Developing an Architecture for the Software Subsystem of a Learning Technology System – an Engineering Approach

Avgeriou Paris
National Technical University of Athens
Dept. of Electrical and Computer Engineering
pavger@softlab.ntua.gr

Retalis Simos
Department of Computer Science
University of Cyprus
retal@softlab.ntua.gr

Papasalouros Andreas
Skordalakis Manolis
National Technical University of Athens
Dept. of Electrical and Computer Engineering
{andpapas, skordala}@softlab.ntua.gr

Abstract

There exists an urgent demand on defining architectures for Learning Technology Systems (LTS), 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 an architecting process for only the software subsystem of an LTS. We base our work upon the LTSA working standard of IEEE LTSC, which serves as a business model and on the practices of a well-established software engineering process. Special emphasis is granted on imposing a component-based nature on the produced architecture.

1. Introduction

Learning Technology Systems (LTS) are learning, education and training systems that are supported by the Information Technology. Examples of such systems are computer-based training systems, intelligent tutoring systems, web-based distance learning systems and so on.

It is common knowledge that the application of Learning Technologies does not comprise a panacea to the problem of accomplishing knowledge-driven education and training and performing the "educational shift" from teacher to learner-centered [1]. Even though LTS are quite promising in aiding to the accomplishment of this cause, undoubtedly a vast amount of research needs to be conducted in order to move from promise to practice [2]. Much of this research effort is focused on developing system architectures for LTS.

In this paper we profess the numerous advantages of introducing a component-based architecture for the software subsystem of LTS, seen from a software engineering point of view. The added value of our work is the proposal of a component-based architecting process for the software part of Learning Technology Systems, i.e. a software architecting process. This process has three important key aspects: it is founded on the higher-level architecture of IEEE P1484.1 Learning Technology Systems Architecture [http://ltsc.ieee.org/]; it adopts and customizes a big part of the well-established, widely-adopted, industry-leading software engineering process, the Unified Software Development Process (USDP) [3]; and it is fundamentally and inherently component-based.

The structure of the paper is as follows: In section 2 we provide the theoretical background of the process, which derives both from the software engineering discipline and the LTS standardization efforts. Section 3 deals with the description of the process itself, focusing also on the fact that it receives input from LTS working standards and that special care is taken to produce an inherently component-based architecture. Section 4 contains conclusions about the added value of our approach and future plans.

2. Theoretical background

We consider Learning Technology Systems, to be comprised of a human subsystem (learners, tutors, administrators etc.), a software subsystem, and a subsystem of miscellaneous non-software resources (workstations, computer networks, printed material etc.). In this section we will present the theoretical background of this paper, by focusing on two different concepts: 1) the holistic architecture of LTS, that contains all the aforementioned subsystems and is an interdisciplinary subject of study from engineering, instructional theory and design etc.; 2) the specific architecture of the software subsystem, which is a subject of study of the software engineering discipline. The first concept is being discussed because the holistic architecture of an LTS can actually provide the business model for the software subsystem of the LTS. The second concept is naturally

0-7695-1013-2/01 $10.00 © 2001 IEEE

being discussed because it will briefly outline the software engineering process and other concepts, needed to comprehend the proposed architecting process.

The largest effort on developing an LTS architecture has been carried out in the IEEE P1484.1 Learning Technology Systems Architecture (LTSA) workgroup. The LTSA deals with the Learning Technology System as a whole, encompassing a software system, human resources and other non-software resources and their interactions. 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 the 5 layers, where each layer is a refinement of the concepts in the above layer: "Learner and Environment Interactions", "Human-Centered and Pervasive Features", "System Components", "Stakeholder Perspectives and Priorities", and "Operational Components and Interoperability - codings, APIs, protocols". Similar work of defining abstraction-implementation levels has recently commenced within the ISO/IEC JTC1 SC36 [http://jtc1sc36.org].

To pinpoint the exact relation of the architecting process under study with LTSA, we must clarify that an architecture produced by this process cannot be straightforwardly matched into a layer of the LTSA. On the other hand, an architecture produced this way, defines software components that derive from Layer 3 components; it also takes under consideration several of the stakeholder perspectives of layer 4 and deals with some of the low-level issues of layer 5. A final point is that the LTSA does not deal with specific details of implementation technologies necessary to create the system components, while our approach suggests technologies of this kind, because they comprise a fundamental aspect of a software architecture.

As far as the involvement of software engineering to the proposed architecting process is concerned, we have chosen to adopt the USDP, an architecture-centric, use-case-driven, iterative and incremental process. The USDP incorporates the views, i.e. the most significant modeling elements, of five different models: the use-case model, the analysis model, the design model, the deployment model, and the implementation model. This set of views corresponds with the classic *4+1 views* described in [4]. Except for the five architectural views, the architecture also contains some non-functional requirements, platform decisions, architecture patterns contained and other generic features.

The notation used to describe the architecture is the Unified Modeling Language [5], a widely adopted visual modeling language in the software industry and an Object Management Group [http://www.omg.org] standard.

Another concept that we adopt from the software engineering discipline is the component-based nature of the architecting process. A software component can be deployed independently and is subject to composition by third parties [6]. Components can be plugged together, according to certain rules, and constitute greater components, also referred to as **component frameworks**.

The component-based nature of the proposed architecting process derives from the fifth and final view of the architecture description, that is the implementation model view. Together with the provision of USDP to promote a component-based architecture, our approach further enforces this by proposing binding and implementation technologies for the development of system components.

3. The process of architecting

The architecting process for the software system of an LTS combines the issues discussed in the previous section into a simple process model depicted in Figure 1.

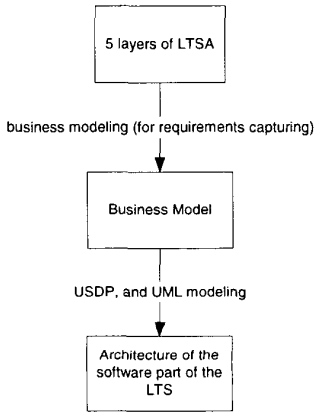

Figure 1- The macroscopic view of the architecting process for the software subsystem of an LTS

The first step produces a business model from the first 4 layers of the LTSA, so that the context of the software subsystem will be firmly grasped and all requirements will be captured. In our case the context of the software subsystem is the LTS itself, as it is particularly seen for the purposes of the system under development, e.g. a web-based distance learning system, an intelligent tutoring system etc. In other words at this stage, particular LTSA stakeholder perspectives must be chosen in order to define the business model. Next, the human activities and computer processes incorporated in these perspectives, serve as business use cases, which are the business processes involved in an LTS. Also the people and other non-human entities, which interoperate with the system, serve as business actors. The business use cases and the business actors together form the business use case model,

which is the first part of a business model. The second part comprises of a business object model, which depicts how the business use cases, i.e. the system's functionality, is realized. The result of business modeling is a complete set of the LTS's processes, fully analyzed, from an Information Technology point of view, as the LTSA does not encompass a theory of learning. In order for the requirements capturing to be completed from the pedagogical point of view, an instructional or learning theory needs to be taken under account [7].

After the business model is specified, the USDP puts into effect the workflows and builds the software architecture. Our aim though, is to produce an inherently component-based architecture with the help of the USDP. How can that be achieved? As stated in [6], 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 models describe 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, as described earlier. We shall follow this pattern in order to enforce the component-based nature in the proposed architecting process. We shall first analyze the system into component frameworks or as we simply call them *subsystems*, describe their interaction and lastly make platform decisions.

The business processes defined in the business model are transformed into the use-case model by refining the business model, and elaborating on those business use cases that relate with the software system to be developed. This results into capturing all the functional and non-functional requirements that are specific to individual use-cases. In the next workflow, that generates the analysis model, every use case will be realized in-depth, and a first-level decomposition of the system into analysis packages will be performed, also showing their dependencies and their contents, which will be used as an input to the design model.

The decomposition of the Learning Technology System is continued during the design model, by specifying the very coarse-grained discrete subsystems, as they have derived from the use case and analysis model. Especially for the purpose of identifying subsystems, the analysis packages, together with their dependencies and contents are being used as a starting point.

These subsystems, that are in essence component frameworks, are meant to be further processed by identifying their contents and specifying their interfaces. The process then continues by building the deployment model of the system, which actually maps the software components into hardware components. Finally, the last workflow of this process produces the implementation model, which defines the executable components and their dependencies on each other.

After the five models of the USDP have been completed, all the component frameworks and interoperations between them have been identified. At the last part of the component-based architecting process, we make platform and implementation decisions, that we consider to be the most suitable for a component-based system. These technologies embodied in a component development model are depicted in Figure 2.

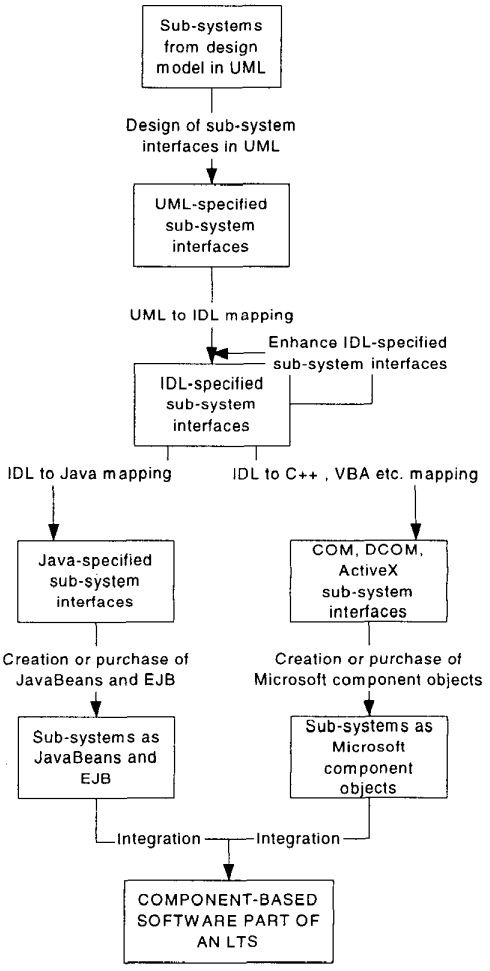

Figure 2- Component development model

The artifacts from the design model, that is sub-systems with textually described interfaces are provided as an input to the above development 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 elaborated, so that a more accurate 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 have concretely defined interfaces 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 final step is to integrate the components through an integration and testing process into the final outcome: the component-based software part of an LTS.

4. Conclusions and future work

Each one of these three key concepts of the proposed process adds special value to the proposed architecture.

To start with, the proposed architecting process professes the same principles as the LTSA, namely [8]: it provides a framework for understanding existing and future systems; it promotes interoperability, portability and reusability by identifying critical system interfaces; and it remains adaptable to new technologies and learning technology systems.

An architecture that is built with the aid of the Unified Software Development Process [3]: helps all concerned stakeholders (e.g. developers, managers, customers) to understand the system through a common language; organizes the development effort, eliminating the communications overhead; fosters reuse of system components; and helps the maintenance and evolvement of the system through development iterations and product lifecycles, thus making the system change-tolerant. Moreover, Software Engineering is unique in that it is heavily driven by risk, and architecture-based development is the primary successful approach in risk-driven engineering [9].

Last but not least, as far as the enforced component-based paradigm is concerned, it is claimed in [6] that component-based architectures are inherently modular and as such have significant software engineering advantages: good modular architectures make dependencies explicit and help to reduce and control these dependencies; are naturally layered, leading to a natural distribution of responsibilities; and it is easier to migrate part of a system by adopting relevant component interface standards.

Based on these points, it is concluded that an inherently component-based software architecture is the right step towards bringing the economies of scale, needed to build affordable, interoperable as well as effective software subsystems of Learning Technology Systems.

We are currently investigating the use of this process into real LTS implementations and the subsequent evaluation of this process. This will raise several issues such as: whether the LTSA is 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; and whether the proposed process indeed leads to a pure component-based system.

References

[1] J. M. Spector, "Trends and Issues in Educational Technology: How Far We Have Not Come", *ERIC-IT Newsletter,* Sep. 2000.

[2] The web-based education commission, "The power of the Internet for learning: moving from promise to practice", Washington DC, December 2000, available on-line at [http://www.ed.gov/offices/AC/WBEC/FinalReport/WBECReport.pdf].

[3] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[4] P.B Kruchten, "The 4+1 view model of architecture", *IEEE Software,* November 1995.

[5] G. Booch, J. Rumbaugh and I. Jacobson, *The UML User Guide*, Addison-Wesley, 1999

[6] C. Szyperski, *Component Software – Beyond Object-Oriented Programming*, ACM Press, 1999.

[7] C. McCormack and J. D. Jones, *Building a Web-based Education System*, Wiley Computer Publishing, 1997.

[8] IEEE Learning Technology Standards Committee, "Draft Standard for Learning Technology Systems Architecture (LTSA)", November 2000.

[9] T. Mowbray and W. Ruh, *Inside CORBA – Distributed Object Standards and Applications*, Addison-Wesley, 1997.