

## A service oriented architecture-framework for modularized virtual learning environments

F. Paulsson<sup>\*,12</sup> and M. Berglund<sup>1</sup>

<sup>1</sup> Umeå University, IML, SE-901 87 Umeå

<sup>2</sup> Royal Institute of Technology, Nada, Lindstedtsvägen 3, SE-100 44 Stockholm

Modularization of digital learning content has been addressed for through the concept of Learning Objects. In this article it is shown this modularized concept can be extended to the Virtual Learning Environment. A general, Service Oriented Architecture (SOA) framework for modularized Virtual Learning Environments was implemented based on the VWE Learning Objects Taxonomy. It is argued that SOA is a suitable approach for modularization, and that the resulting SOA-framework can be used as basis for implementing specialized e-learning services, specified by future standard frameworks and reference models.

**Keywords** VLE; LMS; SOA; Learning Object; e-learning; modularization

### 1. Introduction

The idea of small, context independent, chunks of digital learning material that can be used, reused and aggregated to form larger learning units for use in different contexts, has been around for more than a decade now. These “chunks” of digital learning material are usually called *Learning Objects*. Learning Object is an established term but in [1] and [2] we argue that even though the concept is well established it is still not sufficiently defined – in functional as well as in technical terms – to be useful in a way that actually give Learning Objects the characteristics that they are usually ascribed. In [2] we argue that definitions must be narrowed down and that technical properties must be made explicit. One approach could be to turn to the original source of inspiration and once again be inspired by concepts and ideas from object orientation and component-based software development. This, combined with some basic software architecture considerations, would not only help in solving some of the technical contradictions, but some of the pedagogical issues as well. In [1] we suggested a Learning Object taxonomy based on this idea: the *VWE Learning Object Taxonomy*. The VWE taxonomy is compatible with some of the existing and established taxonomies, such as the Atomic Taxonomy by Wiley [3], but it originates from the idea that there is a need for Virtual Learning Environments (VLE) with similar characteristics as Learning Objects: a VLE that is modularised and composed out of small chunks of functionality that can be aggregated into a VLE that is adapted to a specific learning context, working in symbiosis with the chosen Learning Objects. This idea originates from the assumption that we need a VLE that is as flexible as modular content in order to match this flexibility in the VLE [1]. Besides, it is often hard to establish where the content ends and where the VLE starts in terms of functionality, and this often has the effect that application logics, data and presentation is mixed in a way that makes it impossible for Learning Objects to have the alleged characteristics [2].

This article shows how a Service Oriented Architecture (SOA) approach, combined with an explicit Learning Objects Taxonomy, can be used as a basis for implementing a modular VLE. The resulting architecture is modular in the same modular sense as Learning Objects are considered to be modular.

#### 1.2 Related work: SOA, modularization and the VLE

---

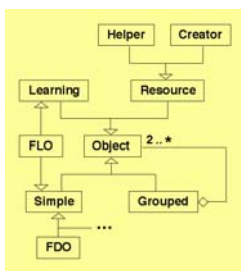
\* Corresponding author: e-mail: m-icte2006@formatex.org, Phone: +34 924258615

The SOA concept is often used as an equivalent to Web Services. Even though this article describes the implementation of an architecture framework based on Web Service technology, we mean that SOA can be used in a wider sense, and that other technologies can be used for SOA as well. An example is the OKI OSID: s, which uses Java RMI [4]. OKI OSID: s are used in the Sakai project [5].

From a general perspective, the amount of research on SOA is quite extensive. There are a couple of characteristics that are commonly emphasised: *reuse*, *interoperability* and *modularization* [6]. Those are all characteristics that can be considered to be among the most important characteristics of Learning Objects [2, 7, 8]. This was of course an important reason for exploring SOA for implementation the VWE LO taxonomy. There is some work especially focusing on SOA and e-learning as well. OKI and ELF are two important initiatives that are compared by Norton in [9]. Liu et al. suggests a SOA-based functional model for LMS and LCMS systems in [10] and in [11], Gütl et al., suggest a SOA approach with a focus on adaptation systems. In [12] Smythe et al. suggest a set of general architectural patterns for e-learning systems. While there are some theoretical work on SOA frameworks and architectural models for e-learning, there are still not many implementations to be found. The focus of existing implementations is mainly a “traditional” focus on exposing functionality from legacy systems as services in a SOA, and rarely on new modular (as in the VWE taxonomy) architectures, where complex functionality can be constructed by aggregating simple, loosely coupled, services. However, most existing SOA work emphasis the benefits of modularity. In most cases there is an understanding of that clear and simple interfaces and data models are required, as well as an understanding of the importance of standards and descriptive information (metadata) about services and modules in a SOA architecture.

### 1.3 The VWE Learning Object taxonomy

The idea behind the VWE taxonomy is to facilitate the separation of application logics, data and presentation by regarding some basic architectural principles. Figure 1 outlines the VWE LO taxonomy, which introduces a couple of new concepts. Of particular relevance are the two different types of *Resource Objects*. The main purpose of the Resource Objects is to add application logics in such way that it is separated from data and presentation. The first type of Resource Object is called *Helper Resource Object*, and its main purpose is to “support” data and content with application logics. In some ways the task of the Helper Resource Object is similar to the task of a web browser plug-in. Hence, Helper Resource Objects are used to make the fundamental building blocks of a Learning Object, such as a Fundamental Data Object (or Fundamental Learning Object according to Wiley), usable in the context of a *Grouped Learning Object* or Learning Module, by providing the application logics needed for data processing, interactivity and presentation.



**Fig 1.** A concept map outlining the VWE Learning Object Taxonomy

The *Creator Resource Object* is the second type of Resource Object and it is primarily used to add functionality to the VLE, or to Grouped Learning Objects. Creator Resource Objects are actually small stand-alone pieces of “software” that can be used as building blocks. Besides acting as building blocks, Creator Resource Objects are responsible for keeping the VLE together, and for managing the VLE internal and external communication and interaction. Each Creator Resource Object has one or more corresponding services. The VWE Learning Object taxonomy is described in detail in [1].

## 2. Implementation of a SOA framework

In this article we explore SOA as an approach to modularization of the VLE. The empirical basis was mainly collected through an experimental implementation of the VWE Learning Object Taxonomy [1].

The main purpose is not to state exactly what services are needed for an e-learning architecture. The focus is instead on a general SOA framework that addresses general needs without adding the limitations that are often present when limiting the approach to concepts such as LMS, LCMS and similar. The assumption is that a general SOA framework can be used as a general layer for implementing additional, more e-learning specialized services, according to different frameworks for this purpose. Figure 2 outlines the SOA framework that is currently implemented in Java. New functionality can however (theoretically) be developed using other technologies as well. As this is an experimental reference implementation of a SOA architecture-framework, there is a certain Java focus for practical reasons.

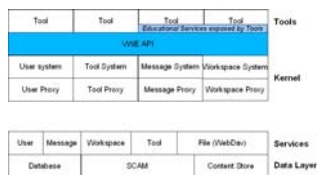


Fig. 2. A bird's view of the SOA framework

### 2.1 Tools and workspaces

A *Tool* is the metaphor used to describe functionality modules that are added by a Resource Object – service pair. Usually a tool consists of a *resource object* and a corresponding *service*. A tool can make use of more than one service and can be composed by pooling existing (and new) services through aggregation of resource components. A tools corresponding service commonly uses services exposed by the general SOA framework and/or by other tools. Figure 3 outlines the general design pattern for tools.

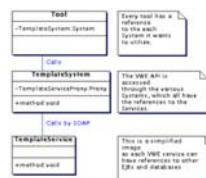


Fig. 3. Outlines the general design for each Tool-Service implementation.

A *Workspace* is the metaphor used for the part of the system that users interact with through the web browser. A user can have any number of associated workspaces, shared as well as personal. A small Java application called the *Kernel* is downloaded to the browser when a workspace is accessed. The Kernel works as a broker and manages all the interaction between the browser and the system. Workspaces are created using a tool called *Workspace Editor* (WE), through the `String createWorkspace(WorkspaceDTO ws)` method. The WE generates the workspace and all corresponding data, including metadata and a Content Package structure for the workspace. The WE uses the *Workspace Service* as its “corresponding service”. Both tools and Learning Objects can be added at creation time. A workspace can be modified at any time, and functionality or content can be added or removed. Tools are dynamically loaded at run-time, which means that a tool is first downloaded when it is requested. Dynamic loading has some advantages; the latest version is always loaded, it decreases initial download and the system is not limited to server execution. There are some disadvantages as well: the system relies on a working broadband connection and it can be sensitive to browser-specific implementations. The current implementation uses Java on the client, but tools could be implemented using any browser-based technology,

such as Flash or ActiveX. In theory, the only requirement for a tool is that it can communicate with the needed services.

## 2.2 A common Service Architecture

All services have a common basic design (see figure 3) that consist of three parts: a *Tool*, which runs on the client, a *System*, which is a Java API for interaction with services and a *Service*, which is the SOAP service that is exposed externally. As is shown in figure 2, there are five basic services. The *User Service* manages everything that is related to users, such as permissions (through the use of ACL), relations to workspaces etc. The User Service stores information in a database or a LDAP catalogue.

The *Message Service* handles all asynchronous messaging through *channels*. Each tool that implements the Message Service creates a specific type of channels. A channel is created when a user executes a tool. The main reasons for designing a common Message Service is to avoid the complexity of several different messaging implementations at the same time, the level of prior knowledge needed is decreased by hiding messaging complexity for tool implementers.

The *Workspace Service* is the service that keeps the VLE together. It is used to create new workspaces, assigning workspaces to users and user groups, load workspaces at logs-in, associate tools with workspaces etc. The Workspace Service API is used for the interaction between tools and workspaces. The structure of a workspace is expressed using IMS Content Packaging. When a workspace is launched, a Content Package is downloaded to the browser by the kernel. The Workspace Service uses a WorkspaceDTO as its data transfer object and the DTO is used as the data type for representing workspace data in SOAP the communication between tools (ToolDTO) and workspaces.

The *Tool Service* is used to locate, load, run, manage, and deploy tools and content. Every component in the system has associated metadata. This means that an extensive amount of metadata has to be managed by the Tool Service and the more modularized a system becomes; the more sophisticated metadata is needed to describe services and artefacts. For this reason all metadata in the system is handled as RDF and stored in a SCAM-repository [13]. Metadata is primarily used for three things: for locating resources, for describing its properties for humans, and for describing its properties for the system. The ToolDTO carries the metadata for a specific tool or object and manages the interaction with the Workspace Service. For performance reasons tool metadata is cached the first time it is accesses. If the tool is a *Helper Resource Object*, it manages the metadata for the associated Learning Objects as well. The metadata is compliant to the *Learning Object Metadata* (LOM) standard [14], with the exception of three system specific metadata fields that are used internally. The choice of LOM was made for interoperability reasons. Every tool must implement the Tool API to function with the system. The reference implementation contains a few example tools and a couple of tools for editing and deploying new tools.

The *File Service* is used for storing and managing files in a distributed file system. File metadata is managed by SCAM, and the files can be stored in any WebDav storage. As metadata is expressed using RDF, the files location must be expressed using an URI. The SCAM ePortfolio is used as WebDav storage in the reference implementation. A problem related to the File Service is keeping files and their metadata together. This is mainly caused by the poor WebDav metadata support and there is no standard way of binding RDF metadata to a file as the URI only works one-way.

## 3. Results, discussion and future work

The SOA framework presented in this article has served as basis for a working implementation that provides a good foundation for modularization of the VLE. By modularizing the VLE, new functional components can be easily added in a way that makes them work as an integrated part of the overall learning environment. In addition, legacy systems can better function together with the VLE. Besides showing that SOA is a practicable path towards modularization, it is evident that it is possible to implement a highly modularized model such as the VWE Learning Object taxonomy.

Many design decisions that have to be made in order to implement a SOA framework for VLE: s. One of the most important design decisions is whether to implement a number of general services that learn-

ing services can implement, or whether to specify a larger number of e-learning specific services – as is the case with ELF. We argue that it is better to start out with a smaller number of general services, which can later be used to implement e-learning specific services. The reasons for this are twofold: first, by specifying exactly what services are needed, there is a risk of bringing in limitations – both technical and pedagogical – of the same kind that exists in many LMS, and second; by implementing a general service framework it becomes much easier to add new pedagogical services where general services can be used as “building blocks”. This makes it possible to support a multitude of upcoming SOA frameworks for e-learning, such as ELF and OKI OSID: s, by adding those in an additional service layer implemented by a corresponding set of VWE Tools. In comparison, OKI and ELF approach this issue differently. While OKI has a general approach, that somewhat resembles our approach; ELF has a much more detailed service architecture, focusing on pedagogical services that are defined in detail.

We will explore how compliance to frameworks such as ELF can be accomplished as an additional layer to the VWE general service framework in our future work. There is also a need to explore how to further enhance the service framework in such ways that it becomes easier for developers to add new components and functionality. It is necessary to make the architecture less Java centric. For the purpose of “proving the point” a Java centric architecture is acceptable, but when a suggestion for a general SOA for modularization of the VLE is presented, there is a necessity to be as technology neutral as possible. At the server (service) side a more general (preferably standardized) set of interfaces is needed, that can replace the VWE API and on the client there is a need to explore more open and browser centric technologies, such as Ajax.

## References

- [1] F. Paulsson and A. Naeve, "Virtual Workspace Environment (VWE): A Taxonomy and Service Oriented Architecture Framework for Modularized Virtual Learning Environments - Applying the Learning Object Concept to the VLE," *International Journal on E-Learning*, vol. 5, pp. 45-57, 2006.
- [2] F. Paulsson and A. Naeve, "Establishing technical quality criteria for Learning Objects," presented at To be published in the proceedings of eChallenges 2006, Barcelona, Spain, 2006.
- [3] D. A. Wiley, "Connecting Learning Objects to Instructional Design Theory: A Definition, a Metaphor and a Taxonomy," in *The Instructional Use of learning Objects*, D. A. Wiley, Ed. Bloomington: Agency for Instructional Technology and Association for Educational Communications & Technology, 2002, pp. 298.
- [4] S. Thorne, C. Shubert, and J. Merriman, "OKI Architecture Overview," MIT, White Paper March 22 2002.
- [5] C. Counterman, G. Golden, R. Gollub, M. Norton, C. Severance, and L. Speelman, "Sakai Architecture." [www.sakaiproject.org](http://www.sakaiproject.org); Sakai Project, 2004.
- [6] E. Söderström, "Serviam Literature Survey Part II Business Value," vol. 2006. Stockholm: Serviam, 2005.
- [7] T. Koppi and N. Lavitt, "Institutional Use of Learning Objects Three Years on: Lessons Learned and Future Directions," presented at Edmedia 2001: World Conference on Educational Multimedia, Hypermedia & Telecommunications, Honolulu, Hawaii, USA, 2003.
- [8] C. Quinn and S. Hobbs, "Learning Objects and Instruction Components (Pre-discussion paper)," *Educational Technology & Society*, vol. 3, 2000.
- [9] M. J. Norton, "A Comparison between the JISC and Sakai Frameworks," vol. 2006, 3rd ed: the Sakai project, 2004.
- [10] L. Xiaofei, A. El Saddik, and N. D. Georganas, "An implementable architecture of an e-learning system," presented at Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on, Montreal, Canada, 2003.
- [11] C. Gütl, V. M. G. Barrios, and F. Mödritscher, "Adaptation in E-Learning Environments through the Service-Based Framework (SBF) and its application for AdeLE (Adaptive e-Learning with Eye-Tracking)," presented at World Conference on E-Learning in Corp., Govt., Health., & Higher Ed. (ELEARN), Washington, USA, 2004.
- [12] C. Smythe, J. Evdemon, S. Sim, and S. Thorne, "Basic Architectural Principles for Learning Technology Systems," IMS, Whitepaper June 17 2004.
- [13] M. Palmér, A. Naeve, and F. Paulsson, "The SCAM Framework: Helping Semantic Web Applications to Store and Access Metadata," presented at the European Semantic Web Symposium 2004, Heracleion Greece, 2004.
- [14] "Final 1484.12.1-2002 LOM Draft Standard." IEEE: IEEE-Standards Association, 2002.