university of groningen

University Medical Center Groningen

University of Groningen

A Layered Component-Based Architecture of a Virtual Learning Environment

Avgeriou, Paris; Retalis, Simos; Skordalakis, Manolis; Psaromiligos, Yiannis

Published in:
EPRINTS-BOOK-TITLE

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2001

Link to publication in University of Groningen/UMCG research database

Citation for published version (APA):
Avgeriou, P., Retalis, S., Skordalakis, M., & Psaromiligos, Y. (2001). A Layered Component-Based Architecture of a Virtual Learning Environment. In *EPRINTS-BOOK-TITLE* University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science.

Copyright
Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license. More information can be found on the University of Groningen website: https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment.

Take-down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): http://www.rug.nl/research/portal. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Download date: 11-10-2022

A Layered Component-Based Architecture of a Virtual Learning Environment

Avgeriou Paris[1] Retalis Simos[2] Skordalakis Manolis[1] Psaromiligos Yiannis[3]

[1] National Technical University of Athens, Department of Electrical and Computer Engineering
Software Engineering Laboratory, Zografou, Athens, 15780, GREECE
E-mail: {pavger, skordala}@softlab.ntua.gr,
[2] Department of Computer Science, University of Cyprus, 75 Kallipoleos St., P.O. Box 20537
CY-1678 Nicosia, CYPRUS, E-mail: retal@softlab.ntua.gr
[3] Technological Education Institute of Piraeus, General Department of Mathematics
Computer Science Laboratory, P. Ralli 250, Athens, 12244, GREECE, E-mail: jpsa@teipir.gr

Abstract. There exists an urgent demand on defining architectures for Virtual Learning Environments (VLEs), so that high-level frameworks for understanding these systems can be discovered, portability, interoperability and reusability can be achieved and adaptability over time can be accomplished. In this paper we propose a prototype architecture for a VLE that professes state-of-the-art software engineering techniques such as layered structure and component-based nature. We base our work upon the LTSA working standard of IEEE LTSC, which serves as a business model, on the empirical results of a web-based instructional system architecture and on the practices of a well-established software engineering process.

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, software, networks). A major part of the technological infrastructure of WbISs is the *Virtual Learning Environment (VLE)*. VLEs are learning management 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]. A VLE 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.

The VLE has 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. VLEs 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 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].

However, it is common knowledge among researchers that VLEs are facing numerous shortcomings, mainly concerning the portability and reusability of learning resources as well as the interoperability between VLEs themselves. Many technology experts are working together to launch a set of methods and standards that will enable re-use, recombination and transfer of content between individuals, institutions and countries [5]. Much of this effort is focused on developing and standardizing system architectures for LTSs in general or LTS components such as VLEs, in order to provide a more systematic development process for these systems and achieve the aforementioned goals. This paper describes a similar effort of defining a layered component-based architecture for VLEs and primarily aims at the direction of discovering a high-level framework for understanding VLEs as systems, their subsystems, and their interactions with related systems.

This paper presents an approach of a layered component-based architecture for a VLE in the context of WbISs and LTSs. The VLE is seen as a software component of a WbIS, which is in turn seen as a special kind of LTS. The reason for defining the context of the VLE is to establish a business case for the construction of the VLE and thus found the architecting process on a sound basis. 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 VLEs, 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. The Context of VLEs

As we illustrated in Section 1, we consider Virtual Learning Environments 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 VLEs, we need to put things into perspectives and take into account the context of VLEs, i.e. the whole of the WbIS. Furthermore, since WbISs are a kind of Learning Technology Systems, they inherit many of the LTS features and they can be ultimately treated as such. 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 VLEs can benefit from basing its foundations on such a strong and commonly accepted background. We thus adopt a three-fold approach: we see VLEs as part of WbISs and the latter as children of LTSs. In other words VLEs are associated with WbISs with an aggregation relationship (part_of) and WbISs in turn are associated with LTSs with a generalization relationship (inheritance). The profit of this approach is that the LTS refined into a WbIS can provide the business case for the VLE under development and can act as the business model in the architecture-centric approach of a VLE engineering process.

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.

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 [6]
- *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. [7]. A most significant part of the special infrastructure is the *Virtual Learning Environment (VLE).*

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

Systems exist and have certain meaning and purpose within certain business contexts. Now that we have identified LTSs and WbISs, we can define VLEs, so that the latter will make sense in the bounds of the former. In consequence VLEs in the business context of WbISs and LTSs, support a number of *features*, or tools or capabilities in order to carry out certain

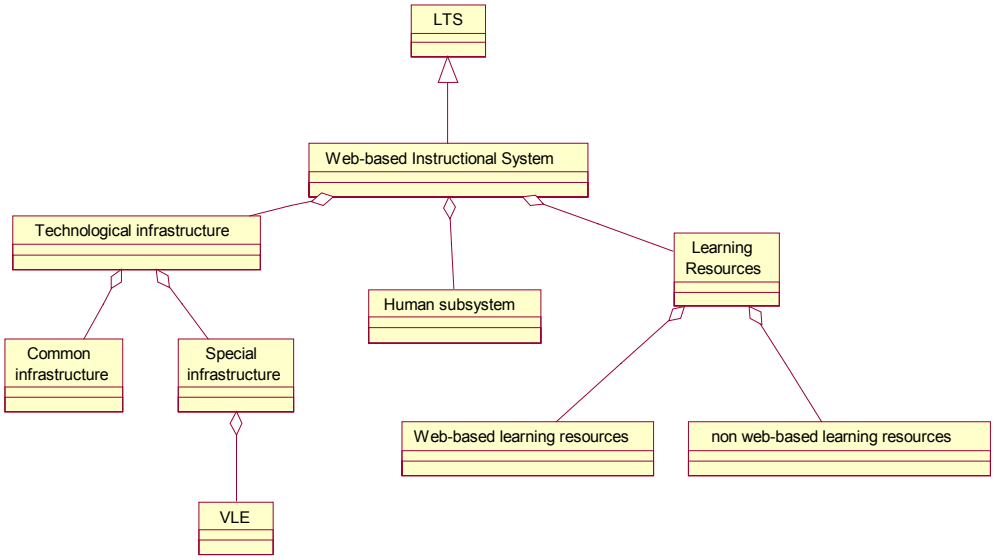

Fig. 1. The decomposition of a WbIS into components

tasks. These features can be classified into certain groups, namely [8]:

- *Course Management***,** which contains features for the creation, customization, 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 etc.
- *Content Management,* which provide features for content authoring and delivery and file management.
- *Assessment Tools*, which provides features for managing on-line quizzes and tests, project deliverables, self-assessment exercises and so on.
- *School-Management,* which provide features for managing records, absences, grades, student registrations, financial administration etc.

Now that all three kinds of systems and their relationships have been presented, i.e. VLEs, WbISs and LTSs, we move on to describe the VLE layered component-based architecture.

3. The Architecture

The proposed architecture is a result of a prototype architecting process that is characterized of four 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 WbIS architecture to refine and constrain the requirements for the VLE; it adopts and customizes a big part of the well-established, widely-adopted, industry-leading software engineering

process, the Unified Software Development Process (USDP) [9]; 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 component frameworks, but also in identifying the appropriate binding technologies for implementing and integrating the various components. Further analysis of the architecting process can be found at [10]. The notation used to describe the architecture is the Unified Modeling Language [11], a widely adopted modeling language in the software industry and an Object Management Group standard [http://www.omg.org].

Our aim in generating the VLE architecture is to produce an inherently component-based architecture with the help of the USDP. How can that be achieved? As stated in [12], a software system architecture in the component-based paradigm consists of a set of component frameworks, an interoperation design for the component frameworks, and a set of platform decisions. This statement corresponds with the architecture description given in the USDP, where the architectural views of the five models correspond with the component frameworks and the interoperation design between them, from five different viewpoints, while platform decisions are matched with the rest of the architecture description dictated by the USDP. This correlation is depicted in Table 1. We shall follow this pattern in order to describe the component-based nature in the proposed architecture. We shall first analyze the system into component frameworks or as we simply call them *subsystems*, describe their interaction and lastly make platform decisions.

USDP architecture	**Component based architecture**
Use case model view Analysis model view Design model view Deployment model view Implementation model view	Set of component frameworks, and the interoperation design for the component frameworks
Requirements that are architecturally significant but are not described by use cases. A brief description of the platform, the legacy systems, the commercial software.	Platform decisions

Table 1. Correspondence between a component-based architecture and the USDP architecture

The first-level decomposition of the Virtual Learning Environment 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. This decomposition is combined with the enforcement of the "Layered Systems" architecture pattern [13, 14], 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 [14]. 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 Unified Software Development Process utilizes the aforementioned architectural pattern by defining four layers in order to organize the subsystems in the design model. According to the USDP, a *layer* is a set of subsystems that share the same degree of generality and

interface volatility and 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 GUI.
- *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 a VLE 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, it also defines dependencies between them, which are realized through well-specified interfaces. The list of sub-systems 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. User profile management (for teachers, tutors, administrators this mainly includes personal data; for students it also includes grades, statistics of navigation, working groups and projects they are assigned to etc.)
2. Hypermedia authoring (web page editing, design templates)
3. Hypermedia delivery (delivery of hypermedia pages concerning e-book, glossary, index, calendar, course description etc., personalization per user)
4. Assessment (on-line quiz or exam, project deliverables, self-assessment exercises)
5. Searching which applies to all learning objects through metadata
6. Course management (creation, customization, administration and monitoring of courses)
7. User management (registration in courses, authentication, rights, views, student tracking)
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 whiteboard, desktop, file and application sharing)
2. File management
3. Content management (content packaging)
4. Business objects management (connection with database, persistent object factory)
5. Metadata management

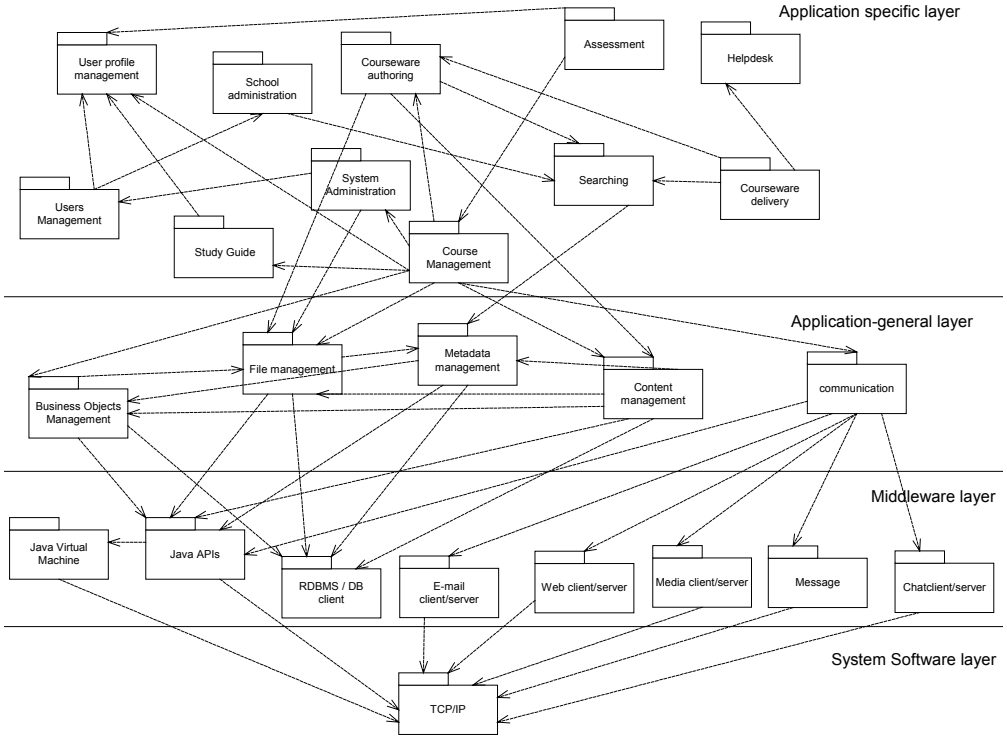

Fig. 2. The layered architecture of the component-based VLE

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

1. JVM
2. E-mail client/server
3. Java APIs (RMI, JFC/Swing, JDBC, JMF etc.)
4. Web server/browser
5. Media server/client
6. Message server/client
7. RDBMS/ DB client
8. Chat server/client

The *system-software layer* subsystem, which contains the software for the computing and networking infrastructure, is:

1. TCP/IP

These subsystems, that are in essence component frameworks, are meant to be further processed by identifying their contents, that is design classes, use-case realizations, interfaces and other design subsystems (recursively). Furthermore, each subsystem must provide an interface of its own, in order to represent its own functionality. All the above constitute the interoperation design of the subsystems, i.e. is the rules of interoperation among all the frameworks joined by the system architecture. We will not analyze these interoperations further for reasons of lack of space and we will only suffice to show the dependency relationships between the subsystems in Figure 2.

After the five models of the USDP have been completed, all the component frameworks and interoperations between them have been identified. The last 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 binding 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, the Interface Definition Language, and the Java, C++ and VBA programming languages. The application of these technologies results in components implemented as JavaBeans and Microsoft Component Objects. The component development process comprised of such technologies is depicted in Figure 3. The artifacts from the design model, that is sub-systems with textually described interfaces are provided as an input to this model. These interfaces are then designed with concrete UML notation and then mapped into the Interface Definition Language (IDL), which is an ISO standard for formally defining interfaces. Because the UML to IDL mapping is incomplete, the produced IDL interfaces need to be enhanced, so that a contractual specification can be achieved. The next step is to transform the IDL interfaces into the implementation platform, in our case Java or Microsoft technologies, through the Java IDL API, or the Microsoft IDL APIs. The components now are concretely defined in the programming language, and they can either be constructed from scratch, or acquired from existing implementations and possibly modified to exactly fit the 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 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 VLE.

In order to achieve interoperability and portability between different VLEs, the establishment of a component-based architecture featuring the component development technologies proposed above is necessary but not sufficient. An even more significant issue that needs to be taken under account is the adoption of standards for the development of each component. For example the metadata management component can be developed to conform to the IEEE LTSC Learning Object Metadata working standard, or the assessment component may adopt the IMS QTI standard [http://www.imsproject.org/]. Unfortunately most of these standards have not finalized just yet.

4. Conclusions and Future Work

We have portrayed a layered component-based architecture for a VLE, 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 [9, 11, 15]. 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 a VLE is interoperability and portability, which is a fundamental feature of component-based architectures and is achieved by

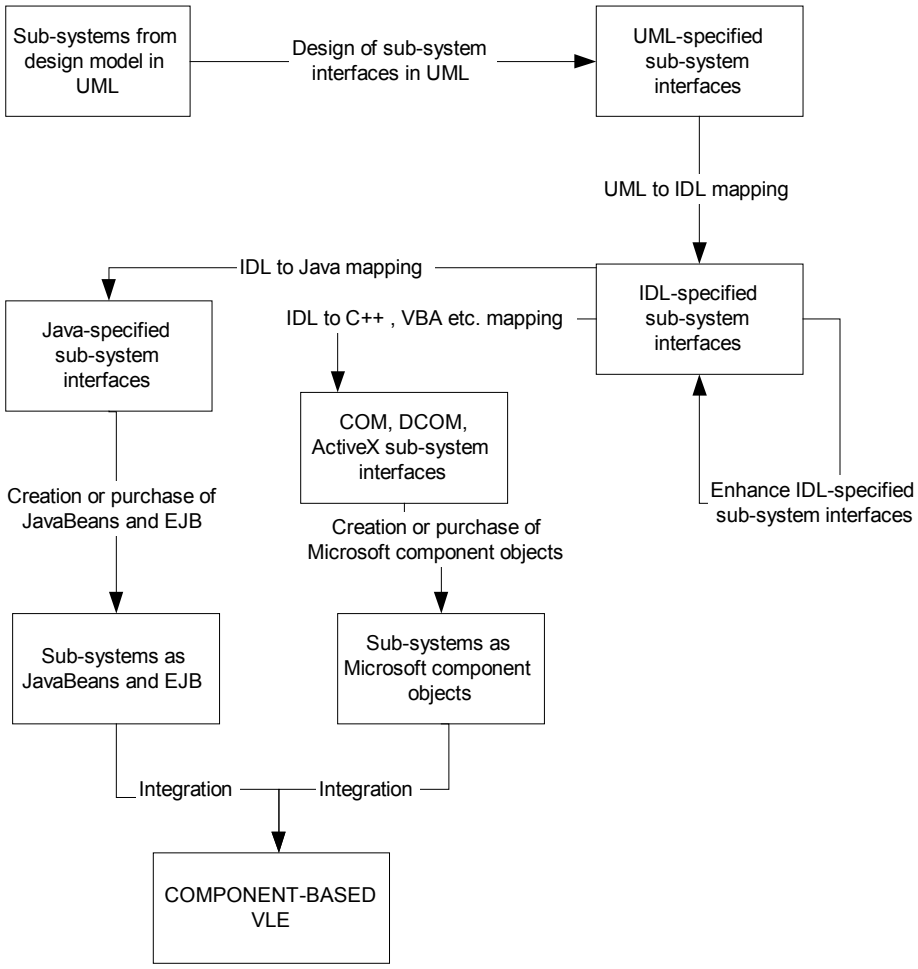

Fig. 3. Component development process

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, 13, 14]. 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 affordable, interoperable as well as effective Virtual Learning Environments.

We are currently investigating the implementation of the proposed architecture into a prototype VLE by enforcing the whole of the USDP. This will raise several issues such as: whether the LTSA and the prototype WbIS architecture are able to provide a full, well-documented business model; how can a learning theory be combined with the business model in order to provide a full set of system requirements; whether the USDP, which is a generic software engineering process, works well in this type of applications; whether the binding technologies and platforms proposed, will efficiently help in the software system implementation. A final 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 VLEs, and will be based on extensions of the UML in combination with existing ADLs and development methods [16, 17].

References

1. IEEE Learning Technology Standards Committee: Draft Standard for Learning Technology Systems Architecture (LTSA). Draft 8, April 2001.
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. Hodgins, W.: Into the future, A vision Paper. Commission on Technology & Adult Learning. February 2000, available online at [http://www.learnativity.com/into_the_future2000.html].
6. 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.
7. 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 Architecture. London: Open University Press, 1996.
8. 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.
9. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, 1999.
10. Avgeriou, P., Retalis, S., Papasalouros, A., Skordalakis, M.: Developing an architecture for the Software Subsystem of a Learning Technology System – an Engineering approach. Proceedings of the International Conference of Advanced Learning Technologies, Madison, Wisconsin, USA, 6-8 August 2001.
11. Booch, G., Rumbaugh, J., Jacobson, I.: The UML User Guide. Addison-Wesley, 1999.
12. Szyperski, C.: Component Software – Beyond Object-Oriented Programming. ACM Press, 1999.
13. Shaw, M., Garlan, D.: SOFTWARE ARCHITECTURE – Perspectives on an emerging discipline. Prentice Hall, 1996.
14. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, 1998.
15. Bosch, J.: Design and Use of Software Architectures. Addison-Wesley, 2000.
16. 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.
17. 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.