- the most important design classes and their organization into packages and subsystems, and the organization of these packages and subsystems into layers;
- some use case realizations;
- an overview of the implementation model and its organization in terms of components into packages and layers;
- a description of the tasks (process and threads) involved, their interactions and configurations, and the allocation of design objects and classes to tasks;
- the description of the various physical nodes and the allocation of tasks (from the Process View) to the physical nodes.

The next part of the component-based architecture concerns platform and implementation decisions, so that the architecture is completed, and the development team is assisted in implementing it into a physical system. In the architecture described in this paper, we propose certain implementation technologies and platforms that we consider to be the most suitable for a component-based system. These technologies implement the component-based paradigm using object-oriented techniques, specifically the Unified Modeling Language, and the Java, C++ and VBA programming languages. The application of these technologies results in components implemented as JavaBeans or Microsoft Component Objects. The component development process, comprised of such technologies, is depicted in Figure 5.

The artifacts from the design model, that is sub-systems with UML-defined interfaces are provided as an input to this model. The next step is to transform the UML interfaces into the implementation platform, in our case either Java or Microsoft technologies. This forward engineering process can be easily automated with CASE tools such as Rational Rose [http://www.rational.com/rose] or Control Center [http://www.togethersoft.com/products/controlcenter/index.jsp], that generate abstract code from UML models. It is noted that we have included both Java and Microsoft, as alternative implementation platforms, for reasons of completeness, since they are the state-of-the-art component technologies. It is up to the development team to make the

choice between them. For the architectural prototype presented in the next sub-section we have chosen the Java platform.

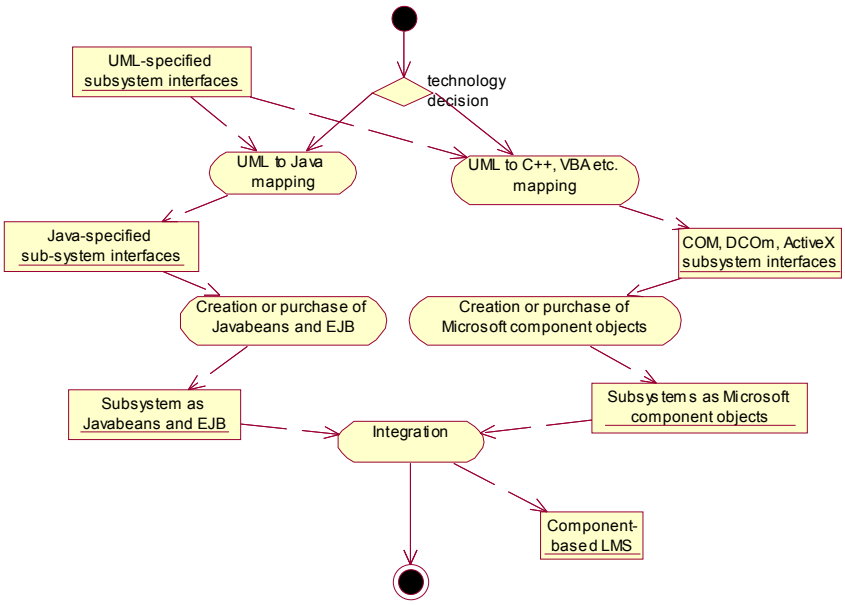

Fig. 5. Component development process

After the component interfaces are concretely defined in the programming language, they can either be constructed from scratch, or acquired from existing implementations and possibly modified to exactly fit their interfaces. The result is the implementation of the sub-systems as JavaBeans or Enterprise JavaBeans (EJB), which is the Java form of components, or, as Microsoft component objects (COM/DCOM objects, ActiveX controls etc.). The possible re-use of components is one of the areas where the component-based approach thrives. The final step is to integrate the components through an integration and testing process into the final outcome, i.e. the LMS.

Additional issues related to the architecture description, such as the legacy systems, the commercial software, the architectural patterns to be used etc. are also quite important and are outlined as following. In the proposed architecture there are no legacy systems since, the whole prototype system is being developed from scratch. As far as the commercial systems, we have adopted several of them such as the mySQL RDBMS [http://www.mysql.com] and the Resin Web Server and Servlets engine [http://www.caucho.com], the Sun 1.3.1 Java Run Time Environment, as well as some outsourced java packages such as mySQL JDBC driver, the Java Media Framework API, etc. The architectural patterns that have been used, as seen in the catalogue com-

posed in [23, 27, 28] include: the *layered* style as aforementioned; the *Client-Server* style has been used extensively, especially in the communication management components; the *Model-View-Controller* style in the GUI design, which is inherent in all Java Swing UI components; the *blackboard* style in the mechanisms that access the database in various ways; the *Virtual Machine* and the *object-oriented* style which are both a result of the implementation in Java; the *event systems* style for notification of GUI components about the change of state of persistent objects.

The final concerns that need to be addressed in this architectural description are the desirable qualities of the architecture, also known as nonfunctional requirements. Software architectures can be evaluated according to specific criteria and are designed to fulfill certain quality attributes [5, 8, 28]. It is noted that no quality can be maximized in a system without sacrificing some other quality or qualities, instead there is always a trade-off while choosing on supporting the different quality attributes [5, 8, 28]. We have decided to evaluate the architecture using two techniques: by evaluating the architectural prototype, and by informally assessing the architecture itself using our architectural experience combined and supporting it with the appropriate line of reasoning. Even though the evaluation results are out of the scope of this paper, we will only mention here the quality criteria that we have adopted from [28] and used in our research: performance, security, availability, functionality, usability, modifiability, portability, integrability, interoperability, reusability, testability, time to market, cost, projected lifetime of the system and targeted market.

3.2 The Architectural Prototype

An architecture is a visual, holistic view of the system, but it is only an abstraction. In order to evaluate the architecture in terms of the quality attributes it promotes, we must build it. Therefore, the software architecture must be accompanied with an **architectural prototype** that implements the most important design decisions sufficiently to validate them - that is to test and measure them [5, 8, 19]. The architectural prototype is the most important artifact associated with the architecture itself, which illustrates the architectural decisions and help us evolve and stabilize the architecture.
In order to assess and validate the proposed architecture, a prototype was engineered that implements the main architectural elements. The prototype LMS is named "Athena" and Figure 6 depicts some of its tools in action. There was the option of choosing a platform, Java or Microsoft-based as already shown in Figure 5. Our choice was the Java platform because it is an open technology, rather than proprietary, and based on a Virtual Machine, thus promoting portability. The specific technologies used are applets, servlets, Java Beans, Enterprise Java Beans, Java Server Pages, as well as the JFC/Swing, RMI, JDBC, 2D Graphics, JMF and JAF Java APIs. The eXtensible Markup Language (XML) was used as the default language for the representation of data that were not stored in the database.

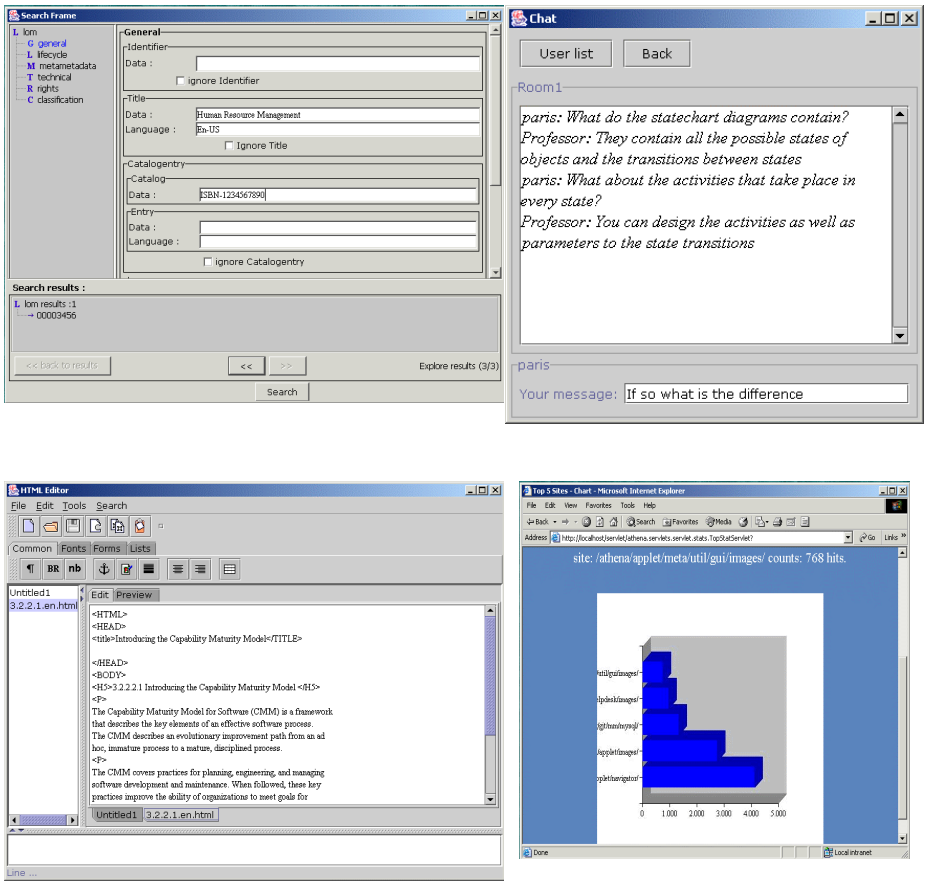

Fig. 6. Screenshots of the architectural prototype

The decision on implementing the prototype was to emphasize on the implementation of the majority of the components, and not on providing the full functionality of them. About 75% of the total number of components have been implemented or acquired and put into operation, even though some of them do not offer the complete functionality prescribed in the system design. More specifically and with reference to the layered architecture illustrated above, the components that were acquired from third parties are: WWW Server and servlet engine (Resin), WWW browser (Internet Explorer), FTP Server, Mail Server, RDBMS (mySQL), Audio/Video Conferencing client (MS Netmeeting), Sun Java Run Time Environment. In addition the components that were implemented from scratch are: Main Subsystem, User Management, Courseware Authoring, Course Management, Searching, Assessment, Help Desk, Data Base client, Raw Data Management, Business Objects Management, FTP client, Metadata Management, Calendar, Communication System (E-mail client, Chat server

and client, Whiteboard server and client, Announcements tool), Data Store Access API.

Finally there was an attempt on adopting international standards within the various components in order to promote interoperability of LMSs and portability of the learning resources. For that purpose we have developed the metadata management component conforming to the IEEE LTSC Learning Object Metadata working standard [16]. We have also implemented the assessment component in order to adopt the IMS Question and Testing Interoperability Standard [29]. Unfortunately most of these standards have not finalized just yet, but the aim of adopting them at such an early stage was to explore the feasibility of implementing them into our components. Furthermore, as it will be shown later, the system has been designed with the quality of modifiability in mind, and therefore changes in future versions of the standards should be easy to incorporate.

4 Conclusions and Future Work

We have portrayed a layered component-based architecture for an LMS, which uses the IEEE P1484.1 LTSA and a prototype WbIS architecture as a business model, adopts the architecting practices of the Unified Software Development Process and grants special emphasis on enforcing a component-based nature in it. Each one of these key concepts adds special value to the proposed architecture.

It has been strongly supported that an architecture-centric development process professes numerous advantages [5, 9, 20]. In general, the purpose of developing software architecture is to discover high-level frameworks for understanding certain kinds of systems, their subsystems, and their interactions with related systems. In other words, an architecture isn't a blueprint for designing a single system, but a framework for designing a range of systems over time, thus achieving adaptability, and for the analysis and comparison of these systems [1]. Furthermore, an all-important necessity for an LMS is interoperability and portability, which is a fundamental feature of component-based architectures and is achieved by identifying critical component interfaces in the system ‘s architecture. Portability of components also leads to reusability, a keyword in the development of affordable systems. Component-based software architectures promote reuse not only at the implementation level, but at the design level as well, thus saving time and effort of ‘re-inventing the wheel’. Moreover, architecture-based development offers significant Software Engineering advantages such as: risk mitigation, understanding of the system through a common language, effective organization of the development effort, and making change-tolerant systems. Finally the utilization of the ‘Layered Systems’ architectural pattern further promotes modifiability, portability, reusability and good component-based design as it allows the partition of a complex problem into a sequence of incremental steps [9, 23, 28]. Based on these points, it is concluded that an inherently layered component-based software architecture is the right step towards bringing the economies of scale, needed to build Open Learning Management Systems: LMS that can interoperate and exchange learning material, student data, course information; LMS that can be ported to any plat-

form, independently of operating system and hardware configuration; LMSs that give their designers the ability to remove and insert plug-and-play components at will.

We are currently examining several issues in order to extend and elaborate on the work presented in this paper. First of all we are investigating on the way, that a learning theory can be combined with the business model in order to provide a full set of system requirements. Another issue that is being currently examined is the development of an Architecture Description Language (ADL) that will be customized to describe software architectures especially for the domain of LMSs, and will be based on extensions of the UML in combination with existing ADLs and development methods [30, 31, 32].

Moreover, the new research steps will be towards the use of design patterns that will complement the proposed WbIS model. Design patterns are a good means for recording design experience as they systematically name, explain and evaluate important and recurrent designs in software systems [33]. They describe problems that occur repeatedly, and describe the core of the solution to these problems, in such a way that we can use this solution many times in different contexts and applications. Looking at known uses of a particular pattern, we can see how successful designers solve recurrent problems. In some cases, it is possible to give structure to simple patterns to develop a pattern language: a partially ordered set of related patterns that work together in the context of certain application domain. This work will be in line with the research efforts that are being performed by [34, 35, 35].

References

1. IEEE Learning Technology Standards Committee (LTSC), Draft Standard for Learning Technology Systems Architecture (LTSA), Draft 9, November 2001, http://ltsc.ieee.org/.
2. Oleg, S., Liber, B.: A framework of pedagogical evaluation of Virtual Learning Environments. Available online at [http://www.jtap.ac.uk/reports/htm/jtap-041.html], 1999.
3. McCormack, C., Jones, J.D.: Building a Web-based Education System. Wiley Computer Publishing, 1997.
4. Lowe, D., Hall, W.: Hypermedia & the Web: An Engineering Approach. John Wiley Ltd., 1999.
5. Bosch, J.: Design and Use of Software Architectures. Addison-Wesley, 2000.
6. Szyperski, C.: Component Software – Beyond Object-Oriented Programming. ACM Press, 1999.
7. Eriksson, H. and Penker, M., 2000. Business Modeling with UML - Business Patterns at work, John Wiley & Sons.
8. P. Clements, R. Kazman, M. Clein, Evaluating Software Architecture, Addison-Wesley, 2002.
9. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, 1999.
10. Sun Microsystems, Understanding Distance Learning Architectures, A White Paper, 1999.
11. G. Collier, "Elearning application Infrastructure", Sun Microsystems white paper, http://www.sun.com/products-n-solutions/edu/whitepapers/index.html, January 2002.
12. ISO/IEC 10746-3: 1996, Information technology. Open distributed processing. Reference model: Architecture.
13. Ford, P., Goodyear, P., Heseltine, R., Lewis, R., Darby, J., Graves, J., Sartorius, P., Harwood, D., King, T.: Managing Change in Higher Education: A Learning Environment Ar-

chitecture. London: Open University Press, 1996.
14. Lindner, R.: Proposals for an Architecture WG and new NPs for this WG - Expertise and Role Identification for Learning Environments (ERILE). available online at [http://jtc1sc36.org/], 2001.
15. McConnell, D. (1994). *Implementing computer-supported cooperative learning*, London: Kogan Page.
16. IEEE Learning Technology Standards Committee, (LTSC), Draft Standard for Learning Object Metadata (LOM), Draft 6.4, 2001, http://ltsc.ieee.org.
17. Avgeriou, P., Papasalouros, A., Retalis, S.: Web-based learning Environments: issues, trends, challenges. Proceedings of the 1st IOSTE symposium in Southern Europe, Science and Technology Education, Paralimni, Cyprus, May 2001.
18. S. Retalis and P. Avgeriou, 2002. "Modeling Web-based Instructional Systems", Journal of Information Technology Education, Volume 1, No. 1, pp. 25-41.
19. Kruchten, P., 1999. The Rational Unified Process, An introduction, Addison-Wesley.
20. Booch, G., Rumbaugh, J., Jacobson, I.: The UML User Guide. Addison-Wesley, 1999.
21. Rumbaugh, J., Jacobson, I. and Booch, G., 1999. The UML Reference Manual, Addison-Wesley.
22. Avgeriou, P., Retalis, S., Papasalouros, A., Skordalakis, M., 2001. Developing an architecture for the Software Subsystem of a Learning Technology System – an Engineering approach, Proceedings of International Conference of Advanced Learning Technologies, 6-8 August 2001, Madison, Wisconsin, IEEE Computer Society Press, pp. 17-20.
23. Shaw, M., Garlan, D.: Software Architecture - Perspectives on an emerging discipline. Prentice Hall, 1996.
24. IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE std. 1471-2000).
25. The Rational Unified Process, 2000, v. 2001.03.00.23, Rational Software Corporation, part of the Rational Solutions for Windows suite.
26. Kruchten, P., 1995. The 4+1 view model of architecture, IEEE Software, 12(6), pp. 42-50.
27. Buschmann, F., Meunier, R., Rohnert, H., Sommertland, P. and Stal, M., 1996. Pattern-Oriented Software Architecture, Volume 1: A System of Patterns, John Wiley & Sons.
28. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 1998.
29. IMS Global Learning Consortium, 2001. IMS Question & Test Interoperability Specification- Best Practice and Implementation Guide, version 1.2.1, http://www.imsproject.org/.
30. Robbins, J.E., Medvidovic, N., Redmiles, D.F., Rosenblum, D.S.: Integrating architecture description languages with a standard design method. Proceedings of the 1998 International Conference on Software Engineering, 1998.
31. Medvidovic, N.; Taylor, R.N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering, vol.26, (no.1), IEEE, Jan. 2000. p.70-93.
32. Medvidovic, N.; Rosenblum, D.; Redmiles, D.; and Robbins, J.: Modelling Software Architectures in the Unified Modeling Language, ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 1, January 2002, pages 2-57.
33. Gamma, R. Helm, R. Johnson & Vlissides, J. (1995). *Design Patterns: elements of reusable object-oriented software*, Addison Wesley.
34. Rossi, G., Schwabe, D. & Lyardet, F. (1999). Improving Web information systems with Navigational patterns, *International Journal of Computer Networks and Applications*, May 1999.
35. Garzotto, F., Paolini P., Bolchini D., & Valenti S. (1999). Modeling-by-patterns of web applications, *Lecture Notes in Computer Science*, 1727, Springer.
36. Lyardet, F., Rossi G., & Schwabe, D. (1998). Patterns for Dynamic Websites, Proceedings of PloP'98, Allerton, USA.