Design and Implementation Strategies for IMS Learning Design

Design and Implementation Strategies for IMS Learning Design

(Ontwerp en Implementatiestrategieën voor IMS Learning Design)

“*In the beginning the Universe was created. This has made a lot of people very angry and has been widely regarded as a bad move.*”

Douglas Adams

Design and Implementation Strategies for IMS Learning Design

Proefschrift

ter verkrijging van de graad van doctor
aan de Open Universiteit Nederland
op gezag van de rector magnificus
prof. dr. ir. F. Mulder
ten overstaan van een door het
College voor promoties ingestelde commissie
in het openbaar te verdedigen

op vrijdag 7 november 2008 te Heerlen
om 16.00 uur precies

door

Hubertus Franciscus Vogten

geboren op 16 februari 1967 te Sittard

SIKS Dissertation Series No. 2008-27
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research in this thesis has been carried out at the Open Universiteit Nederland as part of the OTEC Learning Networks Programme.

Printed by: Datawyse Maastricht
Cover design: Jeroen Storm

ISBN 978-90-79447-06-0

Design and Implementation Strategies for IMS Learning Design

Hubert Vogten

Synopsis

The IMS Learning Design (LD) specification, which has been released in February 2003, is a generic and flexible language for describing the learning practice and underlying learning designs using a formal notation which is computer-interpretable. It is based on a pedagogical meta-model (Koper & Manderveld, 2004) and supports the use of a wide range of pedagogies. It supports adaptation of individual learning routes and orchestrates interactions between users in various learning and support roles. A formalized learning design can be applied repeatedly in similar situations with different persons and contexts. Yet because IMS Learning Design is a fairly complex and elaborate specification, it can be difficult to grasp; furthermore, designing and implementing a runtime environment for the specification is far from straightforward. That IMS Learning Design makes use of other specifications and e-learning services adds further to this complexity for both its users and the software developers.

For this new specification to succeed, therefore, a reference runtime implementation was needed. To this end, this thesis addresses two research and development issues. First, it investigates research into and development of a reusable reference runtime environment for IMS Learning Design. The resulting runtime, called CopperCore, provides a reference both for users of the specification and for software developers. The latter can reuse the design principles presented in this thesis for their own implementations, or reuse the CopperCore product through the interfaces provided. Second, this thesis addresses the integration of other specifications and e-learning services during runtime. It presents an architecture and implementation (CopperCore Service Integration) which provides an extensible lightweight solution to the problem.

Both developments have been tested through real-world use in projects carried out by the IMS Learning Design community. The results have generally been positive, and have led us to conclude that we successfully addressed both the research and development issues. However, the results also indicate that the LD tooling lacks maturity, particularly in the authoring area. Through close integration of CopperCore with a product called the Personal Competence Manager, we demonstrate that a complementary approach to authoring in IMS Learning Design solves some of these issues.

Table of Contents

Chapter 1

Introduction

Introduction

In 1998 the first steps towards the formal definition of an educational modelling language were taken by the Open University of the Netherlands (OUNL). This resulted in the definition of an XML (W3C, 2003) language called Educational Modelling Language (EML) (Koper & Manderveld, 2004; Hermans, Manderveld, & Vogten, 2004; Koper, Hermans, Vogten, & Brouns, 2007). The development of EML has been an iterative process of design, implementation, test and evaluation over approximately three years. The EML specification (EML 1.0, 2000; EML 1.1, 2002) has now been evaluated in experimental settings within OUNL but also by interested external organizations. A series of Edubox (Tattersall, Vogten, & Hermans, 2005b) tools were developed to make these pilots possible.

Reinforced by the positive experiences with EML, a slimmer version of EML 1.0 was nominated for standardization by IMS and approved by the IMS Technical Board in February 2003 under the name IMS Learning Design (LD) (Koper & Olivier, 2004; IMSLD-IM, 2003; IMSLD-BPG, 2003; IMSLD-XB, 2003; Koper & Tattersall, 2005). LD is a generic and flexible formal language, which is computer-interpretable, for capturing and describing the learning practice and underlying learning designs; it is based on a pedagogical meta-model and supports the use of a wide range of pedagogies; it supports adaptation of individual learning routes and is capable of orchestrating interactions between users in various learning and support roles. Especially these multi-role, multi-user aspects set LD apart from other specifications such as IMS Simple Sequencing (2006) and SCORM (2008). The same formalized learning design can be deployed over and over again via a runtime environment using different persons.

For LD to succeed, a reference runtime implementation was needed which could support educational designers in better understanding the specification. The existing Edubox EML toolset was ill-suited for this purpose because its design and architecture is very monolithic, and specifically adjusted to OUNL circumstances. Although LD and EML are based on the same pedagogical meta-model (Koper, 2001; Koper & Van Es, 2004) and are thus conceptually comparable, their language implementations are quite different. Edubox would require a complete overhaul to cope with these language changes.

A new reference runtime environment therefore had to be designed from scratch. In contrast to the Edubox toolset, it should be reusable in different environments and circumstances. From this stems the first research and development question of this thesis:

> *i) How can a fully compliant reusable reference runtime environment for the IMS Learning Design specification be designed and implemented?*

LD relies and builds upon other e-learning specifications. For example, IMS Question and Test Interoperability (IMSQTI, 2006) is used for assessment functionality, and IMS Content Packaging (CP) (IMSCP-IM, 2003) is used for bundling units of learning. Although the LD information model describes how these specifications should be incorporated into LD at the lexical and syntactical level, little has been written about the implications for the runtime. In addition, LD provides for the integration of learning support services such as forum and chat services. These are integrated by declaration and should be linked into the runtime during the learning design execution. The following XML snippet shows the declaration of a forum service in LD.

```
<service identifier="GB_Confer_SO">
  <conference conference-type="asynchronous">
    <participant role-ref="GB"/>
    <participant role-ref="Teacher"/>
    <participant role-ref="Expert"/>
    <item identifier="I-GB_Confer_SO" identifierref="RES-ITALY "/>
  </conference>
</service>
```

There are many implementations for these specifications and learning support services; often they play integral roles in learning management systems. Therefore, using these implementations rather than re-implementing them from scratch is a sensible prospect. It should be possible to integrate these existing implementations with the LD runtime. This leads to our second research and development question:

> *ii) How, given a reference implementation for the IMS Learning Design specification, can implementations for other e-learning specifications and learning support services be integrated generically at runtime level?*

We will address research question (*i*) by elaborating it. A reference implementation being fully LD compliant implies that all the three specification levels (see chapter 2 for details) are supported. Any valid LD instance should be executable on this reference implementation. The resulting runtime behavior should be in complete compliance with the specification. An implementation such as LAMS (LAMS, 2008), for example, which was inspired by LD, does not meet this criterion because it is not capable of importing any arbitrary LD compliant learning design. Furthermore, the implementation should act as a reference for practitioners who want to better understand the specification by allowing them to run and validate their own learning designs. Those interested in developing their own LD runtime environment should be supported by a validated design that demonstrates how to implement a runtime environment.

There are several reasons why implementing an LD runtime environment is not a trivial task. LD is an elaborate specification merely by its size alone: it consists of more than 250 distinct language elements on top of those already defined in CP. It is also a domain-specific language (Deursen, Klint, & Visser, 2000), which implies that implementers of an LD runtime should have ample involvement with the instructional design domain. In addition, it is a declarative language that can be interpreted in two ways, both of which apply to LD. In the first interpretation, declarative language is used in the sense of ‘what something is like, rather than how to create it’. We have already mentioned the example of a forum service declaration. This characteristic of LD allows authors, for example, to declare quickly which supporting services should be available during runtime. However, an apparently simple declaration requires much scaffolding by runtime implementers in their applications. In the second interpretation, declarative language is synonymous for a non imperative programming language such as XSLT (W3C, 1999). The LD condition elements may at first seem imperative, but they are in fact also declarative, and resemble the production rules of a production system (Brownston, Farrel, Kant, & Martin, 1985). The runtime must continuously decide which conditions should be evaluated at any given moment. Furthermore, it must also detect when to stop evaluating these conditions to avoid getting stuck in potentially endless loops. Evaluation of the conditions, however, is not triggered by forward or backward chaining (as is the case in production systems), but rather by events.

LD is also a persistent language (Atkinson, Bayley, Chilsom, Cockshott, & Morrison, 1990), implying that the lifetime of entities, particularly properties, supersede their runtime execution lifetime. Thus alterations to properties and other entities should be automatically persisted by the LD runtime environment. Finally, LD shares the characteristics of workflow languages such as BPEL (IBM, BEA Systems, Microsoft, SAP AG, & Siebel Systems, 2006) and XPDL (Workflow Management Coalition, 2005), allowing the multi-user modelling of learning flow. The runtime environment for LD should be capable of orchestrating this workflow.

All these characteristics combined make clear that any runtime implementation, regardless of how clever its design may be, will demand considerable time and resource investment. Therefore, it is likely that most implementers would prefer to reuse an existing LD runtime implementation rather than build their own from scratch. The reference runtime implementation should therefore be reusable, providing hooks that allow integration into various environments and architectures. It should make as few assumptions as possible about the environment in which it will be integrated. It should thus also not make assumptions about the user interface. A clear separation exists between the engine of the LD runtime, dealing with the processing of LD business rules, and the representation in a user interface. This thesis focuses on the engine of this runtime. Reusability also entails that it is possible to tune and adapt the reference runtime environment to specific situations and research.

In chapter 2 a detailed overview of LD is provided as a background for the reader. Although the LD specification itself is not the topic of this thesis, ample understanding of the specification is helpful in grasping some of the issues involved in the design and implementation of a runtime environment. We see that the LD specification is available at three levels, referred to as A, B and C. These levels are incremental and inclusive, meaning that level B includes level

A, and level C includes levels A and B. We posit that an application is only *fully* LD compliant when it implements level C of the specification.

We illustrate that level A is domain specific and declarative. It provides for the declaration of users and user groups in the form of roles. Furthermore, learning and support activities can be defined with their environments, consisting of resources and services. These activities may be grouped into sequences and selections. These structures make up the major building blocks of the specification. Next we see that it is possible to combine these building blocks into a *learning flow*. This learning flow orchestrates which activities have to be performed when and by whom; it also defines dependencies and relationships between these activities.

Level B adds a new dimension to the LD specification via the introduction of properties and conditions which make it possible to model runtime adaptations in the design (Burgos, Tattersall, & Koper, 2007). They resemble structures typically found in regular programming languages, but with a twist. We explain that, for example, properties have an instantiation scope and are assumed to be persistent beyond their execution lifetime. Furthermore, we show that the order of interpretation of the LD conditions is not imperative, that is, not determined by the order in which they appear in the design. Properties and conditions together provide means for personalization and adaptation during runtime.

The final step towards level C of the specification is relatively small: level C only adds a notification mechanism. Notifications inform users about changes of properties in the system, which can trigger users to take further action.

Next we see how LD instances are packaged using the IMS CP specification to construct units of learning (UOL). These UOLs are the input for the LD learning design engine discussed in the following chapter. Finally, we define some high level requirements which have to be met by any LD engine.

Chapter 3 introduces the notion of an engine as a software component capable of interpreting a UOL by providing a personalized view of it. This is the result of applying all rules defined in the UOL as defined by LD. The engine output is an XML format that can be used by a player to generate the user interface. This thesis focuses on the engine's design and implementation, although an example player is also provided as part of the software release.

We demonstrate how an engine design can be formulated when we take the perspective of a finite state machine (FSM) (Sipser, 1997). A state is represented by the values of the properties defined in the UOL; however, it is also defined by progress information within the learning flow. We demonstrate that such progress information can also be captured with the same properties if we slightly extend the instantiation scopes defined in LD. We distinguish between explicit properties and implicit properties: the former are defined in the UOL by the UOL's author, whereas the latter are defined by the engine when parsing the UOL. We also clarify the concept of a run, and see how this helps assigning real users to a UOL; users must be assigned to one or more roles defined in the UOL for each run. We discuss how the concept of runs and roles influences the scoping of the properties, and show what extensions the scoping definitions need in order to support the notation of the explicit properties. The scope of a property determines how many distinct instances of this property will be generated by the engine during runtime. Combining the population of the UOL with the run and the role concepts leads to the conclusion that an LD engine may be perceived as a collection of FSMs in which each distinct FSM is defined by the user, role and run identities.

Next we define the input and output alphabets of the FSMs. Both are expressed in the form of events specific to the UOL. These events lead to state changes and can trigger an output function. In turn, an output function can trigger new events, leading to new state changes. This mechanism causes state changes to ripple through the chain of events.

We then take a closer look at the event handling by way of an event dispatcher, a collection of LD rules defined by the UOL, several event handlers and a property store. We show that property changes trigger one or more event handlers depending on the rules defined by the UOL. These event handlers can modify properties themselves and thereby raise new events. We show that it is possible to express the LD business logic such as, for example, determining the completion status of an act by event handlers. However, it is also possible to express the conditions defined in the UOL through such event handlers, which solves the problem of determining when to evaluate the conditions. Their evaluation is also triggered by state changes.

Finally, we discuss in more detail how the UOL is parsed, showing that the UOL is split up into smaller fragments. These fragments are personalized when retrieved by inserting the property values as defined for the user in a particular role and run into the XML snippets. We show that personalization of the UOL has become almost trivial through the introduction of the FSM.

This approach provides a solid guideline for implementing an LD runtime environment. Using this design we built the CopperCore runtime engine, which has been released as open source and can be used as reference implementation by anyone interested in building their own application or integrating the engine into their own environment.

Chapter 4 describes the CopperCore reference implementation and, more specifically, the provided APIs. In this chapter we discuss how CopperCore can be reused through the provided Application Programming Interfaces (APIs). First, CopperCore's two APIs are described. The first interface, the CourseManager, deals with administrative tasks such as the publication of UOLs as well as the creation of users and their assignment to runs and roles. It contains all calls needed to prepare a UOL for execution; we discuss the most commonly used calls in more detail.

The next interface is the LDEngine API. The name itself suggests that it is focused on the execution of the UOLs prepared earlier through the

CourseManager. We discuss the three major calls of this API: *getActvityTree(), getEnvironmentTree() and getContent().* All these return fairly extensive personalized XML fragments that need further rendering by the calling party: for each call we give a detailed description of the returned XML format. After discussing the APIs, a sequence diagram that depicts an example of interaction between a user, a client LD player and the CopperCore engine is investigated in more detail.

The second part of the chapter addresses CopperCore's architecture. We show how the APIs and some of the constructs discussed in chapter 3 are implemented with J2EE (J2EE, 2007). We motivate our choice for the J2EE specification and discuss implementation strategies that can be pursued when deploying CopperCore.

To address research question (*ii*), in chapter 5 we discuss the integration of other specifications and learning support services with CopperCore. We use the example of IMS QTI to describe the CopperCore Service Integration (CCSI) architecture. The CCSI development took place sometime after the public release of CopperCore; we demonstrate how CCSI can be embraced with minimal intrusions on existing code. We see that CCSI can be wedged between a CopperCore client and the CopperCore engine, requiring minimal client side code changes. We also illustrate how the proxy pattern (Gamma, Helm, Johnson, & Vlissides, 1995) helped us achieve this objective.

We acknowledge that there may be many implementations for each service and specification; new specifications and services may also emerge and require integration. We show how we used the bridge pattern (Gamma et al., 1995) for this purpose, and introduce the concept of adapters. CCSI requires that adapters be provided for each specification and service, including CopperCore itself. Using the IMS QTI integration, we show that there is a need for these adapters to communicate with each other. We introduce a dispatcher for this purpose that acts as a kind of service bus, relaying events between the adapters. Furthermore, we show how this dispatcher also acts as a factory, providing for the dynamic linking of adapter implementations.

Finally, we discuss the implementation of CCSI and provide some screen captures of a working example of the UOL with some IMS QTI items incorporated. This example illustrates how answering a question influences the learning flow defined in the UOL. We argue that the same principle applied for the IMS QTI integration can be applied to a whole range of services.

Having addressed our two research and development questions, in chapter 1 we reflect on the impact of CopperCore and CCSI on the LD community by reviewing the use of both products in other research and developments. The CopperCore development has been an iterative process carried out in the context of several externally funded projects. We discuss how these projects provided valuable validation opportunities for both the engine and the service integration. We also discuss the UNFOLD project that provides a platform for the LD community. We briefly touch upon the research and developments presented in the context of UNFOLD, and elaborate on the role of CopperCore and CCSI.

Finally, we take a closer look at the reuse of CopperCore in the TELCERT and ELeGI projects. These projects acknowledge CopperCore as the de facto reference runtime environment for LD. We conclude that a number of learning design authors have been using CopperCore as a reference to help them better understand the specification; at the same time, they have also been validating the engine by deploying and testing numerous designs for all specification levels. We show that the engine has been reused in various settings and that new services have been successfully developed for CCSI. We argue, therefore, that we have successfully addressed our two research and development questions.

However, we also conclude that the omission of an easy-to-use authoring environment has hampered further uptake of the specification. In chapter 7, we address this authoring issue by providing a complementary approach to LD authoring which relies on close integration of CopperCore and CCSI. We argue that the current toolset for LD authoring requires ample understanding of the specification. Furthermore, we argue that the current toolset, including CopperCore, favors a top-down design approach, yet many practitioners favor a more bottom-up approach to designing their learning (such as that provided by the Personal Competence Manager, or PCM). Moreover, for many of these practitioners the LD learning curve is too steep.

We start by presenting the TENCompetence (TENCompetence consortium, 2007) domain model and explaining its main concepts. Next we describe the architecture of the PCM, followed by an in-depth description of the PCM using the wireframe designs available during writing. We show how users can easily create and edit their own competence plans and activities. These activities can be structured by creating competence development plans (CDP), which are targeted at attaining certain competences. The PCM allows for easy arrangement of the activities within a CPD by way of a graphical editor, it also allows for the sharing of most of these entities with other users, which users can edit together.

We demonstrate how the TENCompetence concepts can be mapped onto LD, which makes it possible to capture a competence development plan in a UOL. We recognize that working within the PCM brings ease of editing without too much concern about design formalization. The UOL however, provides the advantages of a formal learning design; LD's major advantages include quality assurance, reusability, advanced designs and accountability. It is up to the user to decide whether the advantages of formalization through a UOL outweigh those of the native PCM editing environment.

We show that an exported UOL can be enhanced with the regular LD authoring tools. It can then be imported back into the PCM by creating a new action that links to this UOL, thus closing the editing loop. This requires the close integration of an LD runtime environment: we argue that CCSI and consequently CopperCore can be used for this purpose. The editing cycle allows practitioners to adopt a bottom-up approach to designing their learning – no LD knowledge is required at this stage. Those same practitioners may decide at some point that capturing their design in a formal specification is preferable. If the design needs further enhancement, ample LD knowledge is required; however, a working design is available as starting point for this editing, which can be very helpful.

In the remaining sections we reflect on the technical implications of importing the UOL into the PCM. We explore the issues involved in the provision of services, assignment to runs and population of roles. We argue that CCSI provides a good starting point for this integration, but also recognize that some issues require further analyses.

Finally, in chapter 8 we review our findings and elaborate on our experiences during the development and use of CopperCore and CCSI. We also reflect on their shortcomings and propose specific areas for further research and development.

Chapter 2

IMS Learning Design

IMS Learning Design

The first part of this chapter provides an overview of the LD specification and serves as background for readers unfamiliar with it. The last section provides some requirements and considerations for the runtime implementation of LD. Although LD itself is not the focus of this thesis but rather a given, a basic understanding of the specification will help to understand the challenges for the design and implementation of an associated runtime. Koper (Koper, 2005a) states that every learning practice has an underlying learning design, just as every building has an underlying architecture. As in architecture, where similar buildings can be constructed using the same design, the learning design can also be applied over and over again in similar situations. However, this learning design can be implicit, and in education there is no common practice of using a formal notation for such a learning practice. Koper also indicates that this lack of a formal, commonly understood notation hampers broader communication about effective educational practice, and impedes the evaluation of existing designs.

Derived from theory, examples and patterns, Koper set out the following requirements for a formal learning design notation.

“1. The notation must be comprehensive. It must describe the teaching and learning activities of a course in detail and include references to the learning objects and services needed to perform the activities. This means describing:
   - How the activities of both the learners and the staff roles are integrated.
   - How learning resources (objects and services) are integrated.
   - How both single and multiple user models of learning are supported.
2. The notation must support mixed mode (blended learning) as well as pure online learning.
3. The notation must be sufficiently flexible to describe learning designs based on all kinds of theories; it must avoid biasing designs towards any specific pedagogical approach.
4. The notation must be able to describe conditions within a learning design that can be used to tailor the learning design to suit specific person or specific circumstances.
5. The notation must make it possible to identify, isolate, de-contextualize and exchange useful part of learning design (e.g. a pattern) so as to stimulate their reuse in other contexts.
6. The notation must be standardized and in line with other standard notations.
7. The notations must provide a formal language for learning designs that can be processed automatically.
8. The specification must enable a learning design to be abstracted in such a way that repeated execution, in different settings and with different persons, is possible.”

These requirements led to the development of LD 1.0, first released on 20 January 2003. LD is a formal language using XML as its meta-language. The official documentation can be found on the IMS (2003) website. IMS has developed a set of three document types for its specifications: an information model (IMSLD-IM, 2003), a best practices and implementation guide (IMSLD-BPG, 2003) and an XML binding (IMSLD-XB, 2003) document. The information model is the normative description of the specification. The best practices and implementation guide provides more information and background, including scenarios, examples and guidance to specification implementers. Finally, the XML binding document describes the XML grammar used to capture the concepts described in the information model. LD recognizes three levels of the specifications – A, B and C. Each level builds on the previous one, and extends the specification with additional features. Learning designs as well as implementations are considered compliant to a level. For learning designs this means they only use elements of that specific level; for implementations, all language elements defined for that level are supported. Compliancy to a higher level automatically implies compliancy to all lower levels.

In the following sections, LD is discussed in detail. First, its base concepts and constructs are described as defined in level A, followed by the more advanced features covered by levels B and C.

IMS Learning Design concepts

Figure 2.1 depicts the core components of LD level A.

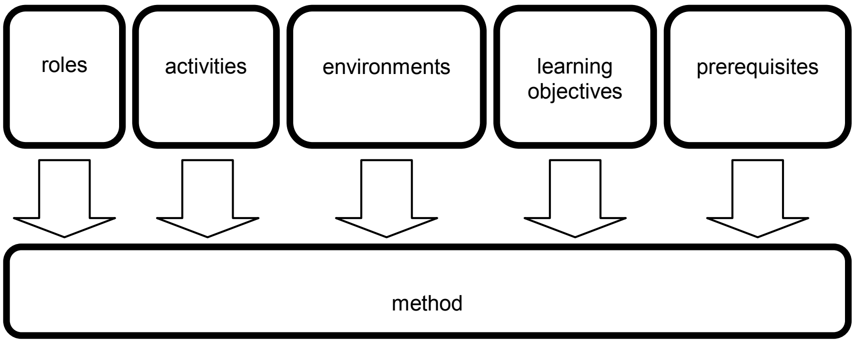


Figure 2.1 Level A components of IMS Learning Design

Level A is the most basic version of the specification; it includes core components only. The backbone of LD is formed by the learning design, which is the wrapper for all the concepts we discuss in the following sections. The method describes the workflow of learning and teaching processes defined by the learning design. We shall call this the *learning flow*. To construct the learning flow, the building blocks of activities, roles, environments, learning objectives and prerequisites are required. For the time being, it is sufficient to think of the method as a construct that defines *who* should do *what* at a particular *moment* in time.

ROLES

Roles consist of a collection of role elements. A role is an abstraction of the responsibility a user has during the learning process. Two role types are distinguished in LD: learner roles and staff roles. These can be organized into hierarchies specifying their dependencies. A user who has been assigned to a role deep in the hierarchy is also implicitly assigned to all ancestor roles. Besides the distinction between staff and learner roles, no particular semantics are imposed or assumed by LD, and the learning designer is free to add any number and type of roles to the design. Although roles are primarily intended to describe the nature of a user's involvement in the design, they can also be used to group people. Defining the roles and hierarchies is part of the design process; populating the roles with actual users, however, is done during runtime.

Additional instances of these roles may be instantiated during runtime if the learning design permits. This way the exact number of instances needed for a particular role can be defined during runtime. For each role, a minimum and maximum number of users that must/may be assigned can be specified. Furthermore, it is possible to indicate whether sub-roles are exclusive, meaning that the same user cannot be assigned to more than one sibling role simultaneously. A simple XML snippet of some role definitions might look like this:

```
<roles>
  <learner identifier="Learner" create-new="allowed" max-persons="10">
    <title>Learner</title>
  </learner>
   <staff identifier="Teacher" create-new="not-allowed">
     <title>Teacher</title>
     <staff identifier="Recorder">
       <title>Recorder</title>
     </staff>
  </staff>
</roles>
```

This snippet declares one learner role that may be instantiated many times during runtime. In each instance of this role only 10 people may be assigned. Furthermore, a staff role with the name ‘Teacher’ is defined that may be instantiated only once; this role can be further refined as a ‘Recorder’.

ACTIVITIES

Activities are the next building blocks for the method. They define what a learner or teacher has to do according to the learning design, and come in two flavors: learning activities and support activities. The learning activities are targeted at learning and performed by users in the learner roles. In contrast, the support activities are intended to be performed primarily by members of the staff roles. However, in certain pedagogical scenarios these support activities have to be performed by learners as well; in such cases, the support activity is a special kind of learning activity.

Besides semantic differences, the support activity also has an additional reference to roles which represent the users who will benefit from this support. How activities are completed is defined by the learning design. In LD level A, for example, they can either be completed by choice of the user or, alternatively, by a time limit. In level B, more sophisticated completion mechanisms are available. The learning designer can add feedback that should be presented by the runtime once an activity is completed. Each activity also has an environment, which associates the necessary learning resources with the activity. Activities themselves can be structured into sequences and selections via the activity structure element. Sequences group activities in a fixed order; these activities must be rendered in this order during runtime, and each following activity will only become available after the previous one has been completed. Completing the last activity will also complete the sequence itself. Selections also group activities, but, unlike sequences, all activities will be rendered at once. Users may pick activities from this list of activities. Selections have thresholds defining the minimum number of activities to be completed before the activity structure itself is completed. Activity structures may nest other activity structures, allowing the creation of fairly complex hierarchies of sequences and selections. Although the activity structure also defines part of the teaching–learning flow, it is not considered part of the method.

A sample XML snippet of some looks like this:

```
<activities>
  <learning-activity identifier="Reflect">
    <title>Reflect on Experience</title>
    <activity-description>
      <item identifier="Reflect_AD_res" identifierref="an_id"/>
    </activity-description>
    <complete-activity>
      <user-choice/>
    </complete-activity>
    </learning-activity>
..
..
  <activity-structure identifier="Reflection"
                            structure- type="sequence">
    <title>Reflection</title>
    <learning-activity-ref ref="Reflect"/>
    <learning-activity-ref ref="Describe"/>
    <learning-activity-ref ref="Create_page"/>
    <learning-activity-ref ref="Post_page"/>
    <learning-activity-ref ref="Review_Outcomes"/>
  </activity-structure>
</activities>
```

The snippet above defines a learning activity with the title ‘Reflect on Experience’. Activity descriptions define items, which can be thought of as placeholders for resources. In principle, all resource types and formats are allowed, but in practice these resources are often web pages. When elaborating on level B, we see that LD also has a special resource type that requires different treatment. Resources are either bundled with the learning design itself, or are accessible via an absolute URL. The activity defined in the XML snippet can be marked as completed by the user (the runtime should present the user with an option to do so). Note that this activity is atypical because it does not contain a reference to an environment.

The next part of this snippet shows an activity structure ‘Reflection’ that has been defined as a sequence. This sequence is built by referencing to learning activities. The runtime should present these learning activities in the same order as defined, giving access only to all completed activities and the first incomplete activity.

ENVIRONMENTS

An environment is a container of resources aggregated into hierarchies. Resources can be simple learning objects or, alternatively, one of following learning services: send mail, conference, index search and monitor. LD allows the extension of this list in the future. Learning objects can refer to one or more resources that are either bundled with the design itself or available through an absolute URL. The send mail service provides the means to send a message to users in particular roles. The conference service declares either synchronous or asynchronous conference facilities. The index search service defines access to the learning design artefacts either by defining indexes or by defining a free text search facility. The runtime is responsible for providing these services during execution of the design.

Environments can be nested, allowing their reuse in different contexts. They are typically referenced from activities. The following code snippet is an example of an environment:

```
<environments>
  <environment identifier="Italy_Background_Env">
    <title>Italian Background for the Treaty of Versailles</title>
    <learning-object identifier="Italy_Background_LO">
      <item identifier="Italy_BG_item" identifierref="Italy_BG_res"/>
    </learning-object>
  </environment>
  <environment identifier="Italy_Serbia_Confer">
    <title>Italy-Serbia Forum</title>
    <service identifier="Italy_Serbia_Confer_SO">
      <conference conference-type="asynchronous">
        <participant role-ref="ITALY"/>
        <participant role-ref="SERBIA"/>
        <participant role-ref="Teacher"/>
        <participant role-ref="Expert"/>
          <item identifier="I-Italy_Serbia_Confer_SO"
                identifierref="RES-ITALY_Agree_AD_res"/>
      </conference>
    </service>
  </environment>
</environments>
```

The snippet above is of a nested environment. The root environment element merely contains two other environments. The first sibling environment contains a learning object, and the second sibling environment an asynchronous conference service.

LEARNING OBJECTIVES AND PREREQUISITES

The learning objective element defines the intended learning outcome, while the prerequisites define the required entry level. Both learning objectives and prerequisites can be defined either in an informal, human-readable form, or through other, machine-readable specifications such as IMS RDCEO (2002). Learning objectives and prerequisites can be defined at the level of the complete design and/or that of the learning activities.

METHOD

With all building blocks in place, we now elaborate the method. The method element ties all concepts together by constructing plays, which are informed by the theatrical plays. These plays orchestrate which teachers and learners represented through their roles are 'on stage'. Furthermore, plays define what should be done when, and prescribe how the teachers and learners should interact. Like a theatrical play, the teacher–learner interactions are divided into acts that offer synchronization moments. Within each act, teachers and learners are supposed to perform their activities. All must have done so – or have left the stage, to use the dramatic metaphor – before the next act can start. Role parts are used to define which activities must be performed by whom; they associate a role with an activity, implying that members of that role should perform that activity. Alternatively, they can associate a role with an environment which is a shorthand notation for an implicit activity instructing the user merely to study the resources in that environment.

An act can contain many role parts, all of which will be performed concurrently. This can be compared with actors being on stage simultaneously. The role part is considered complete when all users have completed the associated activity: the learning design defines when the act is completed. In level A, the options are limited to defining the role parts that must be completed or, alternatively, setting a time limit after which the act is completed automatically. This time limit can be absolute or relative to the moment of initial deployment of the design. The play can be completed either by completing the last act or by defining a time limit. Finally, the method can be constructed by more than one play. Each play could, for example, offer a different learning style. Multiple plays are provided concurrently.

The following is an XML snippet of the method element:

```
<method>
  <play identifier="play1">
    <act identifier="firstact">
      <title>ACT1: VERSAILLES OVERVIEW</title>
      <role-part identifier="RolePart1">
        <title>Title rolepart 1</title>
        <role-ref ref="All"/>
        <learning-activity-ref ref="Versailles_Overview"/>
      </role-part>
      <complete-act>
        <when-role-part-completed ref="RolePart1"/>
      </complete-act>
    </act>
    <act>
      <title>ACT2: INTRODUCTION TO PREPARATORY PHASE</title>
        <role-part identifier="RolePart3">
          <role-ref ref="Learner"/>
          <learning-activity-ref ref="Preparation_Intro"/>
        </role-part>
        <role-part identifier="lastrolepartact2">
          <role-ref ref="Support_Staff"/>
          <support-activity-ref ref="Support_Preparation_Intro"/>
        </role-part>
        <complete-act>
          <when-role-part-completed ref="lastrolepartact2"/>
        </complete-act>
    </act>
    <complete-play>
      <when-last-act-completed/>
    </complete-play>
  </play>
</method>
```

The example above shows a play with two acts. The first act contains one role part. This act is complete when all members of the ‘All’ role have completed the ‘Versailles_Overview’ learning activity. As soon as this act has been completed, the next act will become available. Members of the ‘Learner’ role can perform the learning activity ‘Preparation_Intro’, while simultaneously, members of the role ‘Support_Staff’ can perform the support activity ‘Support_Preparation_Intro’. The act is complete when the last role part has been completed by all members in that role. Finally, the play is complete when the last act has been completed.

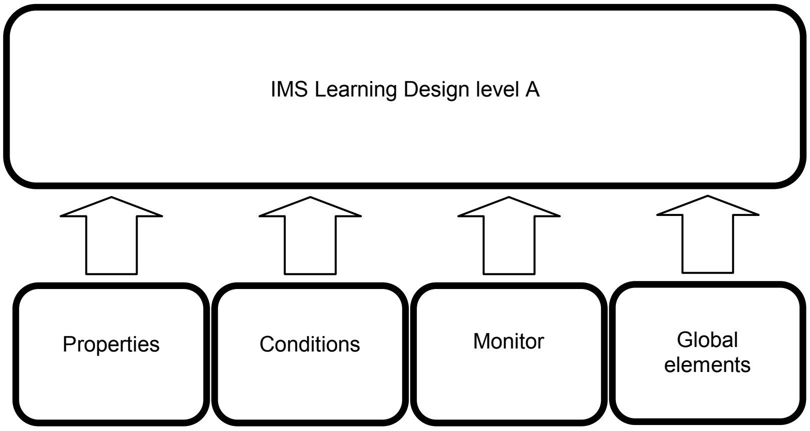


Figure 2.2 IMS Learning Design level B add-ons

With LD level A it is possible to model multi-role and multi-user learning designs. However, the possibilities for designing runtime personalization are still rather limited. Level B adds four new concepts to the LD core, as depicted in figure 2.2 that enable runtime personalization.

PROPERTIES

Properties are similar to variables as defined in common programming languages such as C or Basic. They have a unique identifier and are of a certain data type. LD recognizes the following property data types: *string, boolean, integer, real, uri, datetime, file, text* and *duration*. Each property is capable of holding a single value corresponding to its type. Furthermore, each has to be declared explicitly before it can be used. Valid values can be limited by adding restrictions to each property declaration. The runtime environment is responsible for enforcing these restrictions, which can be used, for example, to limit user input. Each property can optionally be seeded with an initial value.

Unlike most programming languages, properties are persistent beyond the lifetime of a learning design runtime session. The runtime environment is expected to ensure this persistence. Properties also have an instantiation scope, determining how many occurrences of a property should be created during runtime. The following scopes are defined in LD: *local*, *local personal*, *local role*, *global personal* and finally *global*. These scopes determine when a new property should be instantiated. For example, a local personal property will be instantiated for each learning design user. The scopes are closely related to the repeated deployment of a learning design during runtime; this deployment instance is called a *run* (IMSLD-BPG, 2003; Tattersall et al., 2005a). A run provides a context for assigning users to the roles of a learning design. An LD runtime environment should be capable of deploying multiple runs of the same learning design where each run provides a context for locally scoped properties. Table 2.1 depicts the relationship between the values of the property scope attribute and their instance occurrences.

Table 2.1 Relationship between property scope and their instantiations

Property scope	Occurrence
Local	One for each run
Local personal	One for each user in a run
Local role	One for each role instance[1] in a run
Global personal	One for each user
Global	One instance only

[1] Additional role instances can be created during runtime for some roles. These new role instances also lead to the creation of additional role property instances where appropriate.

Properties can be grouped together, which allows them to be addressed by a single reference. The following XML snippet defines some properties and combines them into a property group.

```
<properties>
  <globpers-property identifier="email">
    <global-definition uri="http://coppercore.org/email-new">
    <title>Email address</title>
    <datatype datatype="string"/>
   </global-definition>
  </globpers-property>

  <globpers-property identifier="username">
    <global-definition uri="http://coppercore.org/username-new">
      <title>User name</title>
      <datatype datatype="string"/>
    </global-definition>
  </globpers-property>

  <locpers-property identifier="prop1">
    <datatype datatype="real"/>
    <initial-value>12.50</initial-value>
    <restriction restriction-type="maxInclusive">12.50</restriction>
    <restriction restriction-type="fractionDigits">2</restriction>
   </locpers-property>

  <property-group identifier="group1">
    <title>Group 1</title>
    <property-ref ref="email"/>
    <property-ref ref="username"/>
    <property-ref ref="prop1"/>
  </property-group>
</properties>
```

The snippet above declares two global personal properties containing an email address and a user name. These properties will be instantiated once for each user. The snippet also depicts the declaration of a local property of type *real* with a maximum of 12.5. When viewing this property, two fraction digits are rendered. It will be instantiated for every user and run combo. Finally, the snippet also demonstrates how these properties can be grouped together to allow easy reference to them via the group ID.

GLOBAL ELEMENTS

Property values can be manipulated by two other level B constructs: global elements and conditions. Global elements are XML constructs that extend the W3C XHTML specification. Four global elements are defined: *set property*, *get property*, *set property group* and *get property group*. These constructs should be rendered by the runtime environment as either entry fields providing the possibility to change the property value, or text fields showing the properties' value. The runtime environment should ensure that restrictions defined for these properties are respected when a user enters data for them. The property elements in the global content have an attribute that determines whether this property refers to the user's property or to that of a user being supported. The support context is determined by the LD element that refers to this global content. Support activities and the monitor objects provide these support contexts by referring to the roles that are should be supported or monitored. It is up to the runtime to provide a mechanism that allows the selection of a specific user from the total list of users being supported or monitored. The set property element also has an attribute that limits the number of times a property may be set by a user.

LD uses the IMS content package specification (CP) (IMSCP-IM, 2003) as means for packaging the learning design and the associated resource together. This packaging is discussed in more detail later, but for the time being it can be thought of as a zip file containing all resource files as well as the learning design itself. The LD item model was informed by and based on CP. It binds resources to the learning design; each item is associated with a resource element which either links to one of the files in the content package or to an external resource via an absolute URL. Resource elements have type attributes that determines the kind of resource referenced. LD supports two resource types: ‘webcontent’ and ‘imsldcontent’. Whenever a resource contains global elements, the type of resource should be set to ‘imsldcontent’. This triggers the runtime to parse this resource for global elements and render the content accordingly. We will see in chapter 5 that we extend the supported resource types to enable the integration of other specifications and learning services.

The following XML snippet depicts a resource with one item that has a reference to a resource with global content.

```
<learning-object identifier="review_student_lo">
       <item identifierref="review_student_res"/>
</learning-object>
..
..
<resource identifier="RES review_student_res" type="imsldcontent"
         href="review_student.xml">
</resource>
```

The following XML snippet is the actual content of the “review_student.xml” file referenced by the resource in the previous snippet. Because the resource type is defined as ‘imsldcontent’, the content must be rendered in a special manner. The value of ‘datetimestarted’ property for a supported person must be rendered, and user input control should be generated for properties ‘test2’ and ‘test3’.

```
<?xml version="1.0"?>
<html xmlns:imsld="http://www.imsglobal.org/xsd/imsld_v1p0"
      xmlns="http://www.w3.org/1999/xhtml">
<head><title>Example of global content</title></head>
<body>
<p>When did the user start?</p>
<imsld:view-property ref="datetimestarted" property-of="supported-person"
view="title-value"/>

<p>Enter your thoughts below</p>
<imsld:set-property ref="thoughts" property-of="self" view="title-value">
</imsld:set-property>
```

```
<p>Should the role property be visible?</p>
<imsld:set-property ref="test2" property-of="self" view="title-value"/>
<p>Should the role property be visible with a collapse and expand
control?</p>
<imsld:set-property ref="test3" property-of="self" view="title-value"/>
</body></html>
```

CONDITIONS

The second construct for manipulating property values, besides the global elements, are conditions. These conditions consist of an antecedent and a consequence. In the condition “if X then Y”, the “if X” part is the antecedent and “then Y” is the consequence. These conditions can be compared with those found in programming languages. What sets them apart from most programming languages, however, is the fact that the conditions are not imperative, meaning that the order of the evaluation is not determined by the order in which they are entered in the learning design. Instead, they resemble the production rules of a production system. Their antecedents must be continuously monitored by a runtime system to determine when to evaluate the consequences. A consequence might be an instruction to show or hide IMD LD constructs such as activities, items, environments or parts of the global content. It might also be the manipulation of property values: the change of a property value could cause antecedents of one or more other conditions to evaluate to true. This results in the execution of their consequences, and so on. The runtime must pay special attention to avoid ending up in an infinite loop. It resembles a production system but with a twist – it processes the ripple effect caused by events and consequences, rather than attempting to find a solution via forward or backward reasoning.

Both the antecedent and the consequence make use of expressions built around an operator and zero or more operands. An operator can act also as an operand for another operator. LD contains a number of Boolean operators such as *and*, *or*, *is*, *not*, *greater than* and *less than.* These Boolean operators are typically the root for antecedent expressions. For the consequence, additional operators such as *sum*, *subtract*, *multiply*, *divide* and *no-value* are available. Besides these generic operators, LD-specific operators include *member of role*, *date time activity started* and *time unit of learning started*. All expressions use the Polish[2] notation, also known as prefix notation. This is most natural notation to use when representing expressions in a hierarchically structured language such as XML.

The following XML code snippet illustrates some example conditions.

```
</conditions>
  <if>
    <and>
      <not><no-value><property-ref ref="result"/></no-value></not>
```

[2] Invented by Polish logician Jan Łukasiewicz.

```
      <greater-than>
        <current-datetime/><property-value>2003</property-value>
      </greater-than>
    </and>
  </if>
  <then>
    <change-property-value>
       <property-ref ref="result"/>
      <property-value>
        <calculate>
          <sum>
            <property-ref ref="result"/>
            <property-value>1</property-value>
          </sum>
        </calculate>
      </property-value>
    </change-property-value>
  </then>

  <if>
    <is-member-of-role ref="learner"/>
  </if>
  <then>
    <show>
      <class class="calculate"/>
    </show>
  </then>
  <else>
    <hide>
      <class class="calculate"/>
    </hide>
  </else>
</conditions>
```

The antecedent of the first condition is fired when the property ‘result’ has a value and the current date time is after 2003; the consequence of the first condition states that the value of a property ‘result’ will be set to the sum of the value of property ‘result’ and 1. The second condition states that all XHTML with class attribute ‘calculate’ will be visible whenever the user is a member of the learner role. Otherwise, these elements will be hidden.

MONITOR

The monitor element, finally, is an additional service added by LD level B. It allows property values to be viewed or set for members of a specified role, or for the current user. Which properties should be viewed is defined by items referring to resources of type ‘imsldcontent’. Thus the monitor element itself provides the context for any property references, and the ‘imsldcontent’ resource type defines which properties must be shown or set. The next code snippet is an example of the monitor element stating that members of the role ‘Learner’ will be monitored. The ‘imsldcontent’ resource ‘portfolio.xml’ declares which properties are monitored.

```
<service identifier="portfolio_view">
  <monitor>
    <role-ref ref="Learner"/>
    <item identifierref="portfolio_res"/>
  </monitor>
</service>
..
..
<resource identifier="portfolio_res" type="imsldcontent"
          href="portfolio.xml">
</resource>
```

The content of the ‘portfolio.xml’ is depicted below:

```
<?xml version="1.0"?>
<html xmlns:imsld="http://www.imsglobal.org/xsd/imsld_v1p0"
      xmlns="http://www.w3.org/1999/xhtml">
<head><title>Portfolio</title></head>
<body>
<p>The portfolio data are:</p>
<imsld:view-property ref="document_for_activitiy_a" property-
of="supported-person" view="title-value"/>
<p />
<imsld:view-property ref="document_for_activitiy_b" property-
of="supported-person" view="title-value"/>
</body></html>
```

The XHTML document states that two properties, ‘document_for_activity_a’ and ‘document_for_activity_b’, will be examined.

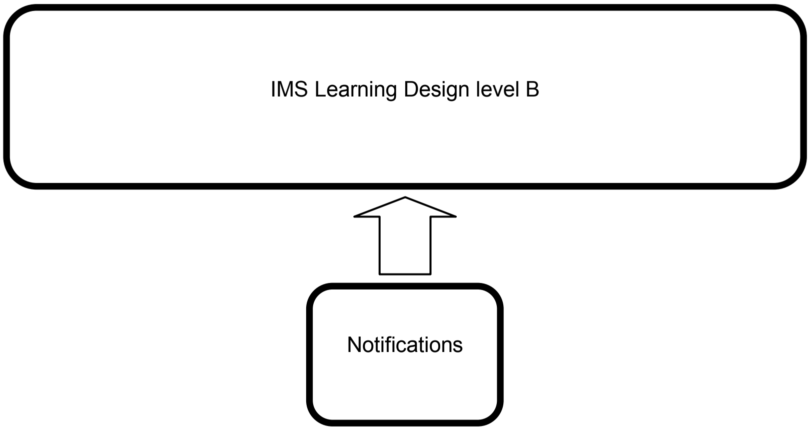


Figure 2.3 IMS Learning Design level C

Figure 2.3 depicts the notification extension contributed by LD level C. Notifications alert users of events that have occurred during runtime. They can be triggered by the completion of learning activities, support activities, acts, plays and the learning design itself. They consist of a message sent to all notification receivers. There is the possibility to attach a learning activity or support activity to the notification message; the runtime must ensure that this activity is visible and accessible by the notification receiver. This effectively overrules any other visibility rules defined for this activity. The following XML code snippet depicts an example of a notification triggered as part of the consequence of a condition.

```
<if>
  <and>
    <not><no-value><property-ref ref="value1"/></no-value></not>
     <is>
       <property-ref ref="operator"/>
       <property-value>divide</property-value>
     </is>
  </and>
</if>

<then>
  <change-property-value>
    <property-ref ref="result"/>
    <property-value>
      <calculate>
        <divide>
          <property-ref ref="value1"/>
          <property-ref ref="value2"/>
        </divide>
      </calculate>
    </property-value>
  </change-property-value>
  <notification>
    <email-data username-property-ref="username"
                email-property-ref="email">
      <role-ref ref="Learner"/>
    </email-data>
    <learning-activity-ref ref="Review-calculation"/>
    <subject>Result has been processed</subject>
  </notification>
</then>
```

The XML snippet above defines that every user in the role ‘Learner’ will receive a notification when property ‘value1’ has a value and property ‘operator’ equals ‘divide’. Furthermore, the learning activity ‘Review-calculation’ will be made available to these users.

For each of the major LD building blocks described in this section, more detailed information can be found in the LD information model.

IMS Learning Design binding and packaging

In the previous section we have seen several examples of XML snippets. These snippets represent valid IMS Learning Design according to a particular binding. The binding defines LD’s formal grammar. In theory, there could be many bindings for the specification, but in practice only one is currently defined. XML schema (W3C, 2007) is used as a formalism for describing the grammar and thereby also the binding. For LD, three incremental XML schemas have been defined. This makes it relatively easy to validate whether a learning design is lexically level A, B or C compliant, but XML schemas’ limitations make it impossible to validate the learning design completely. Additional validation steps are required to ensure the correctness of the design.

Besides defining a grammar, LD also specifies how a learning design should be packaged. The package is called a Unit of Learning (UOL): a complete, self-contained unit of education or training such as a course, module, lesson, etc. LD recommends the use of CP for this purpose. We have already seen in the previous section that the item model used by LD is informed by CP. A content package consists of a manifest and associated resources; the manifest contains one or more organizations, which describe how the resources are structured. In the case of a UOL, the manifest is a learning design. A CP – and therefore also a UOL – is often zipped into a single file for ease of use, although this is not obligatory.

Figure 2.4 is a graphical representation of the structure of a UOL. It shows the UOL containing a manifest that has metadata, a learning design and resources. These resources may refer to one of the physical files included with the UOL.

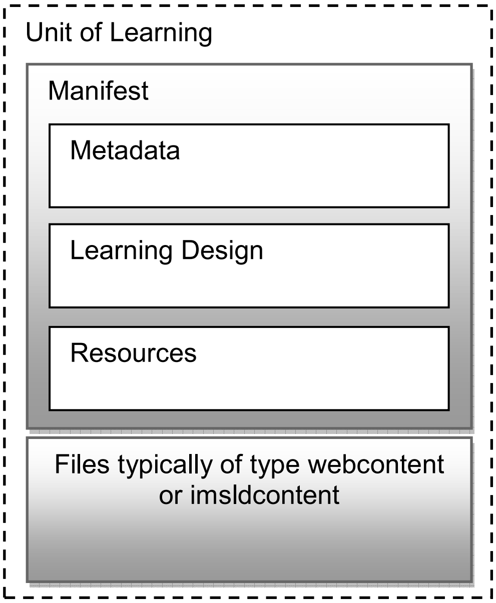


Figure 2.4 The structure of a Unit of Learning

More practical details about LD and example UOLs can be found on the learning networks site on LD at http://imsld.learningnetworks.org/.

Runtime considerations

The best practices and implementation guide (IMSLD-BPG, 2003) provides some insights into the main requirements for the implementation of a runtime environment for LD. In this section we present an overview of these requirements. Figure 2.5 depicts a high-level view of the main components of an LD runtime implementation as set out in the guide.

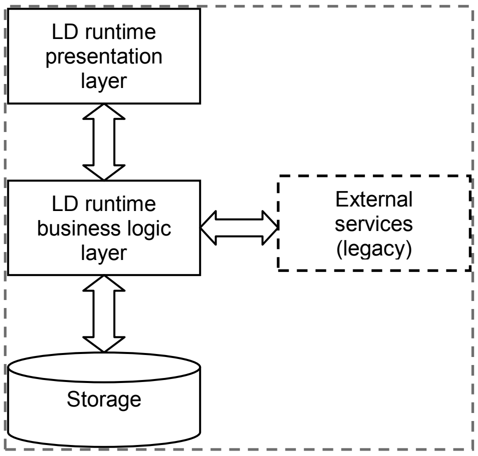

LD runtime presentation layer
LD runtime business logic layer
External services (legacy)
Storage

Figure 2.5 High level view on IMS Learning Design runtime

This figure depicts a data store taking care of the runtime persistence. This includes, amongst others, the persistence of property values, run memberships and role memberships. Business logic rules as defined by LD and the specific UOL are applied to the data persisted in the storage. The outcomes of applying these rules are rendered to the user via the presentation layer. We will call the layer responsible for applying the business rules the *LD engine*, or *engine* for short. The layer responsible for rendering the outcomes of this engine we will call the *LD player*, or *player*. Finally, external services can connect to the business logic layer for tight integration into existing learning management systems. The type of services integrated can range from educational services, such as forums, to administrative services like user registration. This thesis focuses on the design and development of the LD engine and the integration of services and other specifications. Although we also provide an LD player as a reference, we do not deal with the specifics of the player and its user interface aspects.

From the detailed descriptions of LD above, we can derive the following main categories of runtime requirements.

VALIDATION

A reusable engine cannot make assumptions about the source of the UOL. A UOL can be created via an authoring environment that only produces valid UOLs, but it can also be created with a text editor and zip utility. The latter is, of course, error prone. Therefore, every UOL needs to be validated. This validation provides authors with useful feedback about mistakes they might have made in their designs, and ensures that no unexpected or undesired behavior will occur during runtime. Validation of the UOL involves checking the correctness of the zip file, the lexical validity of the XML files using the available schemas, referential integrity within the XML files and the resources of the UOL for completeness.

Next to this technical validation, the learning design requires a semantic validation for those constructs that cannot be expressed by an XML schema. An example would be validating the constraints defined for the role element. Validation should ensure that the minimum number of persons assigned to a role is fewer than or equal to the maximum. A more complex example of semantic validation deals with the typecasting of properties. We have seen that properties have data types and that expressions allow the property values to be changed. These assignments can require type-implicit type conversions, also known as coercions. LD does not clearly specify how to deal with these coercions. Two approaches are possible when determining how the coercions need to be validated: runtime dynamic coercion and parse time static coercion. The first attempts the typecasting 'just in time' during runtime. The advantage is that typecasting possibilities are maximized because the property values are used to determine whether the typecast is possible. For example, value '1' can be converted to an integer, whilst 'one' cannot. The drawback is that runtime errors can occur because conversion is not possible. The second coercion type is more restrictive, and determined during parse time. The parser determines the typecasting based on the data types of the expression, and not on the actual property value. Here the cast from string to integer will not be allowed. The drawback is that far fewer expressions are allowed; the advantage, however, is that these errors occur during parsing and not during runtime. The CopperCore engine implementation that we present in this thesis uses parse-time coercion to avoid unexpected behavior during runtime.

Some validation can only be done during runtime. We have seen that properties can have value constraints; the engine should enforce these constraints by validating the values set for these properties. We have also seen role assignment constraints that should be enforced by the engine. How to deal with these runtime constraints is up to the LD player implementation. A player could duplicate the validation mechanisms, preventing the user from entering invalid data. Alternatively, a player could rely on the engine for reporting back errors. The latter is simpler, but far less user friendly. The engine has to ensure that both types of validation are possible, and must therefore provide the player with sufficient data to allow preemptive validation.

Parsing

The UOL needs to be parsed and the defined learning flow interpreted. From the learning design method section within the UOL, business rules need to be extracted and executed during runtime. Executing these rules will result in a personalized view of the learning design for each user. Interpreting the rules involves correct interpretation of the completion rules and timely interpretation of the conditions. It is likely that state data of multiple users is involved when evaluating these business rules. Parsing can be handled in many different manners: it could, in theory, be done just in time, taking the complete UOL, unpacking it and interpreting it for each and every request to the engine. This, clearly, is not the most efficient approach. It is more sensible to pre-parse the UOL and dissect it into more manageable parts that will be interpreted.

PUBLISHING

A UOL needs to be imported into the engine before it can be delivered during runtime. At a minimum, an engine needs to create a handle for accessing this UOL. It makes sense to precede the publishing by the validation of the UOL, preventing incorrect UOLs from being deployed. Furthermore, the publishing provides a good starting point for the pre-parsing and pre-processing of the UOL.

How to deal with UOL republications is a challenge for which several implementation strategies are possible. An engine could disallow republication, thereby avoiding any of the associated problems. Alternatively, it could allow republication but automatically create an additional handle for the republished UOL – thus, one for the old publication and one for the new. In effect, this means that both UOLs will run side by side. An engine could also override the old publication, which has the advantage that existing runs are also updated, thus allowing corrections of design errors after initial delivery. The disadvantage, however, is that a number of runtime problems could occur because, for example, properties have been added, removed or changed in the new design. This could leave an engine in an undefined or unexpected state. In the CopperCore engine discussed in this thesis, we allow the overwriting of a UOL by a new version because we feel the advantages outweigh the disadvantages.

PROVISIONING

The learning design refers to resources packaged within the UOL. The runtime environment has to ensure that these resources are available for rendering on the user's computer. Furthermore, it must ensure that the UOL itself is provisioned. For this purpose, the UOL must be instantiated. An instance of a UOL is known as a run; each run has its own identity and lifetime cycle, users are enrolled for a run, and all population is done in the context of a run. Users may be assigned to multiple runs of the same UOL but with, for example, different role assignments in each run. The runtime should provide the means to create and manage these runs.

POPULATION

We saw earlier that LD uses the role element as an artefact for representing users during design time. These placeholders must be populated with real users during runtime. Therefore, any runtime environment should provide the means to allocate real users to the roles whilst guarding the constraints placed upon them. Furthermore, it should be possible to create new instances of these roles during runtime if the learning design specifies that this is allowed. Each user will have an individual state for a UOL run. These states incorporate the user's progress as well as the values for each property defined in the UOL. They should be persisted and accessible by the runtime at all times, as LD coordinates multiple roles for multiple users. This requirement imposes limitations on the possible engine designs.

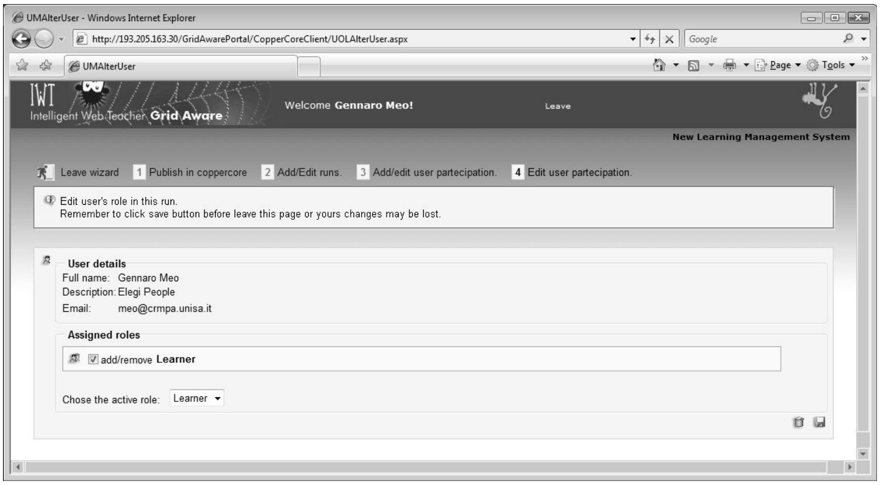
PERSONALIZATION

The engine must ensure that the content presented to the user is adapted according to rules defined in the UOL. We have seen how this personalization can be specified through roles, properties and conditions. Therefore, an engine needs the user, run and role identities to be able to personalize the UOL content for a user. These identities are essential when parsing the business rules and conditions defined by the UOL; therefore, they have to be passed to the engine as context when applying and evaluating the business rules and conditions.

INTEGRATION

The engine has to offer the means to interface with external services. These services are either defined in the learning design or needed for embedding the runtime into existing learning management systems. This connectivity can be very proprietary and closed when the engine is specifically designed to be integrated in a specific learning management system. For a reusable engine, however, it must be as open as possible to allow the engine to be used in various environments and circumstances.

Having briefly discussed the main categories of requirements for an LD engine, a natural, top-down workflow approach emerges (Westera, Brouns, Pannekeet, Janssen, & Manderveld, 2005). The learning design is authored first. Next, this design is published, and if validation does not report any errors, it can be populated by assigning users to runs and roles. The design may then require adaptations based on the outcome of the experience gathered during runtime. These adaptations result in a modified UOL, which can be republished.


UMAlterUser - Windows Internet Explorer
http://193.205.163.30/GridAwarePortal/CopperCoreClient/UOLAlterUser.aspx
UMAlterUser
IWT Intelligent Web Teacher Grid Aware
Welcome Gennaro Meo!
Leave
New Learning Management System
Leave wizard
1 Publish in coppercore
2 Add/Edit runs.
3 Add/edit user partecipation.
4 Edit user partecipation.

Edit user's role in this run.
Remember to click save button before leave this page or yours changes may be lost.

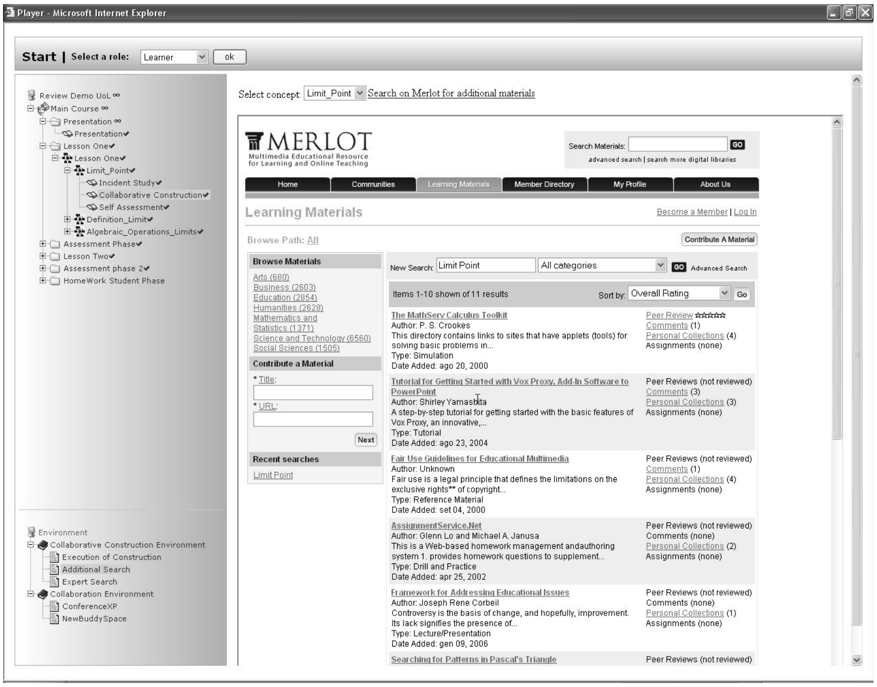
User details
Full name: Gennaro Meo
Description: Elegi People
Email: meo@crmpa.unisa.it

Assigned roles
☑ add/remove Learner

Chose the active role: Learner

Figure 2.6 Example screen shot of publishing process (source: ELeGI project)

Figure 2.6 depicts an example of a typical user interface for managing the publication and population processes. The resulting runtime experience is shown in figure 2.7. Both screenshots were taken from the ELeGI project (ELeGI, 2007), to be discussed in more detail in chapter 1.



Select a role: Learner

Select concept: Limit_Point

Search Materials:

Learning Materials

Browse Path: All

Browse Materials

Arts (680)
Business (2603)
Education (2854)
Humanities (2628)
Mathematics and Statistics (1371)
Science and Technology (6560)
Social Sciences (1505)

Contribute a Material

* Title:

* URL:

Recent searches

Limit Point

New Search: Limit Point | All categories

Items 1-10 shown of 11 results | Sort by: Overall Rating

The MathServ Calculus Toolkit Author: P. S. Crookes This directory contains links to sites that have applets (tools) for solving basic problems in... Type: Simulation Date Added: ago 20, 2000	Peer Review ☆☆☆☆☆ Comments (1) Personal Collections (4) Assignments (none)
Tutorial for Getting Started with Vox Proxy, Add-In Software to PowerPoint Author: Shirley Yamashita A step-by-step tutorial for getting started with the basic features of Vox Proxy, an innovative,... Type: Tutorial Date Added: ago 23, 2004	Peer Reviews (not reviewed) Comments (3) Personal Collections (3) Assignments (none)
Fair Use Guidelines for Educational Multimedia Author: Unknown Fair use is a legal principle that defines the limitations on the exclusive rights** of copyright... Type: Reference Material Date Added: set 04, 2000	Peer Reviews (not reviewed) Comments (1) Personal Collections (4) Assignments (none)
AssignmentService.Net Author: Glenn Lo and Michael A. Janusa This is a Web-based homework management andauthoring system 1. provides homework questions to supplement... Type: Drill and Practice Date Added: apr 25, 2002	Peer Reviews (not reviewed) Comments (none) Personal Collections (2) Assignments (none)
Framework for Addressing Educational Issues Author: Joseph Rene Corbeil Controversy is the basis of change, and hopefully, improvement. Its lack signifies the presence of... Type: Lecture/Presentation Date Added: gen 09, 2006	Peer Reviews (not reviewed) Comments (none) Personal Collections (1) Assignments (none)
Searching for Patterns in Pascal's Triangle	Peer Reviews (not reviewed)

Figure 2.7 Example of runtime experience (source: ELeGI project)

In the following chapters we discuss how an engine can be designed to elegantly meet the requirements. We describe how this engine can be reused in a variety of settings, and examine the integration of other services and specifications. We also investigate how to incorporate this engine into an environment, helping to overcome some issues with the top-down authoring process.

Chapter 3

Designing a Learning Design Engine as a Collection of Finite State Machines

Vogten, H, Tattersall, C., Koper, R., van Rosmalen, P., Brouns, F., Sloep, P., Van Bruggen, J. and Martens, H. (2006). Designing a Learning Design Engine as a Collection of Finite State Machines. International Journal on E-Learning, Vol 5(4), 641-661

Abstract

Specifications and standards for e-learning are becoming increasingly sophisticated and complex as they deal with the core of the learning process. Simple transformations are no longer adequate to successfully implement these latest specifications and standards for e-learning. IMS Learning Design (LD) (IMSLD-IM, 2003) is a representative of such a new specification in the field of e-learning. Its declarative nature, expressiveness and scope increase the complexity for any implementation. This probably is the largest hurdle that stands in the way of successful general deployment of this type of specification.

This article describes how an engine for interpreting LD can be designed as a collection of finite state machines (FSMs). An FSM is a computational model where a system is described through a finite number of states and their transition functions that map the change from one state to another. In the case of LD each state can be seen as constructed from a set of properties which can either be declared explicitly in LD or implicitly by the engine. State transitions are implemented through a mechanism of events and event handlers, completing the finite state machine. By reusing certain type of properties across FSMs it is possible to create an automatic propagation mechanism taking care of group dynamics without the need for any additional efforts. With the FSMs in place, personalization, one of the key features of LD, becomes a simple task. By combining the principles presented in the article, it becomes clear that an elegant design becomes feasible. This is demonstrated in the first actual implementation called CopperCore (Martens, Vogten, Van Rosmalen, & Koper, 2004).

Introduction

As open specifications (and standards) in e-learning are becoming more mature, their richness and complexity increases (IEEE, 2003; IMS, 2003). Early specifications dealt solely with meta-data. Later specifications focused on other, more complex educational processes. Good examples of such emerging new specifications, dealing with pedagogical frameworks, are IMS Simple Sequencing and IMS Learning Design. Implementation of these more complex specifications is not as straightforward. There is a need for additional guidelines to help developers incorporate these specifications into their e-learning systems. This article provides guidelines for implementers wanting to incorporate the IMS Learning Design (LD) specification into their products. The abbreviation LD is used when referring to the specification as laid down in IMS Learning Design (IMSLD-IM, 2003). The abbreviation UOL is used when referring to a learning design instance coded according to LD.

LD is used to specify the learning design of e-learning courses (so-called 'units of learning'). A unit of learning (UOL) is a package that consists of meta-data about the course, the learning design of the course and references to physical resources and/or the physical resources themselves (learning objects and learning services) that are used in the course. By providing a generic and flexible language, the LD specification supports the use of a wide range of pedagogies. It is based on a pedagogical meta-model (Koper & Manderveld, 2004; Koper & Olivier, 2004) supporting personalization of learning routes and reusability. The learning design specification is designed to allow for repetitive use in different situations with different persons and contexts.

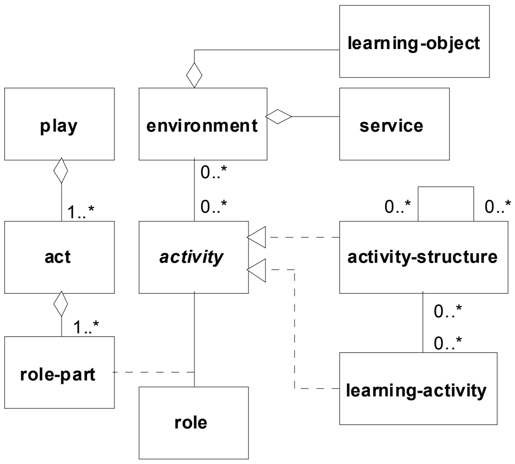


Figure 3.1 A UML (OMG, 2003) class diagram showing the core components of LD

A schematic overview of the core components and interrelationships is provided in figure 3.1. LD starts from the principle that a person is assigned to one or more learner or staff roles. So all references to users, be it learners or staff, are made through these roles and never on an individual (personal) basis. In a role, a person has to perform learning activities to attain specified learning-objectives. Activities can be combined into two types of activity-structures. First, an activity sequence by which the activities have to be performed in the order as specified in the structure. Second, an activity selection, by which a given number of activities may be selected from the number present in the selection. These activities are performed in an environment consisting of learning objects and learning services (communication, search, collaboration, etc.). The order in which activities have to be performed at all is specified per role. LD uses the metaphor of a theatrical play for this purpose. LD consists of one or more plays; a play consists of one or more sequential acts; an act consists of one or more concurrent role-parts. The role-part specifies the activity to be performed by a role when the act is started. The act synchronizes activities of the different roles over time. A role-part of the next act can only be accessed when the current act is completed. There are several conditional constructs that control the completion of an act, which allows the creation of cohorts of users working together. An example of this is the synchronization of tutors and learners by way of an act to ensure a sufficient number of tutors will be available when the learners start with their activities. Finally, the play sequences the acts in such a manner that it meets the learning objectives, given certain prerequisites.

LD also provides properties, conditions, and notifications to personalize learning designs, to enable more elaborate workflows and interactions based on user dossiers.
LD is implemented as an eXtensible Mark-up Language (XML) (W3C, 2003) binding. We assume the reader has good background knowledge of the major

constructs of LD. The full detailed specification of LD can be downloaded from the IMS website (http://www.imsglobal.org); (IMS, 2003), where also the XML bindings in the form of XML Schemas can be found. The LD specification is described at three levels. In this article we always refer to the most elaborate Level C.

LD is a declarative language. This means that it describes what behavior is expected by an implementation supporting LD without stating how this behavior should be achieved. Furthermore LD is an expressive language, which means that it has the ability to express a learning design in a clear, natural, intuitive and concise way, closest to the original problem formulation. Both LD's expressiveness and declarative nature make it ideal for its target audience of educational designers, but difficult for implementers because knowledge about the domain is required and implementation routes and strategies are not obvious.

The following XML code is an example of a small part of an LD instance.

Example 1: declaration of roles
```
<imsld:roles identifier="roles">
  <imsld:learner identifier="novice" min-persons="5"
                 max-persons="10">
    <imsld:title>Novice students</imsld:title>
  </imsld:learner>
  <imsld:learner identifier="advanced" min-persons="1"
                 max-persons="5" create-new="allowed">
    <imsld:title>Advanced students</imsld:title>
  </imsld:learner>
</imsld:roles>
```

The example code above demonstrates both the declarative nature of LD and its expressiveness. Notice that two roles are declared with attributes stating the minimum and maximum number of members for each defined role. For the second learner role it is possible to have *N* instances of this role during execution time due to the declaration of the create-new attribute. LD does not make any assumptions about how, when, and who should be assigned to these roles nor does it state how and when the mentioned constraints should be checked. It merely *declares* valid states.

Another example below shows how LD can express dynamic behavior in a very declarative manner.

Example 2: conditional completion of activity
```
<imsld:complete-act>
  <imsld:when-condition-true>
    <imsld:role-ref ref="tutor"/>
    <expression>
      <imsld:complete>
        <imsld:support-activity-ref ref="mark-assignment1"/>
      </imsld:complete>
    </expression>
  </imsld:when-condition-true>
</imsld:complete-act>
```

This example states that an act will be completed when all tutors have completed a certain support activity with id ‘mark-assignment1’. Apparently LD expects that the completion of activities will be tracked during run time (at least for the activity with id ‘mark-assignment1’) and that the activity is completed for all users in the role 'tutor'. Again, how this is achieved is left up to the implementers of the specification. LD merely specifies valid state transitions.

To produce the learning experience expressed by a UOL, a software component capable of interpreting this UOL is needed. This component is referred to as an ‘engine’. The output of an engine is a personalized version of the UOL according to all rules defined by LD. This article demonstrates how an engine can be designed with relative ease when approached from the perspective of a finite state machine (FSM) (Sipser, 1997). A finite state machine stores the state of a system at any given time. There are a finite number of states. The system may change from one state to another through transition functions. A set of rules working on certain input, the input alphabet determines which transition is performed. By extending the LD’s native property mechanism with new properties, each state is reflected by a set of properties. We will see that state transitions are realized through events and event handlers. With the FSM machine in place, execution of a UOL can be reduced to personalization of preparsed content. How the content is preparsed and persisted is part of what we call the publication process. Finally, we will see that personalization is a matter of a simple XML translation.

The engine as a collection of finite state machines

At the heart of LD are interactions, between users in particular roles or between users and the engine. The results of these interactions can be captured in properties. Properties can be explicitly declared in LD, but there are also properties in LD that are presupposed to exist. An example is a property that captures the completion status of an activity for every individual user. We will call these properties *implicit properties.* The following example shows three LD code fragments. The first fragment declares an explicit property. The second fragment shows that the learning-activity is considered as completed when the value for this explicit property is set. The last fragment shows how the following learning-activity is made visible depending on the completion state of the previous learning-activity. For this purpose the completion state is stored in an implicit property.

Example 3: explicit and implicit property
```
<!-- declaration of the explicit property containing the essay -->
<imsld:locpers-property identifier="essay">
  <imsld:title>Assignment 1</imsld:title>
  <imsld:datatype datatype="file"/>
</imsld:locpers-property>

<!-- create an essay -->
```

```
<imsld:learning-activity identifier="first_assignment" isvisible="true">
  <imsld:title>Assignment</imsld:title>
  <imsld:activity-description>
    <imsld:item identifierref="item1" isvisible="true"/>
  </imsld:activity-description>
  <imsld:complete-activity>
    <imsld:when-property-value-is-set>
      <imsld:property-ref ref="essay"/>
    </imsld:when-property-value-is-set>
  </imsld:complete-activity>
</imsld:learning-activity>

<!-- condition handling the visibility of the next assignment -->
  <imsld:if>
    <imsld:complete>
      <imsld:learning-activity-ref ref="first_assignment"/>
     </imsld:complete>
  </imsld:if>
  <imsld:then>
    <imsld:show>
      <imsld:learning-activity-ref ref="second_assignment" />
    </imsld:show>
  </imsld:then>
```

An FSM consists of a set of possible states, a start state, an input alphabet, a transition function and an output alphabet. A transition function is associated with an input symbol and causes the transition from the current state to a next state. A state change generates the output alphabet. Within the context of LD, the state of each individual user is represented by the set of values of all the properties that are either defined explicitly or implicitly by the learning design. As an engine has to deal with multiple users an engine is a collection of FSMs. FSMs offer a logical, methodical approach towards sequential input processing, that is relatively easy to design and implement and allows one to avoid error-prone conditional programming.

Properties are defined during a publication process. A UOL is parsed and analyzed by the engine during which all explicit and all needed implicit properties are defined and persisted in a database with individual values per user. These values represent the state of these users at any time. Execution of this UOL consists of personalizing the UOL for the user, which is in fact adapting the UOL according to the property values of this user. For example, a UOL can contain additional activities for novice users that are not required for more advanced users. During execution the UOL is personalized for every user depending on the value of a property holding their level of experience. A state represents the position of a user with respect to his or her progress in the UOL. The start state is defined by the initial values of the properties. These initial values are either given in the LD or are set as result of executing other UOLs at earlier stages. The input alphabet is made up of all LD constructs generating events and the transition functions are defined by LD constructs dealing with interactions. When, for example, the engine provides feedback when an activity is completed, the engine reacts to a user action, namely completing an activity. In terms of an FSM, this can be formulated as follows: the engine responds to a change of state that is caused by the user completing an activity. Example 3 translates into an FSM as follows. When the UOL is published properties are created for every user. There are at least two properties. First the explicit property 'essay', next the implicit property 'completion of activity first assignment'. Initially the value of the explicit property is null for all users because the essay has not been created yet and there was no initial value set. The value for the implicit property is set to 'uncompleted' by design. The input alphabet consists of the LD constructs 'upload an essay’ and ‘an essay has been uploaded.'. Transition functions are ‘set property value’, ‘complete activity’ and ‘show another activity’. Once a student creates and sends in an essay, the properties for this student are changed, while the properties for other students remain unchanged. So for that particular student, the activity is completed and the next activity is shown, while for other students the first activity can still be uncompleted and the second activity hidden.

There are a number of cases defined in LD where the change of state itself causes another change of state. A fairly obvious example is the LD construct *change-property-value* that can be triggered by the completion of an activity. In order to cope with these LD constructs when using an FSM, the definition of an FSM must be extended to allow each state to have an output that itself can be an input for the FSM. This type of final state machine is also known as a Moore machine (Sipser, 1997). By introducing this feedback loop, we should be able to deal with chains of state changes that can occur through several LD constructs.

The subsequent sections explain in depth how the concept of FSM is implemented in the engine. First the concepts of runs and roles are introduced; these concepts together with the user are the primary key to access a single FSM from the collection of FSMs. The next section shows how each state is persisted by the use of properties. A number of property types can be distinguished each with their own characteristics and use. The subsequent section deals with the transition function of the FSM. The concept of an event is introduced as the core of the input and output alphabets. It will become clear how the engine is capable of dealing with these events. Then we will return to the start of the process, explaining the importance of preprocessing the UOL. Finally, bringing all the previous concepts together, personalization will be shown to have become a straightforward XML transformation.

Populating the UOL

Before a UOL can be ‘executed’, users (learners, staff, etc.) have to be assigned to it. LD does not refer to users directly, but uses a proxy through roles for this purpose. It is the engine’s responsibility to bind actual users to abstract roles. A ‘run’ is introduced as a pedagogically neutral term for binding a group of users to a UOL by way of a publication.

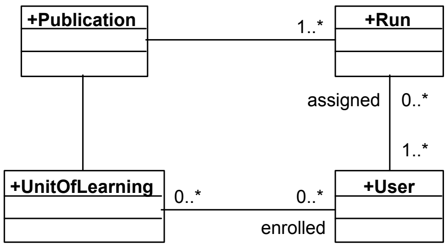


Figure 3.2 A run as an instance of a published unit of learning

Figure 3.2 depicts a UML class diagram of a run showing the run as intermediate between users that are enrolled for a UOL and a publication of this UOL. One or more users are assigned to each run, forming the community of users taking part in the UOL together at the same time. Users can enroll in a particular UOL and are assigned to one or more runs for the UOL. A run is assigned to exactly one publication, which in turn is associated with exactly one UOL. For each publication one or more runs may exist, allowing parallel execution of the same UOL. For now, it is sufficient to understand that a publication is the result of preprocessing a UOL so that it can easily be processed by the engine during execution of the UOL.

Runs provide a mechanism for binding users to the UOL, allowing at the same time multiple reuse of the same UOL, both sequentially and in parallel. Furthermore, it allows users to be grouped together in cohorts. However, individual users still must be mapped to the roles defined in the UOL. In order to satisfy this requirement two new constructs are introduced: 'role-participation' and 'run-participation'. Role-participation defines which roles a user may assume when participating in a run. Run-participation defines the active role for a user in a particular run at any specific moment in time.

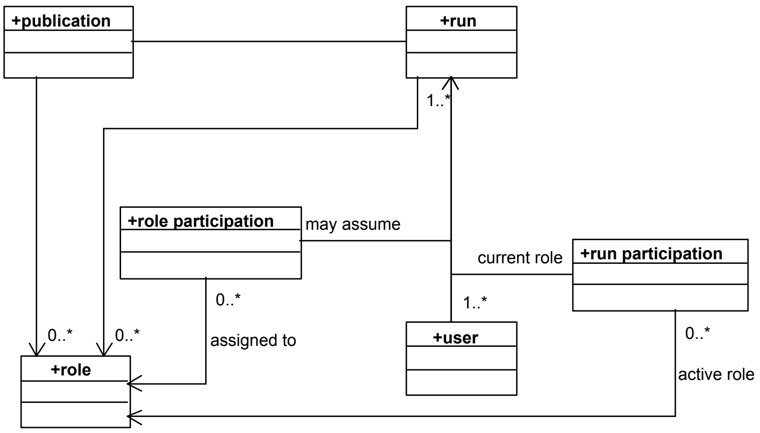


Figure 3.3 Relation between run and role

Figure 3.3 depicts the relationships in a UML class diagram. LD specifies that it is possible to have multiple instances for some roles and the figure shows that the allowable roles are associated with the publication as well as with the run. Role instances can be dynamically created during execution of the UOL as defined by LD. To be able to reuse a UOL, these newly created instances of the roles cannot be associated with the publication since they are different for each run. As a result, some of the roles are associated with the run and should be considered copies (or instances) of roles defined in the UOL. The difference between roles associated with the publication and those associated with the run is reflected in the way information about them is persisted. Information about roles associated with the publication is stored through global UOL properties whereas information about roles associated with the run is stored through local UOL properties. In the following section the difference between these types of properties is explained in more detail. In short, global UOL properties have the same value for all runs of the same UOL; however, local UOL properties can have different values for each run of the same UOL.

With the addition of role-participation and run-participation, all members of a particular role can be determined, thereby satisfying the last remaining requirement with regard to user population that is assigning individual users to roles.

How, why, when, and by whom users are assigned to roles is not part of the functionality of the engine. This is very much dependent on the business model of the party incorporating the engine and is considered to be out of scope for the engine. The engine, however, must provide interfaces allowing the manipulation of the model presented in figure 3.3 Relation between run and role. When doing so, the engine enforces the rules implied by both the model and the UOL preventing the system getting into a state not allowed by the UOL. Examples of such potential invalid states are role assignments to child roles without being assigned to the parent. Another example is the assignment to two roles which are declared to be mutual exclusive through the match-persons attribute on the role element.

We will see that the engine is a collection of FSMs and that the user, run, and role are the primary key when determining which FSM is being referred to at any point in time during execution. Before going into more detail, the property mechanism, which is essential when defining state, is discussed in the next section.

Properties

Properties represent data that need to be persisted. Each property consists of a property definition with one or more property values. The property can be either defined/declared directly, which makes it an *explicit property*, or can be presupposed which makes it an *implicit property*. The property definition determines the type, the default value, the scope, and owner of each property. Initial values are used as the initial state for the FSM. The scope of a property is either local, which means that it is bound to the context of a run or global, which means there is no direct relation with a run. The owner defines to whom or what a property belongs. The combination of scope and owner determines when and how properties are instantiated. The term 'instantiated' is informed by the world of object orientation. A property is instantiated when a new instance of a property, here a new persistent data store, is created according to its definition. The new property is assigned the initial property value of its corresponding property definition. The implicit value 'null' is assigned when no initial property value is defined. This is only needed for explicit properties as implicit properties always have an initial value which is set by the engine when creating this property.

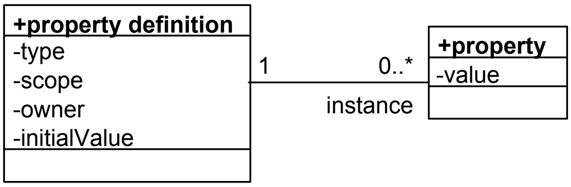


Figure 3.4 Property definition and properties

Figure 3.4 shows a UML class diagram of a property definition and its instantiated property. How and when properties should be instantiated is determined by the scope and owner. Table 3.1, shows valid combinations of scope and owner and describes the instantiation moment and the impact of this for the state.

Table 3.1 Property types per scope and owner

		Scope	
		Local	**Global**
Owner	**User**	A property is instantiated for every user for every run. Parallel runs can result in different states per run as the values may vary per run. Example: essay created, grade received.	A property is instantiated once for every user. This part of a user's state is the same for every run. Example: first name, surname, email address.
	UOL	A property is instantiated for each run. The property is a part of the state of all users of a run. Example: start date of the run; a url for a website, information about roles that are instantiated per run.	A property is instantiated for each UOL and is used for persisting results from the parser. This property is not part of anyone's state. Example: information about roles that do not have instances per run.
	Role	A property is instantiated for each role in each run. The property is part of the state for all the users in the group.	

	Scope	
	Local	**Global**
	Example: essay created together by all members of a role.	
None		A single property is instantiated once and typically contains information which is global for all UOLs and users. This property is not part of anyone's state. Example: general system parameters.

There are some interesting things to note in this table. It becomes apparent that there are different types of properties. Some properties are unique per individual, others for each individual in a run and yet others are common between groups of persons in a particular role or to individuals in a run. Note that scope and owner apply both to implicit and explicit properties.

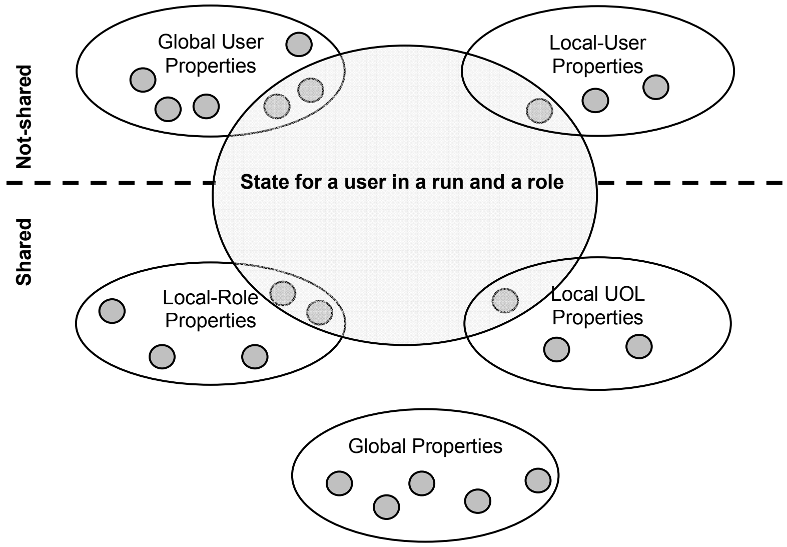


Figure 3.5 State a combination of sets of properties

Figure 3.5 shows how the different sets of properties make up the state for a particular user. Note that part of the state is shared amongst users and that a user can have more than one state at any moment in time if we take the perspective of the engine as a collection of FSMs. It becomes clear that the state is not purely related to the user, but also to the run and the role in which the user is participating. So, from the perspective of the engine as a collection of FSMs, the user, run, and role are the primary key for determining which FSM is being referred to at any point in time. The collection of all states for a user is also known as the user's dossier. Since the FSMs are for a part making use of the same properties, manipulating these properties propagates to all the FSMs involved. This also explains why the initial state for one FSM could be influenced by the final state of another FSM. This interlocking of FSMs provides a mechanism for dealing with group behavior in the engine.

It is important to understand that the engine is responsible for determining the scope and owner for each of the implicit properties it defines. In the Examples 2 and 3 at the beginning of this section it was mentioned that the engine is responsible for adding completed properties for a number of constructs. The engine is also responsible for determining what the ownership and scope of each of the completed properties should be. *Learning-activity, support-activity* but also *activity-structure, role-part, act, play,* and *unit-of-learning* are LD constructs for which the completion status needs to be recorded. The owner and scope for all these completed properties should be user and local. This is true for all except for the *unit-of-learning*. The completion of the *unit-of-learning* can be relevant beyond the run, for example in a curriculum, and its scope should therefore be global. These types of considerations should be made carefully for each implicit property that is introduced.

Another issue to notice in table 3.1 is that a new type of property, the global UOL property, has been added in addition to the ones that are defined in LD. This is a special category of properties, not known in LD, which is used by the engine to facilitate persistence of the parsing results during the preprocessing. Parsing converts the UOL into a format that can be easily interpreted during the personalization stage. The results of this parsing consist of XML documents derived from the original UOL. These XML documents are stored in global UOL properties. By doing so, the engine extends the use of properties as mechanism for persisting state for the FSM towards a more generic store. The extension allows an efficient implementation of the engine with minimal code and optimal reuse.

EVENT HANDLING

We have seen that properties provide the means for persisting state of a user (even multiple states). To complete the idea of FSMs we need a transition function that is capable of changing the state on the basis of an input alphabet. As noted earlier, the engine will be a Moore machine, making it necessary to have a mechanism that can react to a change of a state in the manner required by LD for some of its constructs. These reactions will form the output alphabet.

LD provides some instructions to let the user manipulate properties, and thereby state, directly. Examples are the *set-property* or *user-choice* instructions. However, most constructs change property values in a more indirect fashion.

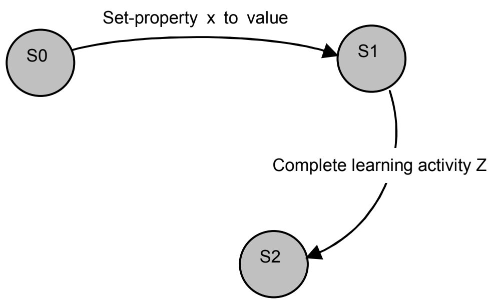


Figure 3.6 Example state diagram

Figure 3.6 shows an example FSM responding to the input alphabet. S0 represents the start state for the state machine for a particular user, run and role. The user interacts through the engine by manually setting a property and thereby changing state. The input is represented by the edge between S0 and S1. We assume that the UOL for which this state machine is drawn, contains a conditional construct that states that setting property x to value y should result in the completion of learning activity Z. The result of this output is state S2 and the output itself is represented by the edge between S1 and S2.

Obvious questions are: what are the alphabets and how can they be ‘read’ and ‘written’? The answer to the first question can be found by thinking of both alphabets in terms of events. Everything that can change the state of an FSM is considered to be an event and the collection of events thus forms the input alphabet of the FSM. The output alphabet consists of the input alphabet extended by additional events as a result of the LD semantics. An example of such an additional event can been seen in Example 3 where the activity is completed when the property essay has been set. This triggers the activity completed event, which becomes part of the input alphabet. The input and output alphabet will vary of course from one UOL to another as the properties defined in the UOL will differ and therefore also the potential events. Events can be classified into two classes: property events which, are triggered whenever a property value is changed and timer events, which are triggered after a defined duration of time.

The output alphabet can consist of events triggered on the basis of changed property values and a number of events that will not cause any state changes. Among the latter are events triggering *notifications* and *e-mail messages.* The remainder of this section deals with the implementation of the event processing mechanism in the engine.

Figure 3.7 shows the architecture of the event handling mechanism of the engine. The property store contains all states of all users. Whenever a property value is changed the property store raises a new event. This event is captured by the event dispatcher.

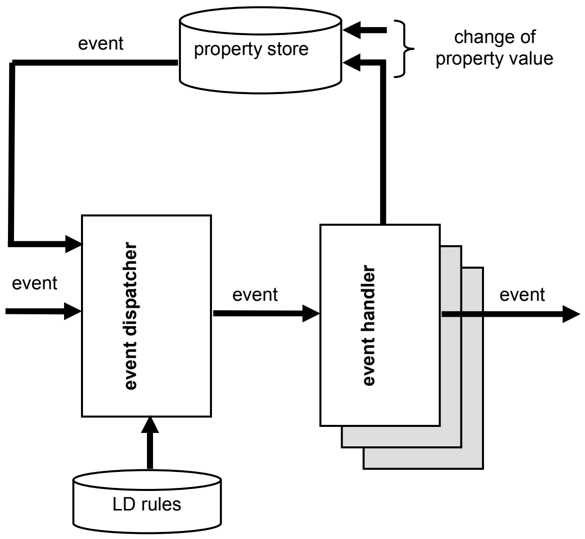


Figure 3.7 Overview of the event handling mechanism

The event dispatcher consults a store containing the rules defined by LD. This store is filled with information during the preprocessing of the UOLs. The event dispatcher requires this information to determine what needs to happen next. In most cases, no information is found in this rule store, meaning no further action is needed. However, on some occasions information is found, determining what the next step should be. Based on the information retrieved, the event dispatcher can determine which event handler to call. Each of the event handlers represents a type of LD rule. For example the LD rule stating the completion of the activity ‘first-assignment’ after the ‘essay’ property has been set in Example 3 is handled by such an event handler.

For LD quite a number of event handlers can be defined amongst which are handlers that process the completion of *unit-of-learning, act, play,* and *role-parts,* as well as handlers that deal with the conditional constructs in general. These event handlers react by changing one or more properties when certain conditions defined by the business rule in LD are fulfilled. This in turn causes one or more new events to be raised forming a chain of events. The event handlers do not necessarily react by changing property values. They may raise events triggering notifications or e-mail messages. Notice that an event can trigger zero, one or more event handlers, and that an event handler can change zero, one, or more properties. Furthermore, the change of properties can supersede the scope of a single FSM because the same properties can be shared amongst different FSMs. Therefore multiple FSMs can change state simultaneously as a result of a single event. An example is the last student who completes a learning-activity. This can cause the containing role-part to be completed for all users in that specific role. This characteristic ensures propagation and, as a result, the synchronization of different roles and groups working together. This propagation can occur within the perspective of a single user having multiple FSMs (one for every role the user may assume) or within the perspective of groups within a run or even at the level of the whole user community known to the engine. It is important to understand that in order for this mechanism to function properly state changes propagating over several FSMs are considered as atomic actions.

Timer events do not start with a change of a property value, but are raised by some timer. The rest of the event handling mechanism is exactly the same as for events raised through change of a property value. It is clear that there is a risk of recursion causing endless loops. It is the responsibility of the validation process during the preprocessing stage to detect these recursions.

Publication

A publication is the result of preprocessing a UOL. We have already seen that the properties and event handling mechanisms depend on the outcome of this process. The part of the engine responsible for this process is called the publication engine.

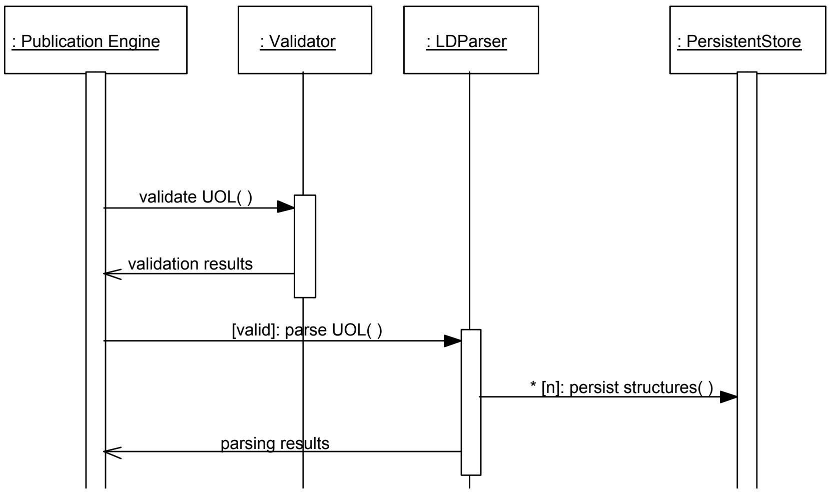


Figure 3.8 Publication process

Figure 3.8 shows a UML sequence diagram representing the publication process. The first step of the publication process is to check the UOL validity. Validation covers a numbers of aspects. The UOL is checked for completeness, that is, whether all locally referenced resources are also included in the UOL. The UOL is validated against the LD schema using a validating parser (for example *Xerces*). These types of validation are straightforward and revolve around XML technology. More interesting types of validation cover the semantics of a UOL. All references are checked to determine if no erroneous cross-references have been made. Examples of such errors would be a *role-ref* referring to a *property*. Another type of semantic validation includes the checks for invalid attribute values: for example, when the minimum number of persons in a role exceeds the maximum number of persons in a role. Recursions can occur whenever and wherever elements can include other elements by reference. The *environment* element is a good example of such a construct. Checking for recursion is especially important to prevent event handlers falling into endless loops.

If the validation is successful, the LD parser is invoked. The LD parser converts the LD into a format that can be easily interpreted during the execution phase. This intermediate XML format is used during the personalization stage. As noted earlier, global UOL properties are used to store these small XML documents. It is important to highlight that the actual resource is not part of such an XML document but is stored separately on a web server and is referenced from these XML documents.

Another important result of the parsing process is the store containing rules that should be applied to a UOL. The event dispatcher retrieves these entries to determine what actions need to be taken when an event occurs. Finally, the publication process is responsible for creating all property definitions both for the explicit and the implicit properties.

Personalization

A UOL is executed when a user in a specific *role* accesses a run of a UOL, which should result in an adapted view of the UOL according to this role and the user's property values. This adaptation process is known as personalization and is one of the core requirements of LD. Personalization involves adaptation of the LD according to rules defined by LD, which describe how the engine should react to certain states. An example is feedback, which only should be provided when the corresponding activity has been completed; in other words, when a certain state has been reached.

Another example is the personalization of the content. Table 3.2 shows the preparsed content for a monitor-object in the left column. The right column shows the result of the personalization. Note that the reference to the property has been replaced with its actual value.

Table 3.2 Example of Personalization

Preparsed XML content	Personalized XML content

`<body>`
`<h1>Monitor student progress</h1>`
`<strong>Score on essay</strong>`
`<imsld:view-property ref="score"`
`property-of="supported-person"`
`view="value"/>`
`</body>`
	`<body>`
	`<h1>Monitor student progress</h1>`
	`<strong>Score on essay</strong>`
	`<cc:view-property>`
	  passed
	`</cc:view-property>`
	`</body>`

Once the FSM is in place, personalization and therewith execution of LD becomes relative straightforward because the majority of the complexities are taken care of by the event handling mechanism.

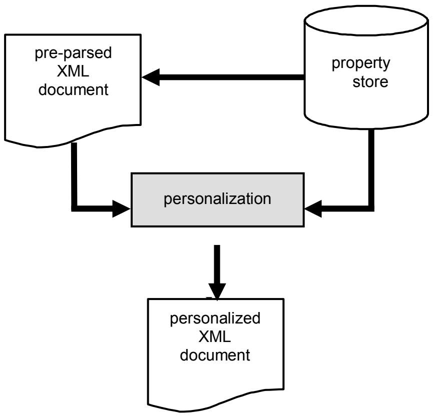


Figure 3.9 The personalization process

The result of the personalization process as shown in figure 3.9, is a personalized XML document. This is created by merging the XML document that was stored as a result of the publication, with the property values from the persistent property store. The exact method of merging the preparsed XML document with the property values varies, depending on the type of element and corresponding rules. The process results in the replacement, addition, or removal of some XML elements. A number of personalization types are defined in LD, which can be classified into the following three classes:

- Personalize the activity tree. An activity tree is the combination of all *play*s and their sub elements. The activity tree is personalized on the basis of the current FSM defined by the run and the current role of the user. Further personalization takes place on the basis of completed and visibility properties which were introduced earlier. The outcome is an XML representation of the activity tree reflecting the current status of the user.
- Personalize the environment tree associated with an activity. The environment tree is adapted using visibility properties in a similar way as is the activity tree, resulting is an XML representation of the activity tree reflecting the current status of the user.
- Personalize the content of various LD constructs. References to properties are replaced by their actual contents and parts of the content may be hidden on the basis of the value for the different class properties. Class properties are implicit properties created during publication which reflect the visibility status (hidden or visible) for classes of content.

In conclusion, it can be said that once the FSM mechanism is in place, personalization is reduced to a simple XML transformation that should obey the rules of LD.

Implementations

The Open University of the Netherlands developed the predecessor of LD, called EML (Hermans et al., 2004) in 1998. EML has very similar objectives to LD, although it is not an open specification and the actual tagging of the XML language is quite different. The consecutive versions of EML have resulted in a number of players. A first prototype was developed in 1999 as a proof of concept, followed shortly after by the first system, called Edubox which went into regular exploitation at the Open University of the Netherlands in September 2000.

Recently we implemented an open source LD engine with the name ‘CopperCore’, which was partly funded by the European Commission through the ALFANET (2004; Van Rosmalen et al., 2004) (IST-2001-33288) project. This engine was built using the design approach outlined in this article and has been made available as an open source product through SourceForge (http://sourceforge.net). The analysis and ideas presented in this article were based on previous experience with the implementations of the Edubox player and put into practice in the CopperCore engine. The first release supports the view that the approach presented in this article results in an elegant, lightweight design capable of supporting the complete LD specification.

Conclusions

With the arrival of the latest specifications and standards for e-learning, the sophistication, expressiveness, and complexity have increased considerably. Simple transformations are not adequate to implement these specifications and standards successfully. LD is a representative of such a new specification. Its declarative nature and expressiveness increases the complexity for any implementation. This is probably the largest obstacle that stands in the way of successful general deployment of this type of specification. Work needs to be done to help the community of implementers to overcome this hurdle.

In this article we have shown that by taking the approach of an FSM, it is possible to break down a complex specification like LD into a few basic constructs that allow elegant and relative lightweight designs and implementations. This breakdown is accomplished by exploiting the property mechanism beyond its direct usage in LD itself. The use of implicit properties helps harmonize the different kind of rules defined in LD, and reduces them to simple property operations. Furthermore the property mechanism acts as a store for the result of the publication process especially for the preparsed XML content. The event mechanism helps break down the large number of rules to their basics in the form of event handlers. Each of these event handlers have dedicated tasks that deal with different aspects of the rules as is laid down by LD, but all have the same basic mechanism. Again this helps to reduce the complexity enormously. Decomposition of the complexity is essential and is achieved by having implementers focus on the proper implementation of the event handlers themselves. Implementers of an event handler do not have to worry about the larger picture as it is dealt with by the event handling mechanism. The same event handling mechanism ensures that reactions to certain events are adequately propagated throughout the whole system. By doing so, all group and role dynamics are automatically incorporated into the engine without additional efforts as the engine is regarded to be a collection of FSMs. By the introduction of the run and the roles, it has become clear what should be considered as primary key for each of the FSMs. We have shown that by selecting the right owner and scope of the properties we can interlock the FSMs which results automatically in the correct propagation of state changes. Again no additional efforts have to be made because the event handling mechanism propagates state changes throughout all interlocked FSMs.

With these constructs in mind, implementation of an engine has not become simple, but far less complex than may have been anticipated at first sight.

Chapter 4

A Reference Implementation of a Learning Design Engine

Martens, H. & Vogten, H. (2005). A Reference Implementation of a Learning Design Engine. In: Koper, R. & Tattersall, C., Learning Design: A Handbook on Modelling and Delivering Networked Education and Training (pp. 91-108). Berlin-Heidelberg: Springer Verlag.

Summary

Since the release of LD there has been a need for a reference implementation of a player for the specification. CopperCore provides a way for implementers to jumpstart building an LD-compliant learning management system. It provides two major APIs to deal with the processing of LD. One covers administration-related tasks while the other deals with the runtime delivery of LD. CopperCore has been implemented using J2EE and the main components are implemented as Enterprise Java Beans. The use of J2EE allows a number of different implementation strategies giving developers the choice between a pure web-based approach and a dedicated native Java client. CopperCore is now readily available to all developers who wish to integrate LD support into their own software. It is released under the GNU General Public License and is available for free through SourceForge at http://www.coppercore.org.

Introduction

From the moment the Learning Design (LD) specification (IMSLD-IM, 2003; IMSLD-BPG, 2003; IMSLD-XB, 2003) was published there has been a need for software capable of processing LD-compliant content. LD is a powerful and complex specification, and it is not a trivial matter to implement an LD player. In response to this need, the Educational Technology Expertise Centre of the Open University of the Netherlands launched an initiative to develop a reusable kernel dealing with the intricacies of processing LD. Since this kernel should be able to be used in different settings, it is not a standalone product but needs to be integrated in a learning management system. The kernel, known as CopperCore, has been developed under the GNU General Public License and is available through SourceForge at http://www.coppercore.org.

CopperCore has the following features:

- A validation routine for the manifest file containing the LD ensuring only valid LD is processed. Validation includes both technical and semantic checks and the validation results are reported.
- An administrative backend with regard to publications, user management, runs and roles. These concepts are discussed below.
- Interpretation of LD and delivery of personalized content according to the rules defined in LD. This is achieved by keeping track of the user's progress and settings.
- Platform independence, based on a strategic choice for Java and J2EE (J2EE, 2007).

This chapter provides background information for implementing an LD-compliant player based on CopperCore. First a conceptual overview is given of the two major functional Application Programming Interfaces (APIs) dealing with administrative tasks and runtime delivery. The next section gives a brief technical overview of the architecture of CopperCore and discusses the technical design decisions. This helps the reader understand the final section dealing with implementation strategies.

Conceptual overview

In order to process LD successfully, CopperCore functionality has been divided into two major parts. The CourseManager handles administrative functionality such as users, roles, runs and publications. In contrast, the LDEngine forms the heart of CopperCore and deals with the runtime delivery of the personalized content as defined in the LD. Well-defined APIs are available for both parts to developers who wish to integrate CopperCore into their own products. The next section provides an overview of the functionalities found in the APIs.

CourseManager

The CourseManager deals with all administrative tasks required in order for the LDEngine to work. The CourseManager covers user management, role assignments, run management and publications. All these concepts are discussed next.

PUBLICATIONS

According to the LD specification a learning design needs to be packaged in a content package (IMSCP-IM, 2003) which is a ZIP file containing all resources. This content package must contain a file named `imsmanifest.xml` containing the learning design itself. All other files in the package are additional resources. A content package containing LD is called a Unit of Learning (UOL). Before a UOL can be deployed, CopperCore creates a publication for the UOL, taking care of several aspects needed during deployment.
First, the UOL is validated to make sure no syntactic or semantic errors are present in the package. Validation includes validation against schemas, validation of the package itself with regard to the resources included, and validation of semantics of the learning design. Detected errors are stored in a list of messages which can be reported back to the user.
Second, CopperCore breaks down the learning design into more manageable parts such as activities, environments, learning objects, roles, etc. Third, CopperCore analyses the roles that are declared in the learning design. This is necessary since users need to be assigned to particular roles before they can start using the system. Finally all content contained in the UOL is copied to a web server directory for retrieval during deployment.
Publishing a UOL can be done by simply calling an API method called `publishUOL`.

USER MANAGEMENT

LD focuses on delivering personalized education. This is achieved by describing a learning design through profiles using the role. CopperCore deals with this personalized delivery by creating a dossier for each user. In order to do so, CopperCore requires users to be defined. For this purpose a user may be added to CopperCore using the *createUser* API call. The only parameter passed is the user id. All other user information needed should be defined in LD as global personal properties and stored in a user's dossier. Once defined, users cannot be deleted.

RUN MANAGEMENT

LD may refer to all users in a role, i.e. a grouping of users. A grouping mechanism is required that allows the division of the user population into smaller cohorts working together in one particular learning design. A group could, for example, represent a classroom, or a number of students participating in a distance learning course. The term "run" is used in this context. Users are never assigned directly to a publication but they are enrolled for a particular learning design by adding them to a specific run. Therefore each publication must have at least a run. If necessary, more runs can be added depending on the particular circumstances. A new run can be created in CopperCore using the `createRun` API call passing the id of the publication as one of its parameters.

The next step is assigning the users to a particular run. As stated earlier, who should be assigned to which run depends very much on the circumstances. It is important to understand that only participants of the same run can cooperate and are "visible" to each other in the same learning design. So when LD refers to all users, in effect it refers to all the participants in a specific run. Users can be added to a run by calling the method `addUserToRun`. Users may be removed from a run by calling `removeUserFromRun`.

ROLE MANAGEMENT

Roles are the main personalization mechanisms of LD and are essential for creating different paths through a learning design. Roles may be seen as a representation of users with a certain profile. It is the task of role management to populate these roles with actual users of a run. Users can be assigned to a role using the method `addUserToRole` and can be removed using `removeUserFromRole`.

Different users can be assigned to different roles, but it is also possible to assign an individual user to multiple roles. However, when the LDEngine delivers the learning design to a user it personalizes the design using the role of the user. Therefore only one role may be active at any moment for each user. This role is called the active role. A user can switch roles at any time by selecting a new active role from the list of roles he or she is assigned to. The method `setActiveRole` sets the active role for a user.

LD defines a hierarchy of roles. This has an impact on the interpretation of the roles. A sub-role is considered to inherit all the properties of its ancestor roles. For example, a sub-role of the role "learner" will inherit the properties of this "learner" role and everything available to the "learner" is also available to its sub-role. CopperCore states that a user may only be assigned to a sub-role when the user is already assigned to the parent of that sub-role. The hierarchy of roles starts with a common root and all users must be assigned to this common root before doing any further role assignments.

LD supports the runtime creation of new roles. For example, if a role is used to group users together with a maximum of ten users, a new role may be created during runtime whenever this maximum is exceeded. In LD these roles have an attribute “`create-new`” with the value “`allowed`”. A new instance of a role can be created by calling the method `createRole`. Users can be assigned to these roles in the same way as with regular roles.

The UML class diagram of role and run model supported by CopperCore is that shown in figure 3.3.

LDEngine

After the UOL is published, users are assigned to the run and to their roles and the delivery of the learning design can start. LD defines a hierarchy of activities to be performed by a role in the method section. For each activity there are a number of resources, learning objects and services, grouped in an environment. Environments are also hierarchies.

CopperCore defines a number of concepts and API calls for retrieving the information contained in these hierarchies which are discussed in detail in the following sections.

ACTIVITY TREE

An activity tree is an XML representation of the method section of LD personalized for a user. Personalization consists of two parts. First, the active role of the user requesting the activity tree is taken into account. Only those activities associated with the active role, or one of its parent roles, will be included in the activity tree. CopperCore deals with this personalization during the publication stage by splitting the method hierarchy up into a number of smaller hierarchies based on the defined roles, using the role-part constructs in LD.

Second, personalization deals with the individual progress of users. This mainly involves keeping track of the completed activities for a user. CopperCore deals with all defined rules in LD, such as the completion of activity structures, acts, plays and the unit of learning. The resulting XML tree is based on the application of these rules on a personal basis. A personalized activity tree can be retrieved by calling the method *getActivityTree*. This method is called in the context of a user in a specific run and returns an XML representation of the activity tree for this user. A visual representation of the underlying schema of this XML response (an activity tree schema) is shown in figure 4.1.

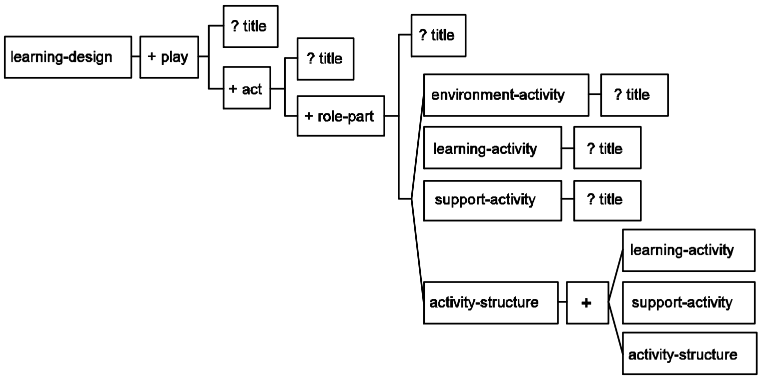


Figure 4.1 Activity tree schema

The schema closely resembles the original LD. However, there are some differences, especially when reflecting the user's progress. The root element of the activity tree is the learning design itself. It contains one or more plays. A play contains one or more acts and an act is made up of role parts. A role part itself contains an activity which is a `learning-activity`, `support-activity`, `activity-structure` or an `environment-activity`. The last is not an activity as such but represents an environment with an implicit activity, such as an activity that instructs the learner to read the documents in the environment. Each of the elements may contain a title which can be used in the user interface when representing a node of the activity tree.

The activity tree contains only those nodes available to the user at the moment of retrieval, which is a major difference from the original learning design containing all potential nodes for all users. This filtering of nodes is only one result of the personalization. Another aspect of the personalization can be seen when examining the attributes of the nodes. Table 4.1 describes each of the attributes.

Table 4.1 Activity tree node attributes

Attribute	Description
`completed`	This attribute may have the value true, false or unlimited. The attribute indicates if a user has completed the node or, if the value is unlimited, that the node should be considered completed. The following nodes have a completed attribute: `act`, `activity-structure`, `environment-activity`, `learning-activity`, `learning-design`, `play`, `support-activity`.

`environment`	This attribute contains a space-separated list of ids belonging to environments of the activity represented by the node. The values of this attribute should be passed when retrieving the environment via the `getEnvironmentTree` API call. This attribute is used in the following elements: `activity-structure`, `learning-activity`, `support-activity`, `environment-activity`.
`identifier`	This attribute is the identifier of the object represented by the node. Note that this is not the identifier of the node itself and therefore multiple nodes may have the same identifier value if they are pointing to the same object. This identifier should be used when retrieving the content of the object represented by the node via the `getContent` API call. The identifier attribute is used in `activity-structure`, `environment-activity`, `learning-activity`, `learning-design`, `play`, `role-part`, `support-activity`.
`isvisible`	This attribute indicates if a node is visible for the user or not. For Level A it means that this value is identical to the value defined initially in the learning design because there are no constructs allowing the value to be changed. The attribute may occur in `learning-activity`, `play`, `support-activity`.
`role`	This attribute contains the role name which was the basis for generating this activity tree. The attribute occurs only in the `learning-design` node.
`structure-type`	This attribute can have the values sequence or selection indicating which type of activity structure is represented by the activity-structure node in which the attribute occurs.
`time-limit`	This attribute indicates that the completion of a node is dependent on a timed event. It occurs in an act, learning-activity, play, support-activity.
`user-choice`	This attributes indicates that a user must indicate when an activity has been completed. There should be a means in the user interface allowing for this. When a user indicates completion of the activity, `completeActivity` should be called. The attribute may occur in `learning-activity`, `support-activity`.

ENVIRONMENT TREE

An environment tree is a representation of the environment and the learning objects and services belonging to one or more activities. The environment tree may be retrieved by calling `getEnvironmentTree` which results in an XML document according to the schema shown in figure 4.2. The root element is `environments` which can contain one or more environments. An environment consists of zero or more learning object, environments and services. There are three types of services: send-mail, conference and index search. `Send-mail` contains the `send-to` element representing the recipients of the mail and the `from` element representing the sender of the mail. The content of the title element should be used to represent a node in the user interface. In LD Level A there is no personalization of the environment tree.

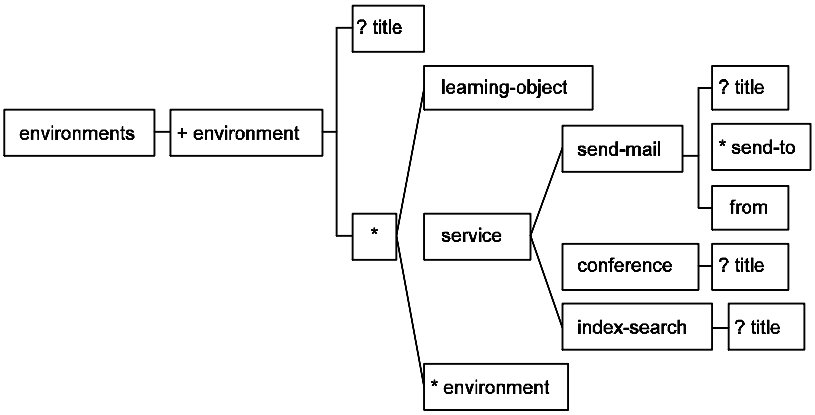


Figure 4.2 Environment tree schema

The attributes in table 4.2 may be defined for these elements:

Table 4.2 Environment tree node attributes

Attribute	Description
`class`	The class attribute allows the nodes to be typed by a space- separated list of types. For LD Level A this attribute should be considered merely as documentation. From Level B onwards it can be used to hide or show these nodes. The attribute may occur in `conference`, `index-search` and `send-mail`.
`conference -type`	This attribute indicates what type of conference is referenced by the conference element. Allowed values are synchronous, asynchronous and announcement. It is the responsibility of the integrating module to provide a link to a service having the appropriate features.

`identifier`	This attribute is the identifier of the object represented by the node. Note that this is not the identifier of the node itself and therefore multiple nodes may have the same identifier value if they are pointing to the same object! This identifier should be used when retrieving the content of the object represented by the node via the `getContent` API call. The identifier attribute is used in `index-search`, `learning-object` and `send-mail`.
`isvisible`	This attribute indicates whether a node is visible for the user. For level A it means that this value is identical to the value defined initially in the learning design because there are no constructs allowing the value to be changed. The attribute may occur in `conference`, `environment`, `index-search`, `learning-object`, `send-mail`.
`parameters`	This attribute contains the parameters defined in a learning design for a service. The attribute may occur in `conference`, `index-search`, `send-mail`.
`select`	This attribute defines who should receive the mail defined by the send-mail element. Allowed values are `person-in-role` and `all-persons-in-role`.
`type`	This attribute contains the type of the learning-object element as defined in LD.
`user-id`	This attribute is used in the `send-to` and `from` elements and contains the user ids of the receivers and sender of the mail. In Level B this will be extended with the email addresses of the sender and receivers of the email. This explains why the `from` element is available here already (for Level A it could be omitted as the sender's identity is known as he or she is typing the mail).

CONTENT

All nodes in both the activity tree and the environment tree may contain content. The content can be retrieved by calling the `getContent` method while passing the identifier of the object to be retrieved as parameter. Content is returned as personalized XML resembling the original learning design content. All content may include a title and metadata if these were defined in the UOL to which the content belongs. The `getContent` call does not return the actual content of the items. Each item contains a fully qualified URL to the location of the resource representing this content. So retrieving the complete content of any element consists of a two-stage process which involves as a first step the retrieval of a personalized XML structure of the content, followed by the retrieval of the resources referenced by the items.

Figure 4.3 shows the schema for the content model of a learning activity (the Learning-activity schema). Like all content objects, a learning activity may contain a title and metadata. Furthermore it may contain learning objectives, prerequisites and an activity. All these elements have exactly the same content structure, starting with one or more item elements which may be surrounded with an optional title and metadata. An item may have zero or more sub-items. Again, an optional title and metadata may be present. An item represents a kind of paragraph structure where the title element should be used as a heading. How this hierarchy is presented in the user interface is left to the integrator of CopperCore. An item has a required Uniform Resource Identifier (URI) (Berners-Lee, 1994) attribute that contains an absolute Uniform Resource Locator (URL) (Berners-Lee, Masinter, & McCahill, 1994 B.C.) to the location of the associated resource. A resource may be any resource that can be rendered in a web browser.

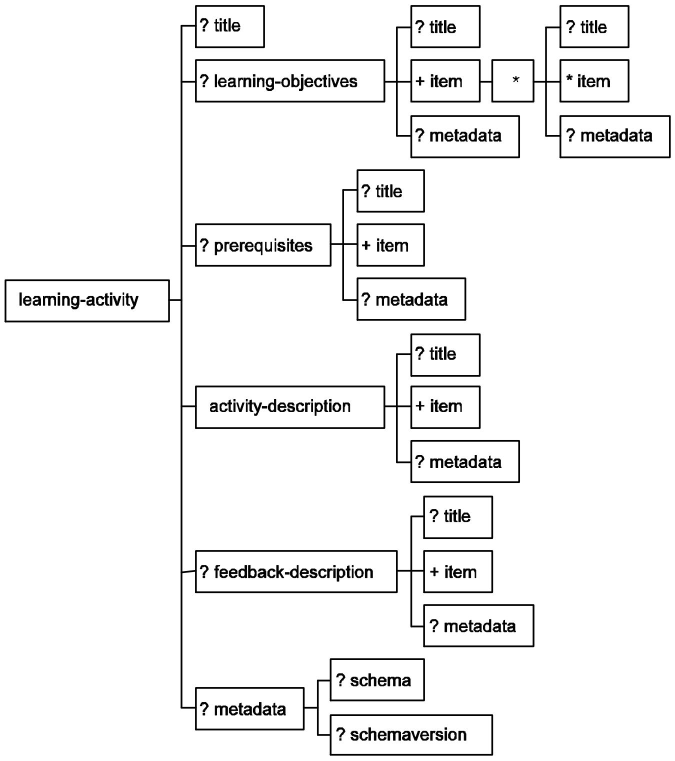


Figure 4.3 Learning-activity schema

The learning-objectives and prerequisite elements can also occur in the content model of a learning design. The feedback-description is only shown when it is present in the original UOL and if the user has completed the learning activity. `Feedback description` may also occur in the content models of the learning design the play and the act and will be present only if the corresponding element has been completed by the user.

Figure 4.4 shows the content model for a learner role (the learner schema). Clearly, the main structure of the content model is very similar for all elements. The information element that may be presented to the user as additional information is new. The information element may also occur in the staff and act element.

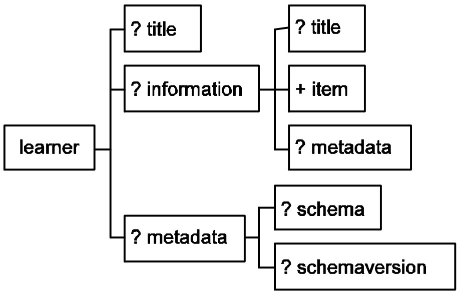


Figure 4.4 Learner schema

The content elements can contain a number of attributes included for reference only. The most relevant are presented in table 4.3.

Table 4.3 Learner tree node attributes

Attribute	Description
`Identifier`	The identifier of the object. It occurs in the elements `act`, `activity-structure`, `environment`, `item`, `learner`, `learning-activity`, `learning-design`, `learning-object`, `play`, `roles-to-support`, `send-mail`, `staff`, `support-activity`.
`isvisible`	This attribute holds an integer value indicating if an object was visible or not. This attribute may occur in the elements `act`, `item`, `play`, `learning-activity`, `support-activity`, `learning-object` and `send-mail`.
`url`	This attribute contains the absolute URL to an resource for which an item is a placeholder. The attribute occurs in the `item` element only.
`Class`	The class attributes assign an element to one or more categories. The visibility of these categories may be manipulated via conditions in Levels B and C of LD. The class attribute can occur in `send-mail` and `learning-object`.

OVERVIEW

Figure 4.5 gives an example of a typical calling sequence of the LDEngine API.

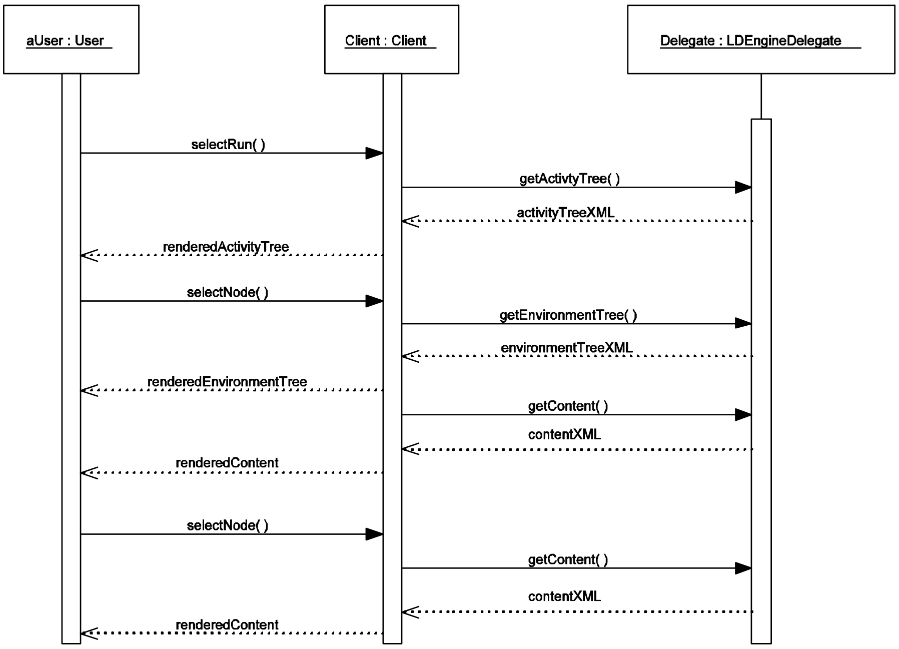


Figure 4.5 Sequence diagram of LDEngine calls

There are three “swim lanes” representing the user, the client integrating CopperCore and the CopperCore LDEngine API. In the example, a user starts by selecting one of the runs, probably from a list of runs for which the user is enrolled. After the user selects the run, the client application retrieves the activity tree for the user and run combination. The activity tree is returned as an XML file as discussed earlier. The client transforms this XML data in such a manner so that the user may select one of its nodes. After the user has selected a node from the activity tree, the client retrieves the environment tree belonging to this node. Both the identifier of the node in the activity tree and the list of environment objects are passed as parameters. As a result, CopperCore responds with the XML representation of the requested environment trees. The client renders this tree into a format suitable for the user. Next, the client retrieves the content for the node selected from the activity tree. The content is returned as XML and the client parses this content so it may retrieve all the needed resources referenced from the item inside the content. These resources are merged or linked and also presented to the user.

The user may now select a node from the environment tree. The client acts on this request by fetching the content from the CopperCore API and rendering the content in a similar fashion to the rendering of the content of the selected activity node.

This is merely a short example of the type of interaction which takes place between the user, client and CopperCore but it gives an idea of the dependencies between the activity tree, environment tree and content.

Technical overview

CopperCore is implemented using Sun's Java 2 Platform, Enterprise Edition (J2EE). The most pertinent reasons for this choice are:

- The kernel should be able to run on multiple platforms supporting multiple operating systems. Java is an obvious choice.
- The kernel should be accessible via web services or similar web-oriented technologies, but should allow for non-web-based access as well. Enterprise Java Beans (EJBs) provide a mechanism for this.
- The kernel should be scalable when necessary. This is another reason for choosing EJBs.

Figure 4.6 shows the technical architecture of the CopperCore kernel. All persisted data is stored in a relational database. CopperCore uses a JDBC driver to access the database. Using this extra layer between the data components and the actual database allows CopperCore to use different DBMSs by just switching the JDBC driver. The "Data Access Layer" is responsible for all interactions with the database and is made up of BMP entity beans. The "Database Access Layer" is split into two major parts.

The first part consists of properties. Although CopperCore currently only implements LD Level A, internally it depends heavily on the property mechanism. The other part of the "Database Access Layer" deals with course administration, which involves concepts such as users, runs, unit-of-learning etc.

The next layer of the architecture is the "Business Logic Layer" and contains all components representing the business logic of CopperCore. This layer is made up of a number of container components which are representations of the learning design that are directly or indirectly accessible through the API. Each container contains all the business logic it needs to adapt itself to the profile of the user accessing the LD component. For this purpose, the container makes extensive use of the property mechanism which contains its own business logic for retrieval and storage of properties. The EventDispatcher and EventHandler components deal with all event handling business logic occurring in the system. Finally the parser deals with the processing of an LD XML instance. It analyses and decomposes the LD into smaller parts suitable for further processing during runtime.

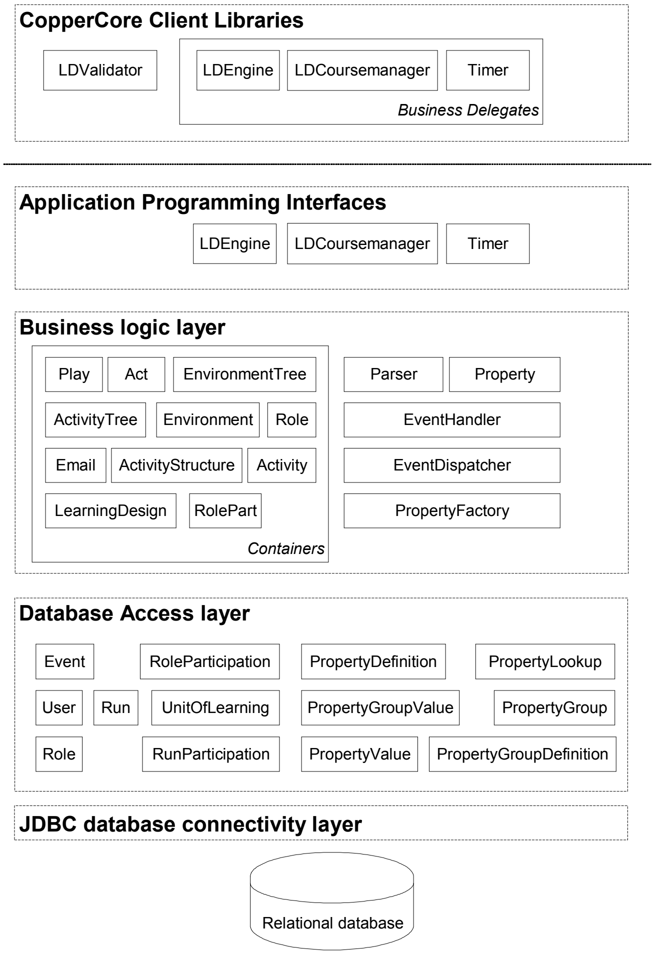


Figure 4.6 CopperCore technical architecture

The next layer comprises three session beans. The first bean is the `LDCourseManager` bean. It deals with all administrative calls necessary to prepare delivery of an LD instance. Typical interfaces offered deal with the publication of an XML LD instance, creation of users, creation of runs and assignment of roles. The second bean is the `LDEngine`. This is the core of the delivery mechanism. This bean handles the personalization of the LD instance for a particular user at a particular time. Calls that deal with the retrieval of personalized activity trees, environment trees and content are available. Finally, there is a `Timer` bean which deals with all time-related events specified in the learning design. Due to implementation restrictions in J2EE the clients should generate timer events on regular intervals by calling `proces()`. CopperCore does not make any assumptions about the granularity of the intervals, by ensuring no time-related events are missed.

The final layer is the “CopperCore Client Libraries” and is not a layer in the formal sense. It is a collection of libraries that should be used by an implementation making use of CopperCore. The most important library is the validator. As the name implies, the validator validates a UOL content package. Several checks are made to see if the package is complete, if the learning design is well formed and valid against the schema, and if the learning design is semantically correct. The library should be called by all clients to make sure that everything is correct before proceeding. In addition to the validator, three business delegates are offered for the three API beans. A business delegate contains the code to make the actual connection to the enterprise bean, making life easier for implementers.

Implementation Strategies

The main design decision when building CopperCore was to give implementers maximum flexibility to use the kernel in the way they see fit. However, this also implies that CopperCore itself is not a complete LD player. To make effective use of CopperCore, the kernel has to be integrated into a larger application. This application has to implement different services, the most important being the graphical user interface (GUI), without which the kernel cannot be used by an end-user. The GUI not only gives the learners and tutors access to the LD, but should also enable administrators to manage the learning process by letting them create new publications, add new users to the system, create a run for a publication, and so on.

The other major service being offered by the application is the possibility to serve the resources which are included in the LD package to the client. CopperCore does not implement a mechanism to deliver this content directly through the kernel. It does, however, extract the resources from the package and stores them on the file system when a UOL is published. Furthermore, CopperCore changes the local references to these resources into an application-specific reference, so the application is able to serve these resources to the end-user upon request. The easiest way to implement this service is to use a web server in the application.

CopperCore has been developed using J2EE. The kernel itself is implemented as three EJBs which must be installed and run on a Java Application Server such as JBoss (JBoss, 2004). This gives CopperCore the flexibility to run on different operating systems, the scalability to cope with load increases and the ability to be called from different kinds of clients (e.g. web-based clients or native Java clients). The downside of this approach is that the J2EE specification does not allow access to the underlying file system. CopperCore requires access to the file system to store the resources found in an LD package. To solve this problem CopperCore contains a CopperCore Client Library which is implemented as a set of Java classes that are used in the context of the calling application. This way access to the file system is allowed. Furthermore the library implements business delegates to hide the implementation details of accessing the remote EJBs which make up the CopperCore kernel.

Figure 4.7 shows the two main approaches to calling CopperCore. A client calls CopperCore directly via Java native calls, or an intermediate server allows clients to call CopperCore via the http protocol using a common web browser. Which approach to choose is up to the requirements of the software clients that access CopperCore. Different aspects of client software influence the decision for either a native Java client or a web browser client. When considering the ease of distributing the client application to the end-users, the web browser of course has the upper hand. No local software installation is required apart from having a recent web browser, which is the case for the majority of users. Updating the software is also easier using this web-based approach – only the web application on the server has to be updated to allow all users access to the latest version of the software. Compare this to delivering a new version of the software to individual users who may have different kinds of software configurations, different operating systems, different Java virtual machines, and so on. Furthermore, versioning becomes an issue as different users may install different versions of the client software.

Another issue is the access to the server. Since CopperCore runs on a Java application server, each client must have access to this server. In most places strict security policies exist making it easier to access the server via the most widely used port 80 for http traffic as opposed to the more obscure ports required for the native Java calls. Finally, rendering the LD content (mainly (X)HTML documents) is easier in a web browser.

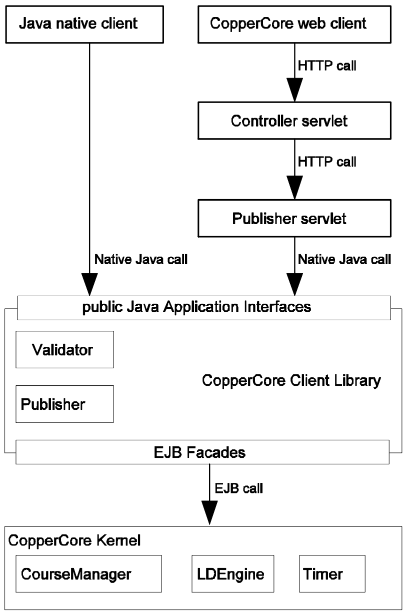


Figure 4.7 Implementation strategies for CopperCore

A native client is usually more responsive, the GUI can be more elaborate, making handling of large amounts of data more intuitive, and avoiding port 80 can make the application more secure by not exposing some of the APIs to the Internet.
A common way of building clients for CopperCore is to create a web client to be used by end-users acting as either a student or a tutor. In other words, these users are all assigned to one or more runs and access the UOL in the context of a role. For a user who administers CopperCore a native Java client might be more appropriate. The demonstration implementation which can be downloaded from http://coppercore.org illustrates this concept. It implements a web-based player used for accessing the LD. Although the interface is rather primitive it illustrates how such a web client could be built. For administrators, a simple command line interface to CopperCore (`clicc`) is implemented as a native Java application.

Building a web client requires implementers to create a web application. A common approach to implementing a web application on the J2EE platform is using servlets to dynamically create the Internet pages that are served to the browser on the end-user's machine. These servlets call the CopperCore kernel on behalf of the client to maintain the actions performed by the user and to retrieve the personalized LD based upon the actions. To ease access to the kernel, the web application should use the CopperCore client library as is shown in figure 4.7.

Building a native Java client is straightforward as far as the kernel is concerned. There are a few clearly defined APIs that can be called. Using the CopperCore client library makes accessing the kernel even easier by hiding all the intricacies of connecting to the remote EJBs. There is, however, one major issue in building a management application in this way. As noted above, an EJB is not allowed to access the file system. To circumvent this problem, CopperCore accesses the file system from within the client library. This client library, however, runs in the context of the calling application. In the case of a management application like `clicc`, this implies that access to the file system is in the context of the application itself. In other words, access to the file system is relative to the location of the application instead of to the location of the server. Being aware of this problem is the major hurdle for an implementer. The problem itself can be solved in different ways: `clicc` takes the easiest approach by running the application on the server itself, another option is to store the resources on a file share on the server, and finally an intermediate server application could be created which stores the resources of a publication in the appropriate place on the server.

Chapter 5

CopperCore Service Integration

Vogten, H., Martens, H.,Nadolski, R., Tattersall, C., Van Rosmalen, P., and Koper' R. (2007). CopperCore Service Integration. Interactive Learning Environments, Vol. 15(2), 171-180

Abstract

In an e-learning environment there is a need to integrate various e-learning services like assessment services, collaboration services, learning design services and communication services. In this article we present the design and implementation of a generic integrative service framework, called CopperCore Service Integration (CCSI). We will concentrate on the integration of two services: CopperCore, an IMS Learning Design service and an IMS Question and Test Interoperability service called Assessment Provision through Interoperable Segments (APIS). One of the design goals of the architecture was to minimize the intrusion for both the services as well as any legacy client that already uses these services. The result of this work is that the flow of learning activities can be made dependent on test results.

Introduction

This article describes the design and implementation of a generic integrative service framework, called CopperCore Service Integration (CCSI) (Vogten & Martens, 2006), for the IMS Learning Design (LD) specification (IMSLD-IM, 2003; IMSLD-BPG, 2003; IMSLD-XB, 2003). This work was done as part of the JISC ELF (Wilson, Blinco, & Rehak, 2004) (JISC, 2006) toolkit strand project called SLeD2 (2005) as a joint effort of both The Open University and the Open University of the Netherlands. The project extended earlier work which involved building an LD runtime service and a corresponding web-based client application called SLeD.

The LD runtime service, called CopperCore (Martens et al., 2004), processes units of learning (UOLs) which are IMS content packages containing a learning design defined in LD. CopperCore does not make any assumptions about the type of user interface used by the calling party. This allows CopperCore to be integrated in web clients as well as rich client platform applications. In fact, CopperCore does not provide any user interface at all, and all methods are only available through an Application Programming Interface (API). Therefore CopperCore cannot be used as a standalone product and must be used as a service integrated into a larger framework or Learning Management System (LMS). CopperCore relies on the provisioning of other services by this framework or LMS for parts of the LD processing.

Some of the services on which CopperCore relies are generic and may be used by other services as well. Examples of such common services are authorization and authentication. Although technically challenging, these types of services are not the focus of our work as they apply to all service oriented architectures. However, there are a number of e-learning oriented services that are tightly integrated with the LD specification that provide our focus. Typically, these can be found in the service section of the LD environment. Note the LD term service refers to the functional concept of a learning service supporting a user in the learning process. The LD term service does not refer to the technical notion of a service as in the term web service although the technical implementation of an LD service could well be achieved by a web service. The LD specification includes a number of services such as a mail service, synchronous and asynchronous conferencing service and an index and search service. LD also allows additional services to be specified when needed.

Furthermore LD specifies how other IMS specifications should be integrated. Examples of such specifications are the IMS Question and Test Interoperability (QTI) specification (IMSQTI, 2006) and the IMS Simple Sequencing specification (IMS Simple Sequencing, 2006). Although these specifications are quite clear on the authoring aspects of their integration, they are not particularly clear on their runtime aspects. An example is the integration of QTI items in the UOL. During runtime there must be a means of reacting to outcomes of QTI assessment items within the learning design workflow.

These implications are not well understood. The CCSI framework provides an extensible solution for the tight integration of loosely coupled services. The cross service concerns in particular are targeted by CCSI, alleviating the calling process from the burden of dealing with these concerns. In the remainder of this article the CCSI framework will be further elaborated by focusing on the integration of the CopperCore service and a QTI service which is called Assessment Provision through Interoperable Segments (APIS) (Barr, 2000). APIS is an implementation of a computer aided assessment service conforming to QTI and is also funded under the JISC ELF toolkit strand.

Integrating IMS Learning Design and QTIv2

With the release of the second version of QTI guidelines (QTIv2) for the integration of LD and QTI were described (IMSQTI-IG, 2006). The integration of LD and QTI revolves around aligning LD properties and QTI variable names. Essentially, when property identifiers and variable names are declared to be lexically identical at design time (that is in LD-based and QTI-based XML), they are considered to be a shared variable in runtime software environments that involve LD and QTI-based processing.

One implementation strategy for the guidelines above could be to build an integrated system combining the functionality of both the CopperCore and APIS service. However, given the considerable efforts that have been invested in the CopperCore and APIS services, this may not be an economically viable solution. Another approach would be an adaptation of both CopperCore and APIS allowing them to directly communicate with each other. This approach has two major drawbacks. First of all this introduces undesired dependencies between services. Secondly, this solution is not scalable as each new service being integrated requires an ever growing integration effort required to support communication with all the others. In the next section the architecture for CCSI is described that has none of the above drawbacks, together with a number of benefits.

CopperCore Service Integration Architecture

In order to make the service integration viable it is essential that the underpinning architecture is not intrusive, meaning adaptation to this architecture should only require minimal changes in the code of the existing services, like CopperCore and APIS and the existing clients using these services. Service and client implementers are unlikely to make it a priority to adapt their code solely for CCSI.

By the introduction of an intermediate service layer composed of a dispatcher and adapters we can meet the above requirements. This approach is a well known in the software industry and is described by the adapter design pattern (Gamma et al., 1995). The adapter pattern converts the interface of a class into another interface clients expect. Adapter lets classes work together that could not otherwise because of incompatible interfaces. In case of CCSI, each adapter is a software component encapsulating a single service implementation. The dispatcher is the central component, responsible for the orchestration between these services. To make this orchestration possible, all adapters share a common API providing the dispatcher a standard interface to all integrated services. Each adapter implements specific code to access the underlying service by implementing this common interface. This way the required code adaptations needed for the service integration are now encapsulated in the adapters, leaving the services untouched.

For each type of service (LD services, QTI services or conferencing services) multiple implementations may exist. In order to make these service implementations interchangeable a contract between the client and the adapter is introduced for each service type in the form of an interface. This principle is described by the bridge pattern, another well known design pattern (Gamma et al., 1995). The bridge decouples an abstraction from its implementation so that the two can vary independently. In the case of CCSI the bridge is the interface that describes the common functionality for the aforementioned service types. Adapters are allowed to extend this functionality by exposing the complete API of the underlying service implementations. Not only does this provide a richer system, it also makes the adapter transparent for any client using the original service. However, clients that make use of the extended functionality will need to be modified when another service implementation is used that does not provide this functionality.

Each interface is accompanied by an abstract adapter. Each abstract adapter implements the default hooks for the dispatcher. This alleviates the implementers of specific adapters from reimplementing these hooks over and over again. The implementations of these abstract adapters can act as proxy for the service preventing. However when needed additional actions may be added by the implementations of the abstract adapters. This principle is also known as the proxy pattern (Gamma et al., 1995).

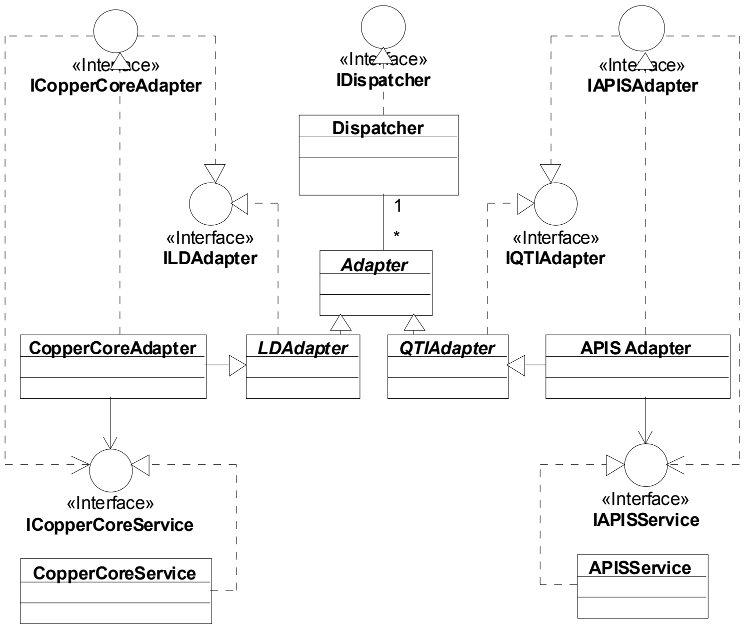


Figure 5.1 CopperCore Service Integration architecture

Figure 5.1 depicts the CCSI architecture. The dispatchers most important role is the propagation of events through all defined adapters. It is the responsibility of the adapters to listen for these events. Vice versa, it is the responsibility of each adapter to trigger the dispatcher when an event occurs that has potential cross service repercussions.

The dispatcher is also responsible for returning an adapter of the requested type to the client, thereby acting as an adapter factory corresponding to the abstract factory pattern (Gamma et al., 1995). This adapter factory is necessary because the types and implementation of the adapters are not known in advance, and may vary even during deployment by simply adding or replacing adapters. Adapters can come in two flavors depending on the way the client wishes to access the adapter. This can be done either via native Java calls or via SOAP web services. All adapters are declared in the CCSI service definition file. This file contains information about the base service type, the implementing Java class and WSDL URL.

Furthermore figure 5.1 depicts two adapter types; an adapter for the LD service and an adapter for the QTI service. Note that there could have been additional adapters for other services as well. The common interfaces for these service types are defined by the interfaces ILDAdapter and IQTIAdapter. Each adapter must implement the interface for its base type. Figure 5.1 also shows two abstract classes: LDAdapter and QTIAdapter These classes implement the hooks for the dispatcher. They act as extension points for any implementation of the LD or QTI services. Both the CopperCoreAdapter and the APISAdapter provide an interface that can be used by client applications. This interface is a replication of the original interface provided by the service that is being integrated, hence the dependency relationship between ICopperCoreAdapter and ICopperCoreService and between IAPISAdapter and IAPISService. By maintaining this relationship between the interfaces the impact for existing clients migrating to CCSI is limited to a minimum. Vice versa, when a service implementation is modified the impact is limited to the adapter acting as proxy for that particular service.

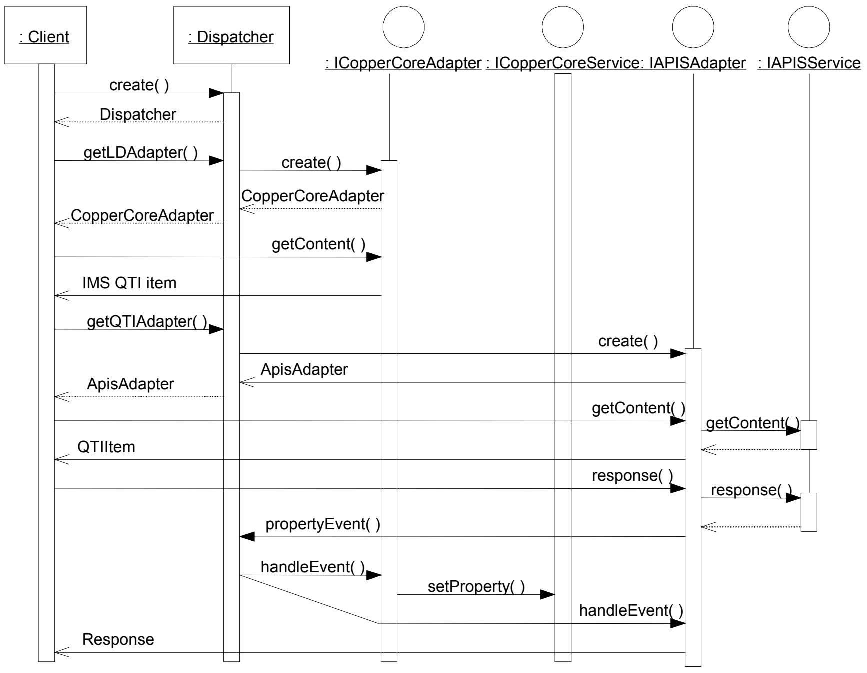
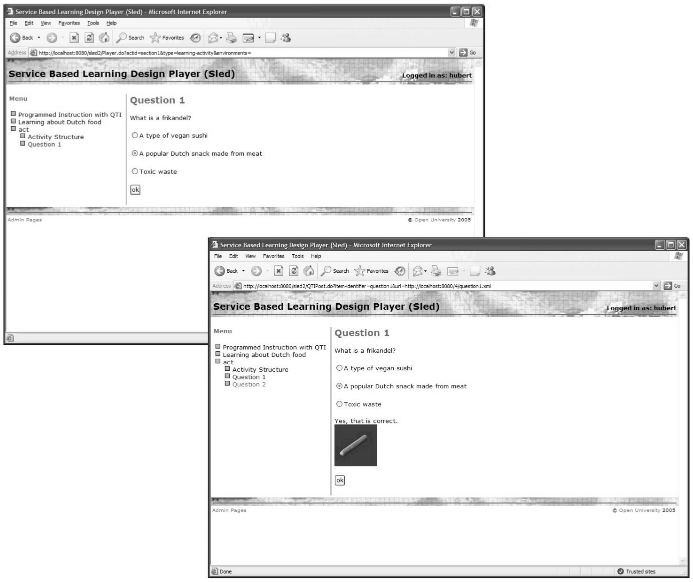


Figure 5.2 Sequence diagram showing the processing of a QTI item and the resulting event handling by the dispatcher

Figure 5.2 depicts a sequence diagram representing the processing of a QTI item within the context of a UOL run. The client (for example SLeD) creates a new instance of the Dispatcher. The dispatcher reads the CCSI service definition file and is informed about all available adapters. In the case of the example we only have the CopperCoreAdapter and the APISAdapter. Next, the client will request a handle for an LDAdapter. Depending on the technology used, an instance of the CopperCore adapter or a URL to the WSDL of the CopperCore adapter is returned. The dispatcher provides the client with an identical API in the CopperCoreAdapter compared to the original CopperCore service. So legacy clients, like SLeD, only have to be slightly modified. At some stage in the process the client retrieves QTI content and reacts by requesting the dispatcher to provide a handle to a QTI adapter. In our example the handle for the APIS adapter is returned. The client makes a request for the rendered content of the QTI item to the APIS adapter. The user response to this item is passed on to the APIS adapter. The APIS adapter processes this response, which results in a change of one of the variables defined by the QTI item's response section. It is the responsibility of the QTIAdapter to notify the dispatcher about this property event. In turn the dispatcher propagates this event to all adapters that have registered themselves as listeners for this event type allowing them to react to this event.

In order to synchronize the value of the QTI outcome variable, a corresponding LD property needs to be defined in the UOL. The CopperCoreAdapter will verify if this property exists and if so the value of the LD property will be set to the value of the QTI outcome. After all adapters have been informed about the property event, the result of the APIS adapter is finally returned to the client.


Service Based Learning Design Player (Sled) - Microsoft Internet Explorer

Address: http://localhost:8080/sled2/Player.do?actid=section1&type=learning-activity&environments=

Service Based Learning Design Player (Sled) — Logged in as: hubert

Menu
- Programmed Instruction with QTI
- Learning about Dutch food
- act
  - Activity Structure
  - Question 1

Question 1

What is a frikandel?

○ A type of vegan sushi
[✓ A popular Dutch snack made from meat]
○ Toxic waste

ok

Admin Pages — © Open University 2005

Service Based Learning Design Player (Sled) - Microsoft Internet Explorer

Address: http://localhost:8080/sled2/QTIPost.do?item-identifier=question1&url=http://localhost:8080/4/question1.xml

Service Based Learning Design Player (Sled) — Logged in as: hubert

Menu
- Programmed Instruction with QTI
- Learning about Dutch food
- act
  - Activity Structure
  - Question 1
  - Question 2

Question 1

What is a frikandel?

○ A type of vegan sushi
[✓ A popular Dutch snack made from meat]
○ Toxic waste

Yes, that is correct.

ok

Admin Pages — © Open University 2005

Done — Trusted sites

Figure 5.3 Two consecutive screenshots of the SLeD client are captured while processing a UOL containing a QTI item.

Integration of other services

CCSI was developed with the integration of different kinds of services in mind, especially those defined in the service section of LD although other types of services are conceivable too. In fact, in SLeD2 a number of adapters for these services were developed such as a search adapter and a forum adapter. The principle of integration is exactly the same as was done for the QTI adapter. However, the type of events that are dispatched may differ per adapter type. For example, for the forum adapter it is relevant to be informed about new runs (Tattersall et al., 2005a) being created for a UOL. A run is a runtime instantiation of a UOL and involves the enrollment of individual users to the defined roles in the UOL thereby populating the UOL. Similarly, it is relevant for the forum adapter to be informed about user subscriptions and role changes within the run of a UOL. The events are generated by the CopperCore adapter and can be picked up by any forum adapter. The forum adapter implementations may react to these events by creating new topics and granting users the corresponding access rights. Momentary two asynchronous forum adapters are developed, one for Moodle and one for Knowledge Network, a proprietary system of The Open University, which can be switched by merely changing the deployment configuration.

Another example of a service currently integrated through CCSI is the search service. Like with the forum adapter, there are two adapter implementations available. One adapter uses the Google API as search service provider. The other adapter uses the aforementioned Knowledge Network as service provider.

Related work

In the field of learning service integration some interesting related work has emerged. The IMS Tools Interoperability Guidelines (TIG) (IMS-TIG, 2006) is worth mentioning here. TIG deals with the interoperability of tools and LMS and is a first attempt to any standardization in this area. It shows some resemblance to the solution presented in this paper although there is a significant difference. The focus of SIG is mainly on technical aspects of the integration and less on the functional integration of the different services. TIG will not deal with any functional inter service dependencies, like the orchestration of property values between services, as shown in our example.

Another interesting, closely related development is the Business Process Execution Language (BPEL) (IBM et al., 2006) for Web Services. BPEL primary focus is the orchestration of SOAP web services. All logic for this orchestration is declared in an XML file which is interpreted by a BPEL engine. Recently tools for BPEL, like engines and editors have become widely available. Assis (Sherrat & Jeyes, 2006), also a JISC funded project, is worth mentioning in this context. In the Assis project BPEL was used for the orchestration of web services handling IMS Simple Sequencing and QTI.

BPEL holds some promising advantages over the presented approach in the paper, like standardization of the workflow solution and the separation of the workflow description from the actual implementation. However, at the same time extra overhead and complexity is introduced by the use of BPEL and one can argue if this outweighs the simplicity of CCSI. Especially in cases where services are not SOAP compliant the approach taken by CCSI has the advantage that it can make use of these services directly. This advantage should not be underestimated as the battle between SOAP and the its light weight counterpart ReST (Fielding, 2000) is not yet decided (zur Muehlen, Nickerson, & Swenson, 2004) either way.

Conclusion

Interoperability specifications like LD and QTI are having an ever growing impact on the e-learning community. As a result the number of implementations is steadily growing; initiatives such as the JISC ELF have demonstrated this via the delivery of several services dealing with these specifications (for example, APIS and CopperCore). However at the same time, runtime inter-specification operability issues are not yet understood. In this article, an approach was presented that deals with the interoperability of e-learning services within the context of LD. As basis for presenting the CCSI solution two service implementations were chosen; CopperCore and APIS. The need for integrating these two components can be explained by the fact that QTI is a natural complement to LD.

Both CopperCore and APIS were independently developed as part of the JISC ELF and both are already being used by legacy systems. The latter introduced an additional requirement as the identified solution must deal with legacy services and legacy clients. The switch to the new architecture should cause minimal intrusions in any existing code. Furthermore, the provided solution should be robust for new developments as the integrated services have their own development dynamics.

The CCSI architecture, informed by a number of design patterns, deals with these requirements by seamlessly inserting itself between the service and client. By replicating the original API the consequences for the client are limited to a switch of services factory. The underlying services do not have to be changed at all. All inter-service issues are dealt with in the adapter and dispatcher. We have seen that there is an adapter for each service type and that an adapter has a contract enforced by an interface per service type. The latter concept makes the adapter robust for changes in the services; it makes it possible to completely switch service implementations with minimal consequences.

Finally, as highlighted earlier the CCSI architecture is not limited to the integration of CopperCore and APIS. Other services such as the forum and search service can and in fact have already been integrated in a very similar manner although the types of events are different. For these services multiple adapter implementations exists which can be interchanged without any need to change any client code. The work on CCSI will be taken up by the European Commission funded TENCompetence (TENCompetence, 2006) programme.

All code for CCSI is available as open source and may be downloaded from SourceForge at http://sf.net/projects/ccsi. For an easy up and running example of CCSI the CopperCore Runtime Environment, also known as CCRT, can be downloaded from http://coppercore.org. This runtime contains deployable versions of the CopperCore service, the APIS service and the CCSI integrative service. Additionally, the SLeD2 player can be downloaded from http://sourceforge.net/projects/ldplayer. Finally, the example UOL can be downloaded from http://dspace.ou.nl/handle/1820/555.

Acknowledgements

The work on this paper has been partly sponsored by the TENCompetence Integrated Project that is funded by the European Commission's 6th Framework Programme, priority IST/Technology Enhanced Learning. Contract 027087 (www.tencompetence.org). The technical work that is described has been sponsored by the JISC ELF programme.

Chapter 6

Impact of CopperCore and CCSI

Introduction

In the previous chapters we discussed our approach to addressing the two research and development questions of this thesis. This resulted in the CopperCore and CCSI developments. In this chapter, we focus on the impact of CopperCore and CCSI on the LD community, which has resulted in their real-world use in various projects and initiatives. This use supports our claim that the presented designs and implementations have successfully answered our research and development questions. The relevant projects shall be described in more detail here.

CopperCore provided the LD community with a runtime to validate designs; LD authors have been using it as a reference for their own designs. At the same time, however, the community has tested CopperCore by deploying numerous designs in a form of real-world validation that did not reveal fundamental flaws in our approach, but rather (besides the occasional bug) some performance issues with the engine. In this chapter we discuss how we addressed these bugs and issues in several development iterations.

We illustrate various real-world cases where CopperCore has been reused, such as reuse of the engine through the APIs, source code, and engine design. We show that CopperCore has established itself as the de facto reference runtime in the LD community. By far the majority of initiatives and projects either use the engine or are referring to it as being the reference implementation. We also present some examples where new services have been successfully developed and incorporated in CCSI. This substantiates the claim that the presented architecture is extensible, but also that our CCSI architecture is generic as well, as some services are new to LD.

Finally, we argue that providing the community with a runtime implementation for LD was an important step in making the LD specification usable. At the same time, we suggest that the LD toolset's lack of maturity, in particular that of the authoring environments, is still a hurdle for the wider uptake of the specification.

Impact and development approach

The most recent version of CopperCore is version 3.1, which along with its sources can be downloaded from the SourceForge website. The chart in figure 6.1 shows the number of CopperCore downloads (about 10,000) since its release, as provided by SourceForge (2008). Of course, these statistics cannot directly be translated into actual usage numbers. But they are at least indicative, and do give an impression of the general interest CopperCore has raised so far. Even if only a small percentage of these downloads have resulted in actual engine use, it still amounts to significant usage numbers, certainly when taking into account that the engine is targeting a niche audience.

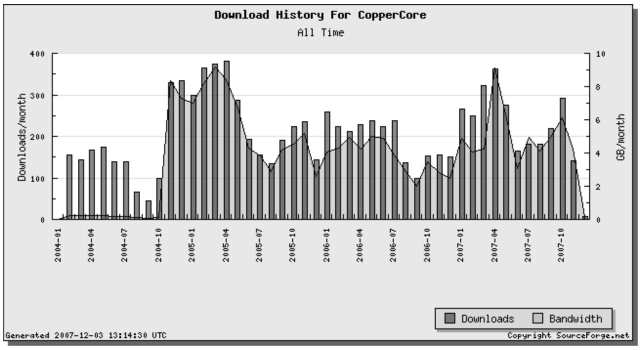


Figure 6.1 CopperCore download statistics from SourceForge

The statistics are cumulative, starting from the first release. Since then a number of new features have been added to CopperCore. The latest version includes support for all three levels of the specification, whereas only levels A and B were previously supported. The provided APIs now also support access via SOAP (W3C, 2000); this is similar to the APIs described in chapter 4 with the exception of the protocol used. An out-of-the-box installable version of CopperCore has been added (CopperCore Run Time, or CCRT) and incorporates CCSI. The example LD player is also shipped by default with CCRT and can render IMS QTI items through the use of CCSI as shown in chapter 5. Anyone wanting to experiment with LD can download CCRT to get started immediately.

All these developments took place in a number of iterations, most of them integral parts of international projects. Besides providing the necessary funding, these projects also provided valuable validation moments for the engine design, its internal workings and the correctness and completeness of the APIs. Other developers have been using the APIs to integrate the CopperCore engine into the larger frameworks of these projects, while authors have been using the engine in developing their own UOLs, and learners have used the outcomes of these developments in pilots. Each iteration provided an opportunity for further improvements and the inclusion of new features. We discuss these projects in some detail and explain what they contributed to the development and validation of CopperCore and CCSI.

A number of related projects and initiatives reused CopperCore and CCSI. Worth mentioning here is the UNFOLD project (UNFOLD, 2007), which aimed to support and facilitate Communities of Practice (CoPs) working with LD and related specifications. The idea stemmed from the so-called Valkenburg group of e-learning experts interested in improving the pedagogical quality of e-learning courses in an interoperable fashion, with user-friendly tools. This group reached consensus that EML and LD provide good starting points towards this ambition. We briefly mention some of the projects presented in the context of UNFOLD and report on the use of CopperCore and CCSI within these projects.

Finally, we also present projects and initiatives not directly related to the core development of CopperCore and CCSI nor disseminated through UNFOLD.

CopperCore development

The development of CopperCore was largely funded by the international projects described below. By aligning our development efforts with these projects we were able to establish a number of major development cycles. These projects thus contributed or still contribute to the core development of CopperCore and CCSI, and also provided opportunity to test the developments.

ALFANET PROJECT

The initial development of CopperCore was undertaken as part of the European ALFANET project (Van Rosmalen et al., 2007; Van Rosmalen & Boticario, 2005; Boticario & Santos, 2007; Fuentes et al., 2005) (IST-2001-33288). One of the key challenges addressed by ALFANET is adaptation of learning to learners' personal interests, characteristics and goals. Learners require content and activities based on their preferences and prior knowledge, not just static, page-turning sequences. ALFANET produced a learning environment that integrates principles and tools from the fields of learning design and artificial intelligence. This environment offers intelligent personalization capabilities that support effective and flexible learning scenarios consistent with the demands of the knowledge society. It was built in three main cycles, each incrementally increasing functionality. The first cycle ended up with a base system operating on top of LD level A. The second included an initial version of all components on top of LD level B, while the third offered an extensive set of adaptive features to choose from. All three cycles used CopperCore for processing the developed UOLs.

Figure 6.2 shows the final architecture of the ALFANET project (Santos, Boticario, & Barrera, 2008). This architecture consists of a dispatcher coordinating the services available in the system. One of these services is CopperCore, which is split into two parts in this diagram, corresponding to the developed APIs. The LD interpreter represents the functionality accessible through CopperCore's LDEngine API, and the Courses Manager represents the functionality available via its CourseManager API (see also chapter 4).

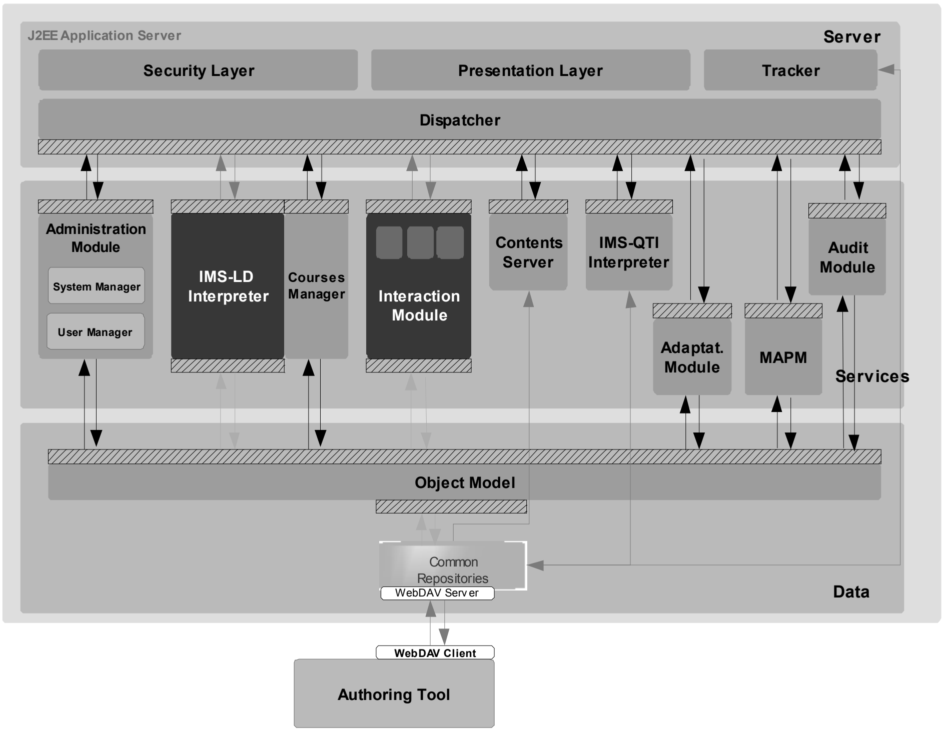
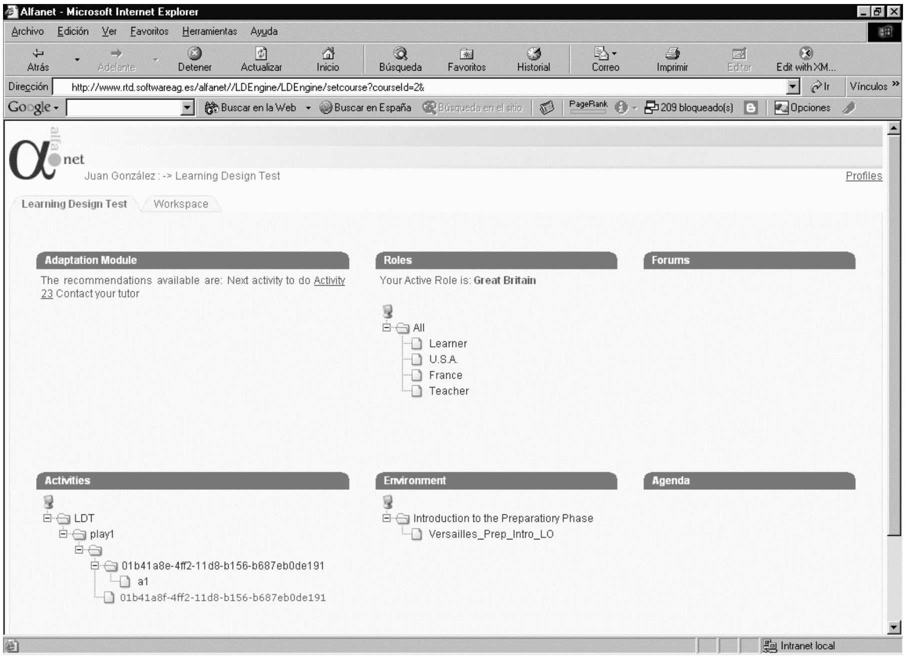


Figure 6.2 ALFANET system architecture

This service-based architecture integrates autonomous services such as CopperCore into the dotLRN learning management system (dotLRN, 2008), which itself is based on the OpenACS (OpenACS, 2008) framework. The dotLRN system also provided the core of the presentation layer. The CopperCore player used in ALFANET was based on the demo web player provided with CopperCore (a snapshot of this integration is shown in figure 6.3). ALFANET also focused on the use of e-learning standards and provided early IMS QTI integration. However, compared to CCSI, this integration is looser and more indirect. Both the LD and IMS QTI specifications run side by side, and integration is established through ALFANET's adaptation module.

Each development cycle included an evaluation round with users from different backgrounds (company employees, private persons and university students) in different domains. There were courses for university students on 'How to teach through the Internet' (UNED) and 'Communication technology' (OUNL); a Spanish course for private German learners (KLETT); and an internal staff training course 'Environment and electrical distribution' (EDP). The evaluation results can be found in (Barrera et al., 2005). These pilots used LD to model their predesigned adaptations (Towle & Halm, 2005) and thus provided the first validation of the CopperCore engine. The work revealed several bugs in engine implementation which were subsequently corrected; but it also showed that the design principles of the engine were sound because the learning designs used for the pilots incorporated complex adaptations. Evaluation of the pilots revealed that authors found the environment complex, while the users found the authoring process too formalized and felt that production and presentation integration were lacking (i.e. no sense of 'what you see is what you get').


Alfanet - Microsoft Internet Explorer
Archivo Edición Ver Favoritos Herramientas Ayuda
Atrás Adelante Detener Actualizar Inicio Búsqueda Favoritos Historial Correo Imprimir Editar Edit with XM...
Dirección
Vínculos
Buscar en la Web Buscar en España Búsqueda en el sitio PageRank 209 bloqueado(s) Opciones
net
Profiles
Learning Design Test Workspace
Adaptation Module
Roles
Forums
Activities
Environment
Agenda
Intranet local

http://www.rtd.softwareag.es/alfanet//LDEngine/LDEngine/setcourse?courseId=2&

Juan González : -> Learning Design Test

The recommendations available are: Next activity to do Activity 23 Contact your tutor

Your Active Role is: Great Britain

```
All
  Learner
  U.S.A.
  France
  Teacher
```

```
LDT
  play1
    01b41a8e-4ff2-11d8-b156-b687eb0de191
      a1
    01b41a8f-4ff2-11d8-b156-b687eb0de191
```

```
Introduction to the Preparatiory Phase
  Versailles_Prep_Intro_LO
```

Figure 6.3 Screenshot of the ALFANET system

CopperCore version 1.0 was released as the outcome of the work for the ALFANET project. This engine was LD level A and B compliant, and provided the LD community with the first runtime ever.

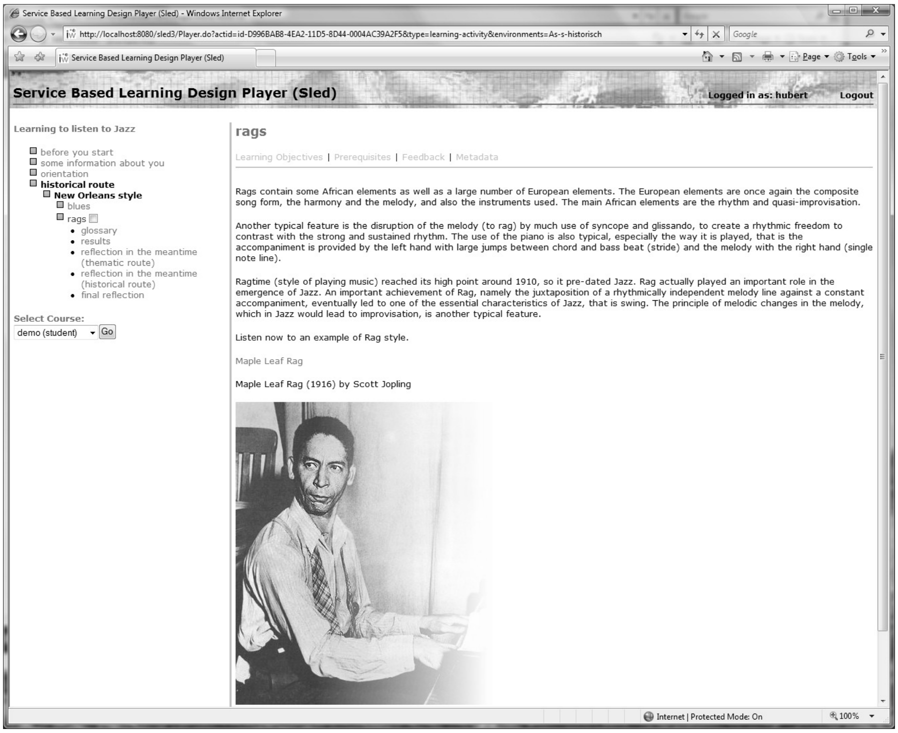
SERVICE-BASED LEARNING DESIGN PLAYER

The development of SLeD (SLeD, 2005; McAndrew et al., 2004; McAndrew, Nadolski, & Little, 2005) has been a collaborative effort of the Open University in the UK (OUUK) and the Open University of the Netherlands (OUNL). The project was funded under the JISC eLearning Programme (JISC, 2006) by four project grants, and also linked to the UNFOLD (UNFOLD, 2007) project.

The first SLeD project (SBLDS, 2004) aimed foremost at developing a service-based LD player. For this purpose a new LD player was developed that used the CopperCore APIs. A web interface was also developed for the CourseManager API. The SLeD player integrated proprietary learning services provided by OUUK, and its development was carried out by OUUK. OUNL was responsible for the further development of CopperCore. This resulted in its second release, this time including support for LD level C and SOAP compliant APIs. The latter opened up the use of CopperCore to a broader range of applications, as it meant the API was no longer dependent on the use of Java. With support from the Reload development team, CCRT was produced. This allowed SLeD to be installed almost out of the box.

The second SLeD project (DLD, 2005) was targeted at content-based demonstrators with a range of features and illustrating the reuse of the outcomes of the previous project. As a sort of showcase, the services provided by the previous project were to be integrated with an existing learning management system. For this purpose, SLeD was loosely integrated with the Moodle system (Moodle, 2006), which provided SLeD with a forum service; it also meant the SLeD player could be used from within Moodle. The outcomes of this project were disseminated to the wider LD user community via UNFOLD, and various UOLs developed and demonstrated using these SLeD tools.

The third SLeD project (SLeD2, 2005) targeted the development of a technical methodology for integrating service calls in LD. This generic approach, implemented through CCSI, was demonstrated by integrating an IMS QTI service (see chapter 4 for details). OUNL was responsible for the design and development of the CCSI framework and the IMS QTI adapter, while OUUK implemented all other adapters (such as the e-Portfolio adapter) and updated the SLeD player to work with CCSI and the new adapters. A number of other services were also integrated as part of the work on this project, including a search service via Google and integration with an e-portfolio service. Other projects, too, contributed service adapters. The e-Adventure (Moreno-Ger, Martínez-Ortiz, Luis Sierra, & Fernández/Manjón, 2007) used CCSI to integrate a game service with LD. The TENCompetence project developed an adapter for integrating widgets (Wilson, Sharples, & Griffiths, 2007). Furthermore, the adapter concepts within the CCSI framework were taken up by the E-Framework Services for Course Evaluation project (EFSCE, 2007).


Service Based Learning Design Player (Sled) - Windows Internet Explorer
http://localhost:8080/sled3/Player.do?actid=id-D996BAB8-4EA2-11D5-8D44-0004AC39A2F5&type=learning-activity&environments=As-s-historisch
Google
Service Based Learning Design Player (Sled)
Page
Tools
Internet | Protected Mode: On
100%

Service Based Learning Design Player (Sled) **Logged in as: hubert** **Logout**

Learning to listen to Jazz

- before you start
- some information about you
- orientation
- **historical route**
  - **New Orleans style**
    - blues
    - rags
      - glossary
      - results
      - reflection in the meantime (thematic route)
      - reflection in the meantime (historical route)
      - final reflection

Select Course:
demo (student) Go

rags

Learning Objectives | Prerequisites | Feedback | Metadata

Rags contain some African elements as well as a large number of European elements. The European elements are once again the composite song form, the harmony and the melody, and also the instruments used. The main African elements are the rhythm and quasi-improvisation.

Another typical feature is the disruption of the melody (to rag) by much use of syncope and glissando, to create a rhythmic freedom to contrast with the strong and sustained rhythm. The use of the piano is also typical, especially the way it is played, that is the accompaniment is provided by the left hand with large jumps between chord and bass beat (stride) and the melody with the right hand (single note line).

Ragtime (style of playing music) reached its high point around 1910, so it pre-dated Jazz. Rag actually played an important role in the emergence of Jazz. An important achievement of Rag, namely the juxtaposition of a rhythmically independent melody line against a constant accompaniment, eventually led to one of the essential characteristics of Jazz, that is swing. The principle of melodic changes in the melody, which in Jazz would lead to improvisation, is another typical feature.

Listen now to an example of Rag style.

Maple Leaf Rag

Maple Leaf Rag (1916) by Scott Jopling

Figure 6.4 Screenshot of SLeD player

The SLeD outcomes were evaluated by Liverpool Hope University using real students in their SLIDe project (Barret-Baxendale, Hazlewood, Oddie, & Anderson, 2005; SLeDID, 2005). One conclusion was that SLeD performed poorly in real-life situations, resulting in unacceptable response times. Therefore, the final project in the SLeD development, D4LD, aimed to mediate these performance issues. Funded by the JISC Design for Learning Programme (D4LD, 2006), the project aimed to improve general SLeD performance to make it more suitable for running real-life courses with the infrastructure. This resulted in the latest 3.1 release of CopperCore and CCSI, incorporating considerable performance benefits. As part of this project, OUUK carried out a performance stress test (Hutchinson, 2007). The results indicated that the performance issues reported by Liverpool Hope had been solved. However, they also show that CopperCore and CCSI are not yet ready for large-scale enterprise level deployment, because overall performance drops considerably with serious user load. Once the load passes a critical threshold, more request are coming in than can processed on average resulting in an ever growing queue of outstanding requests. The performance stress test suggests that the critical threshold lies somewhere between 150 and 200 concurrent users.

UNFOLD

The UNFOLD project was conceived to promote and coordinate the adoption, implementation and use of IMS Learning Design and related specifications. The UNFOLD project team argued that it would take the active involvement of many different professional groups for the IMS Learning Design specification to successfully provide better learning opportunities. But often these groups are not in contact with each other: those developing specifications do not usually work with authors of learning materials, and tools developers do not usually work with teachers and learners. If progress is to be made on these aims, information needs to flow between these disparate groups of people.

To meet this need, UNFOLD's core activity has been to support and facilitate CoPs, groupings of people who come together based on common interests and expertise, creating, sharing and applying knowledge within and across the boundaries of tasks, teams and organizations. Three CoPs were launched in July 2004, for systems developers, learning designers and teachers.

Several CoP meetings were organized, bringing experts together to share their experiences. CopperCore was presented to the broader community of practice during the first UNFOLD meeting. Participants were invited to create their own UOLs by using the Reload (Reload, 2007) authoring tool. The UOLs produced were then evaluated using the CopperCore engine and demo player. During these meetings, the ALFANET and SLeD projects were also presented. The following list summarizes some of the work presented that used the CopperCore engine.

- Reload (Milligan, Beauvoir, & Sharples, 2005; Reload, 2007) has established itself as the reference LD authoring environment. The Reload editor has been extended with a Reload Learning Design Player which provides preview functionality. This player uses the CopperCore engine and a modified version of the LD player provided with CopperCore. CopperCore is bundled together with the Reload Learning Design Player into one installation package. To this end, the Reload and CopperCore teams worked together on the configuration of CCRT.

- Another LD editor and designer is CopperAuthor (Van der Vegt, 2006), which has similar preview functionality to Reload and also integrates CopperCore for this purpose.

- Collage (Hernández-Leo et al., 2006a) is an adaptation of the Reload editor, and allows teachers to design collaborative learning experiences without specific LD knowledge. It makes use of collaborative learning flow patterns representing practitioners' best practices to model the flow of collaborative learning activities. For this purpose, Gridcole (Hernández-Leo, Villasclaras-Fernández, Asensio-Pérez, Dimitriadis, & Marcos-García, 2006b) has been developed which uses an adapted version of the CopperCore engine. This adaptation deals with the integration of several collaborative services.

- In their article 'Learning units design based in grid computing', (Navarro, Diaz, Such, Martín, & Peco, 2007) introduce the notion of grid learning objects as an alternative to user grid computing in e-learning. To demonstrate their approach, the authors implemented a system that integrates a GRID infrastructure and the CopperCore engine.

- Researchers from the COLLIDE group at Universität Duisburg-Essen introduced an extension to the CopperCore engine that uses LD to control learning support environments remotely (Harrer, Malzahn, Hoeksema, & Hoppe, 2005). They achieved this by scripting the learning flow in LD and allowing CopperCore to interact with the learning support environments. Their approach resembles the more generic approach taken in CCSI.

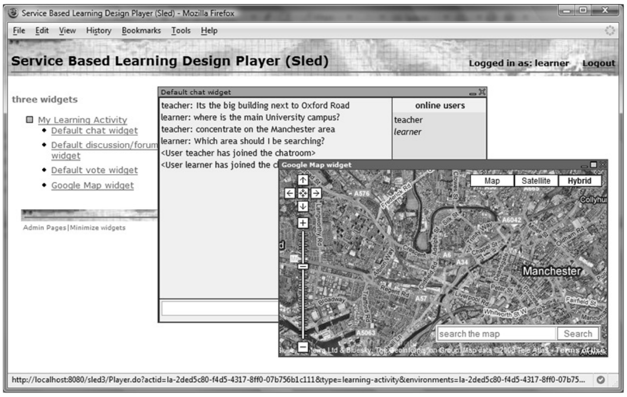
- In 'Crosscutting runtime adaptations of LD execution' (Zarraonandia, Dodero, & Fernández, 2006) the authors explain how they increase UOL reusability by offering designers an alternative to predesigned adaptation by allowing slight alterations of the original design during runtime. Their approach aims for the middle ground between LD's formal, elaborate process of top-down authoring and the need for authors to do quick modifications during runtime. They achieved this without modifying CopperCore. Because they considered these types of modifications as crosscutting concerns, they decided to intervene on the engine's output through aspect-oriented programming (Elrad, Filman, & Bader, 2001). This allowed them to make the necessary changes without having to modify any code to the engine itself.

- GRAIL (Gradient-lab RTE for Adaptive LD in dotLRN) is an alternative LD runtime implementation developed at the Telematics Engineering Department of the Carlos III University of Madrid (Escobedo de Cid, Fuente Valentín, & Guitérrez, 2007). The CopperCore engine design served as a reference for GRAIL's design, which is implemented as extension of the dotLRN framework. This framework was also used in the ALFANET project. The outset for GRAIL is very different from that of CopperCore: the latter is implemented as an independent service that requires further integration into a learning management system such as dotLRN; GRAIL, however, is tightly integrated with the dotLRN framework from the start. This conscious choice for a single framework means GRAIL is directed at end users, whereas CopperCore is more directed at developers. The deliberate choice for the dotLRN framework, or any modular e-learning environment for that matter, makes the integration of e-learning services much easier. First, these services are already provided by the platform, and second, the platform architecture most likely provides the scaffolding that allows this integration. GRAIL also provides a player and an interface for UOL administration. An obvious drawback of the choice for dotLRN is the automatic exclusion of other learning management systems.

In the context of UNFOLD, various UOLs have been produced and tested through CopperCore and can be retrieved from the UNFOLD website. CopperCore was used as a reference by the UNFOLD community to help better understand the LD specification (Griffiths, Blat, Elferink, & Zondergeld, 2005a). As these insights grew, the community also helped test the engine in real practice by deploying various learning designs at all levels of LD. A collection of these designs have gathered and can be downloaded through the Learning Networks (2008) repository. In most cases, CCRT was installed for this purpose either with or without the SLeD player. However, the UNFOLD CoP also reused the CopperCore engine itself via the APIs and/or directly through modifications of the source code or code introspection. Furthermore, CopperCore has been recognized by the CoP as the de facto reference implementation for LD. We therefore conclude that the UNFOLD experience supports our claim of having answered the first research and development question of this thesis.

TENCOMPETENCE

The latest project pushing the CopperCore and CCSI developments forwards is TENCompetence (Koper & Specht, 2007; TENCompetence consortium, 2007), which has devoted a work package to the further development of LD-related tools. As part of this work package, the CopperCore environment has been further extended to support the integration of SCORM and LD (Tattersall, Vogten, Martens, & Koper, 2006). This integration is implemented by extending CCSI with an additional SCORM adapter. In a similar fashion, CCSI has been extended with a widget adapter, allowing the integration of a complete range of widgets in the CopperCore environment (Wilson et al., 2007). Figure 6.5 shows two examples of widget adapters – a chat widget and a Google Maps widget – working with SLeD. The widget adapter allows the easy and flexible adaptation of various widgets to the runtime environment.



teacher: Its the big building next to Oxford Road
learner: where is the main University campus?
teacher: concentrate on the Manchester area
learner: Which area should I be searching?
<User teacher has joined the chatroom>
<User learner has joined the c

teacher
learner

http://localhost:8080/sled3/Player.do?actid=la-2ded5c80-f4d5-4317-8ff0-07b756b1c111&type=learning-activity&environments=la-2ded5c80-f4d5-4317-8ff0-07b75...

Figure 6.5 Screenshot of chat and Google Maps widgets

It will be possible to configure these widgets with the ReCourse editor (Griffiths, Beauvoir, Barret-Baxendale, Hazlewood, & Oddie, 2007), the Reload successor currently being developed in TENCompetence. TENCompetence has finished its second year, and work on developments such as the ReCourse editor and widget framework will continue for at least two years.

This concludes our review of the projects that significantly contributed to the development of CopperCore and CCSI. Next we discuss the UNFOLD project, which acted as a dissemination and discussion platform for many LD-related developments.

TELCERT AND ELEGI

Besides the aforementioned projects, a number of other initiatives made use of CopperCore and CCSI in some form or another. Of these, we briefly mention TELCERT (Nadolski, ONeill, Vegt, & Koper, 2006) and the European Learning Grid Infrastructure project (ELeGI) (ELeGI, 2007; Gaeta, Gaeta, & Ritrovato, 2007). TELCERT's aim is to help transform the adoption of standards-based e-learning products and services by providing tools and test systems that assure interoperability. It produced an LD application profile that could generated reference UOLS, which were tested using CopperCore.

The ELeGI project, an EU-funded integrated project with 23 partners from 9 EU countries has the ambitious goal to 'radically advance the effective use of technology-enhanced learning through the design, implementation and validation of a pedagogy-driven, service-oriented software architecture based on Grid technologies'. The ELeGI project is structured along two main action lines: ELeGI formal and ELeGI informal. Figure 6.6 shows the architecture devised for the formal action line.

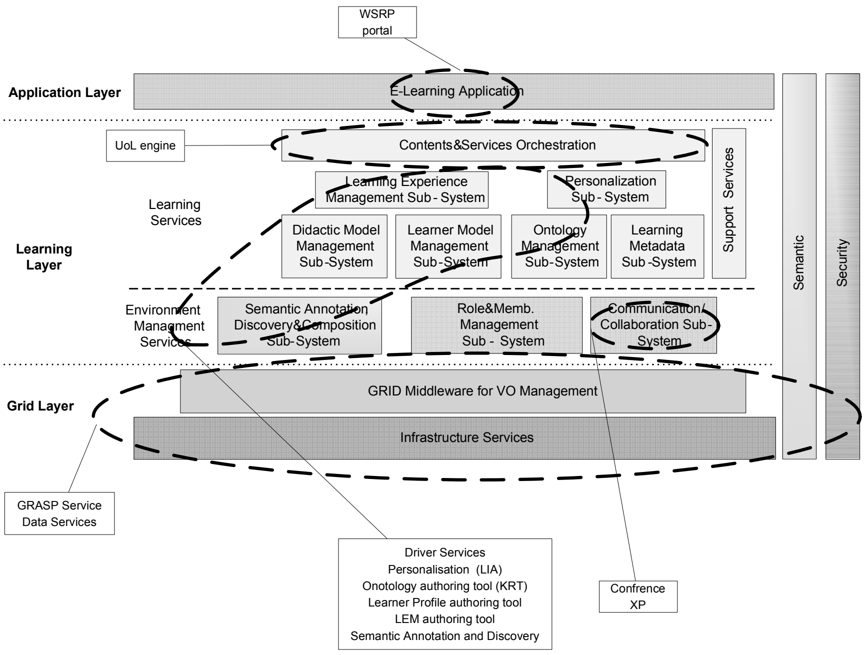

WSRP portal
Application Layer
E-Learning Application
UoL engine
Contents&Services Orchestration
Learning Experience Management Sub-System
Personalization Sub-System
Learning Services
Learning Layer
Didactic Model Management Sub-System
Learner Model Management Sub-System
Ontology Management Sub-System
Learning Metadata Sub-System
Support Services
Semantic
Security
Environment Managment Services
Semantic Annotation Discovery&Composition Sub-System
Role&Memb. Management Sub - System
Communication/ Collaboration Sub-System
Grid Layer
GRID Middleware for VO Management
Infrastructure Services
GRASP Service Data Services
Driver Services
Personalisation (LIA)
Onotology authoring tool (KRT)
Learner Profile authoring tool
LEM authoring tool
Semantic Annotation and Discovery
Confrence XP

Figure 6.6 ELeGI architecture (source: ELeGI final report)

The Contents & Services Orchestration sub-system deals with the issues of execution of UOLs described using the LD. CopperCore has been used as engine for this purpose. The architecture allows the dynamic binding of learning resources, learning objects and services by exploiting the underlying grid layer. CopperCore was modified to enable this dynamic binding of learning resources. The ELeGI project is an example of cross-platform integration because large parts of its infrastructure were developed with .NET.

Conclusions

In this chapter we discussed what impact both CopperCore and CCSI had on the LD community. Indicative of this impact are the SourceForge download statistics, which add up to 10,000 downloads. We also described several projects in detail that either contributed to the development of CopperCore and CCSI, or have been using and reusing them.

CopperCore and CCSI were developed in several cycles as part of work undertaken in the ALFANET, SLeD and TENCompetence projects. Each of these projects led to new releases of CopperCore and CCSI. In addition, they provided not only the necessary resources for these developments but also validation of the design and implementations. Developers not directly involved in the development of CopperCore and CCSI used the APIs to integrate the engine in other frameworks and players. CCSI was extended with new adapters such as a SCORM and a widget adapter. CopperCore's source code was also reused and modified to fit specific purposes within the projects. Although CopperCore and CCSI were first and foremost developed as reference implementations to stimulate further uptake of the LD specification, we also paid attention to performance issues in the D4LD project. Performance stress tests showed that the SLeD environment is suitable for deployment scales of up to 150 simultaneous users.

The UNFOLD CoP have been using and reusing CopperCore and CCSI in various projects with a complete range of learning designs. This real-world usage not only underlines the importance of having a reference implementation for the LD specification, but also provides practical evidence for the soundness of our engine design and implementation. CopperCore helped the CoP better understand the LD, a whole range of UOLs have been produced and validated via CopperCore, and these UOLs helped test the CopperCore engine.

This does not mean there been no criticism. The performance issues were already mentioned. Various other issues, too, were encountered over time, as could be expected. Nevertheless, they all boiled down to coding bugs: so far there have been no problems that would indicate a fundamental flaw in the design of either CopperCore or CCSI. Most reported issues were resolved in the subsequent development cycle, and became available with the next release. Another point of criticism concerned our choice to use J2EE and EJBs for persistence as it was deemed unnecessarily complex. Developers with the necessary skills are scarce, and in retrospect, we have to admit that this has probably limited engine developments to some extent. Some of the benefits we expected as result of our use of J2EE never materialized; optimising an enterprise server does require considerable skill and expertise. In addition, rather than benefiting from increased performance, the EJBs seemed to slow down the engine. Potential gains expected from load balancing were never really put into practice. Emerging technologies such as Hibernate (Hibernate, 2008) and JPA (EJB 3.0 software expert group, 2008) would probably have been better persistence frameworks. It should be emphasised, however, that such theorising is easy with the wisdom of hindsight about trends in software development and use of the CopperCore engine. Initial research into migration towards a lighter framework using a simple servlet container such as Tomcat and a persistence framework like JPA indicates that migration seems feasible without massive effort, though further investigation is needed.

The first research and development question of this thesis was:

i) *How can a fully compliant reusable reference runtime environment for the IMS Learning Design specification be designed and implemented?*

We conclude that we were successful in answering this question with the CopperCore engine and its underlying design, both of which have demonstrated sound in real-world practice. Furthermore, CopperCore has demonstrated in practice to be capable of processing all levels of specification. This can be concluded from the various UOLs of all levels that were created and successfully deployed by the CoPs. The engine has established itself as a de facto reference implementation for LD both for learning design authors and developers. Furthermore, the engine has been reused in various ways and in all kinds of situations, as discussed previously.

The second research and development question was:

ii) How, given a reference implementation for the IMS Learning Design specification, can implementations for other e-learning specifications and learning support services be integrated generically at runtime level?

We conclude that we successfully answered this question through CCSI, which provides a generic framework for integrating other specifications and e-learning services. The SLeD project demonstrated such integration via the CCSI framework. In similar fashion, adapters for gaming and widgets have been developed and integrated. These adapters in particular are evidence of the generic nature of the solution, as they are not part of the standard services originally defined in LD.

We also conclude that CopperCore and CCSI have played a critical role in the uptake of LD. The first hurdle for LD uptake (i.e. having a runtime for the specification) has been overcome. However, at the same time we must conclude that this uptake has been disappointing so far. Although LD generated lots of interest within the educational research sector, it not really left the research and development stages behind. In ‘Creating an 28 weeks course with LD and CopperCore’ (Spang Bovey & Dunand, 2006) and ‘Panning for gold’ (Bailey, Zalfan, & Davis, 2006), the authors argue that the maturity of the LD toolset is a major obstacle to uptake. Elsewhere, the current LD editing environments are identified as major hurdles due the amount of expertise they require (Sodhi, Miao, Brouns, & Koper, 2007). The ALFANET evaluation (Barrera et al., 2005) led to similar conclusions. In ‘Using the IMS Learning Design notation for the modelling and delivery of education’ (Tattersall, Sodhi, Burgos, & Koper, 2007), the authors argue for a balance between a restrictive environment and an unsupportive one, taking into account that teachers do not like prescriptive methods. In ‘Crosscutting runtime adaptations of LD execution’ (Zarraonandia et al., 2006) the authors also argue for more flexibility and adaptation possibilities during runtime. In the next chapter we therefore present a complementary approach for authoring LD which leans heavily on the close integration of CopperCore and an environment for personal competence development.

Chapter 7

Using the Personal Competence Manager as a complementary approach to IMS Learning Design Authoring

Vogten, H., Koper, R.,Martens H. and Van Bruggen, J. (2008). Using the Personal Competence Manager as a complementary approach to IMS Learning Design authoring. Interactive Learning Environments, Vol 16(1), 83-100

Abstract

In this article TENCompetence will be presented as a framework for lifelong competence development. More specifically, the relationship between the TENCompetence framework and the IMS Learning Design (LD) specification is explored. LD authoring has proven to be challenging and the toolset currently available is targeting expert users mostly working for institutions of higher educations. Furthermore these tools reinforce a fairly rigid top-down workflow approach towards design and delivery. This approach it is not always the most suitable model in all circumstances for all practitioners. TENCompetence provides an alternative bottom-up approach to LD authoring via its first implementation: the Personal Competence Manager (PCM). Constructs such as competence profiles and competence development programmes, let users define, modify, and acquire competences they need for achieving their personal goals. We will show how the PCM provides support for these constructs and stimulates the bottom-up development of learning materials. We will also show how these concepts can be mapped towards LD. This allows the *ad hoc* designs of the PCM to be captured in a Unit of Learning (UOL). These UOLs can be enhanced and eventually fed back into the PCM, therewith closing the edit cycle. This editing cycle allows for a gradual integration of bottom-up *ad hoc* designs with more formal top-down designs introducing LD in a gentle fashion.

Introduction

Emerging e-learning standardization initiatives have led to a number of interesting new specifications and standards. One of those initiatives is IMS Learning Design (LD) (IMSLD, 2003; Koper & Olivier, 2004; Olivier & Tattersall, 2005). LD is a formal language for the specification of learning designs using semantically meaningful concepts from the pedagogical domain. The most relevant objectives achieved by applying LD are formalization, reproducibility and reusability of the learning designs. LD is a very expressive specification capable of describing a wide variety of learning designs. However it is also a very complex and complicated specification. The current toolset supporting LD is still very closely and directly informed by the specification itself and requires a profound understanding of the specification. Therefore LD is used mainly in institutions for higher education where sufficient expertise is available to work with the current toolset.

The recently launched TENCompetence initiative targets the development of an infrastructure for lifelong competence development. TENCompetence has the ambition to support formal and informal learning during the lifetime of an individual. TENCompetence ambitions reach beyond the scope of the educational institutions.

The first release of the TENCompetence software is called the Personal Competence Manager (PCM). The PCM provides an integrated environment for both learning and authoring without making a clear distinction between the two modes. This article will show how this aspect can be beneficial for the easy creation of simple units of learning (UOLs). A UOL is the collection of files including the learning design expressed in LD that is ready to be deployed in a suitable runtime environment. We will also see how UOLs can be enhanced and in turn be reused in the PCM closing the editing cycle. In this way a gentle introduction to LD authoring can be achieved using the PCM as an initial more loosely authoring environment. In later stage LD can be used to capture, enhance and redeploy the created learning experience when needed.

IMS Learning Design (LD) tools

LD is targeted at the educational designer allowing ‘learning designs’ to be explicitly modeled using semantically meaningful concepts from the pedagogical domain. Although expressive, the specification is also very complex due to the numerous language constructs, its declarative nature, and its fairly generic vocabulary (Griffiths & Blat, 2005; Olivier, 2004). However, LD was developed with a toolset in mind that would help the educational designer in using LD (Griffiths, Blat, García, Vogten, & Kwong, 2005b). Three years after the release of LD, a user community is established working on the development and enhancements of these tools. So far this has resulted in a toolset dealing with LD editing and authoring aspects on the one hand and runtime delivery aspects on the other hand (Griffiths et al., 2005b). These authors categorize the tools on two dimensions:

1. *Higher vs. lower level tools*: This dimension is related to the level of expertise in LD required from the user of the tool.
2. *General purpose versus specific purpose tools:* This dimension deals with the pedagogical scope of the tools. Specific purpose tools will hide complexity by translating generics into the specific context and filling in and leaving out optional elements where appropriate. Generic purpose tools however, will allow authoring and delivery of LD in all its glory.

Although efforts have been made to create or adapt specialized authoring tools with some success such as COLLAGE (Hernández-Leo et al., 2006a), HyCo-ALD (Berlanga & García, 2007) and MOT+ (Paquette, De la Teja, Léonard, Lundgren-Cayrol, & Marino, 2005), in general most of the available tools that are LD compliant on levels A, B and C must be categorized as generic and still rather low level. They allow the editing of the complete LD specification and keep very close to the specification. A typical example in this category is Reload (Reload, 2007; Milligan, Beauvoir, & Sharples, 2005) which is by far the most popular LD editor at the moment. However, as a consequence, an ample understanding of LD is required to use these tools. An even more profound understanding of LD is required when advanced concepts as described by levels B and C of LD, are required by the design. In general, this level of understanding is limited to expert educational designers and is rarely found in practitioners such as teachers. This leaves many practitioners out of the direct loop of designing and adapting UOLs. Some LD tools available allow limited post design runtime adaptations through code introspection (Zarraonandia et al., 2006). However, these post design runtime adaptations will not be reflected in the UOL and therefore will be lost in the next run (Tattersall et al., 2005a) of the same UOL.

Furthermore, the current toolset imposes a, be it an implicit, top-down approach of the overall design and delivery process. This is further encouraged by the separation of the authoring environments and runtime delivery environments (Tattersall, Vogten, & Koper, 2005c). Typically, elicitation and selection of the type of educational scenarios is the first step in the design process followed by the coding of the scenario into a UOL using the authoring environment. Next this UOL is published so it can be delivered to teachers and learners via a runtime environment such as CopperCore (Martens et al., 2004). This UOL can be adapted, refined and improved in following design cycles repeating the whole process again. This workflow resembles the waterfall approach of traditional software development and has advantages especially in cases where the same UOL is offered to different groups for lengthy periods of time (Tattersall et al., 2005c). This approach can help enhance the quality of the learning experience because educational scenarios are made elicit in a very explicit and formal manner allowing reflection on the quality and effectiveness of the designs. This quality control can be further enhanced by collecting runtime data as is demonstrated in ALFANET (Van Rosmalen et al., 2007). Concluding it can be said that with the current toolset practitioners must adopt this top-down approach and need to have ample knowledge of LD. Therefore, LD has been taken up mainly by institutions for higher education where the required expertise can be found.

In the following sections we will present the TENCompetence domain model (Koper, 2006) followed by the first implementation based on this domain model called the PCM. We will discuss how the PCM can complement the current toolset available for LD. We will discuss how the PCM empowers individual users to create basic UOLs using a bottom-up approach without the need for any specific LD expertise. Furthermore we will discuss how these UOLs can be fed back into the PCM allowing a more controlled and reproducible provisioning of the learning process.

TENCompetence Domain Model

The aims of TENCompetence have been defined on the web site (TENCompetence consortium, 2007) as:
“A competence-based approach to lifelong learning aims to take account of all the informal and experiential learning that an individual acquires during the course of his or her lifetime rather than focusing solely on academic or theoretical achievement. This way an individual can make the most of his or her achievements, be they scholastic, work-based or the result of a leisure pursuit. The concept of competence development bridges the worlds of education, training, knowledge management, human resource management & informal learning in all domains which, hitherto, operated in relative isolation in respect of one another. A competence approach to lifelong learning ensures that the pursuit of a learning goal does not happen in a vacuum, but instead is bound to a precisely defined purpose such as an occupation, a profession, a market or a particular life or work situation.”

TENCompetence is finding solutions for seven major problem areas (Koper & Specht, 2007) currently preventing an infrastructure for lifelong competence development to become a reality. TENCompetence is focusing at the needs of the individual lifelong learner that want to maintain their autonomy and control as much as possible. This aspect of user empowerment is typical for initiatives in the area of Personal Learning Environments (CETIS, 2007). Users are expected to develop their own competences, not merely by taking up competence development courses, but also by actively contributing to these courses.

Before discussing the TENCompetence domain model we have to give our definition of a competence. We define a competence as the estimated ability of an actor to deal with certain critical events, problems or tasks that can occur in a certain situation. This estimation can be based on: self assessment, informal assessments by others, formal assessments by others or automated assessments. Competences can be attributed to an individual person, but also to a team or to an organization. We will use the term actor as a container for individuals, teams or organizations. Dealing with these critical events, problems or tasks requires a number of different competences. This set of required competences is called the competence profile (CP). Actors will develop and maintain many competences during their lifetime and these competences can be considered dispositions of these actors. A competence is a highly situational concept meaning that the definition and understanding of a competence is attributed to the relationship between actor and environment. Some of these competences are highly specific and others are transferable to more general situations. The specific labels we give to competences and CPs are determined by a community of practice that consists of all participants who are regular actors in that situation. Therefore, the competences for the same profession, job or function may vary from community to community even though the required behaviors are exactly the same. Finally, a competence is a latent characteristic of an actor: it is neither directly visible nor measurable. Only the concrete performances of actors are visible. From these performances we infer these latent characteristics and get an idea of the competences these actors have acquired.

The TENCompetence domain model is the conceptual model for lifelong competence development and it describes the various entities and their relationships that play a role. The domain model is informed by our definition of competences, by the principles of LD and finally, by the concepts of learning networks (Koper, 2005b).

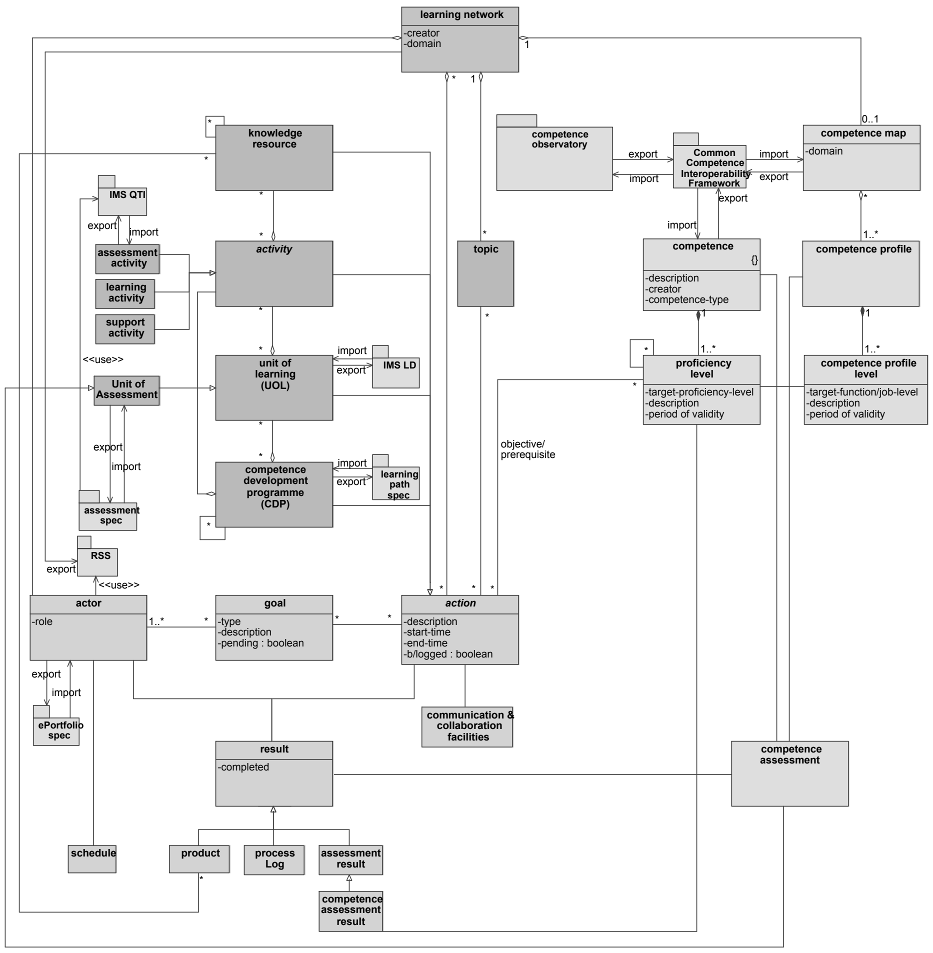


Figure 7.1 UML Class Diagram of the TENCompetence Domain Model

Figure 7.1 depicts the UML (OMG, 2003) class diagram of the TENCompetence domain model. The model is divided into four separate modelling areas: learning materials, actor performance, competence model and finally the learning network, or community of practice, as a container for all these concepts. The domain model will now be elaborated in more detail through these concepts and their relationships.

Actors will perform actions in order to achieve their goals. Typical goals are: keeping up-to-date with a profession; improving particular competences; comparing competences with peers. These actions are always performed in the context of a community of practice which is represented by a learning network in the domain model. While performing actions, actors have the possibility and are stimulated to provide support to each other by means of communication and collaboration facilities.

By performing the actions actors leave traces of their performance behind. These traces can take many forms ranging from mere activity logs to learning outcomes. These traces will be used to infer the measure in which an actor has acquired certain competences. Because competences are highly situational concepts their definitions are specific to the learning network. Competences can be acquired at different levels. These levels are modeled via proficiency levels, each representing a discrete ordinal measure to which a competence has been acquired. Competences can alternatively and/or additionally be assessed through specific competence assessments.

A CP is a collection of competences, targeted at specific proficiency levels which are required to be able to deal with certain critical events, problems, or tasks in a certain situation. CPs can be further split up into CP levels representing the levels of a profession, for example, like trainer, master and trainee.

Each learning network will define and describe its own set of competences and CPs. This set makes up the competence map of that community of practice. Some of these competences are generic and/or common to a domain but merely described differently for a particular community. A competence observatory will maintain the common and more formal definitions and descriptions for these generic competences ensuring transferability between communities of practice. Communities may contribute their competences and CPs to this observatory and thereby share their definitions with other communities. Equally communities may decide to reuse competences and CPs present in the competence observatory.

Finally, the model for actions is informed by the concepts of learning design. Actions can be divided into: knowledge resources, activities, UOLs, and competence development programmes (CDPs). A CDP is an ordered set of activities and UOLs that have to be mastered to attain a certain competence or CP. CDPs can be exported to a learning path specification. We will see how the PCM, besides using LD as formalism for learning design which is quite natural, also uses LD as formalism for this learning path specification.

The Personal Competence Manager

The PCM is a client server application implementing a simplified version of the TENCompetence domain model. The PCM lets users manage their own competence profiles in the context of learning networks for which they are registered. These competence profiles can be used to reflect on their personal competences with respect to this profile. The PCM helps users find most suitable learning materials and learning opportunities for acquiring these competences. Furthermore the PCM encourages users to create and share their personal contributions with the rest of the community. For this purpose design and runtime are closely integrated in the PCM. The PCM does not work with concepts like releases or versions and the learning opportunities are continuously changing and hopefully thereby improving. This is very much in contrast with the top-down approach supported by the current LD toolset.

At the time of writing of this article the design stages have been concluded and coding of the PCM has started. The software is available as open source on SourceForge at: http://sourceforge.net/projects/tencompetence/. Figure 7.2 depicts the overall architecture for the PCM. The PCM is developed as a desktop client application using the Eclipse Rich Client Platform (Eclipse, 2007) allowing it to run on a range of platforms. The client is extensible via the Eclipse plug-in framework. The client communicates with the server using REST (Fielding, 2000) providing an easy to use interface for other clients in the future. The PCM server is deployed on a Tomcat application server. It provides several services which are governed by a servlet handler which in effect is acting as a simple service bus. The server core provides basic provisioning and query services for the data model objects we already encountered in the TENCompetence domain model.

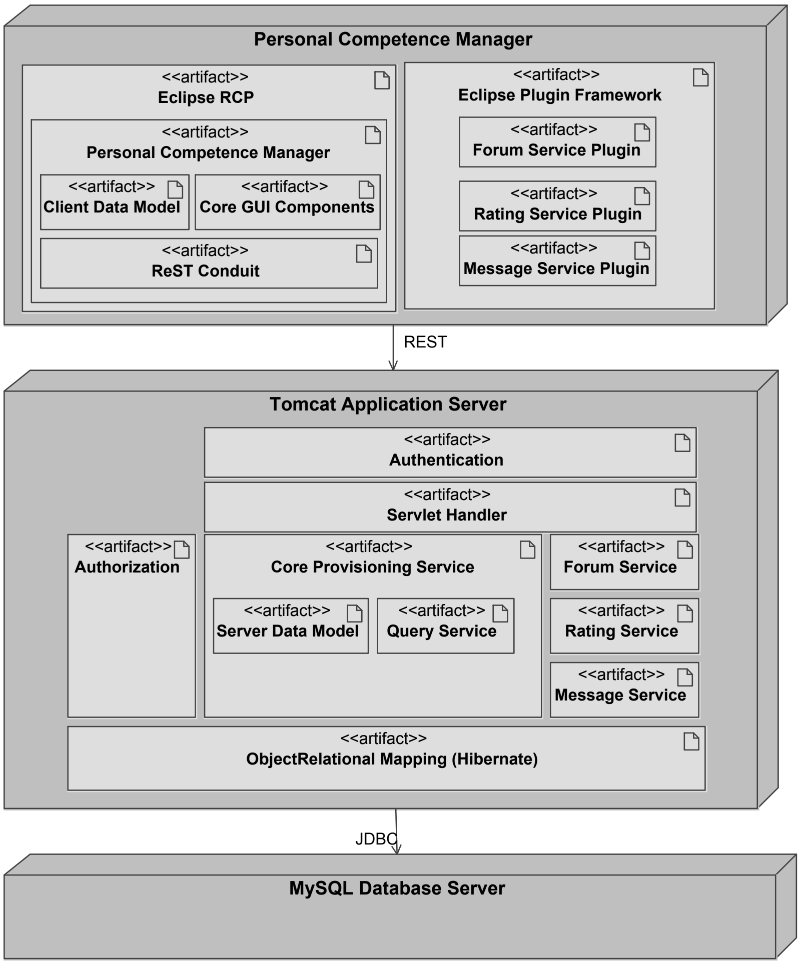
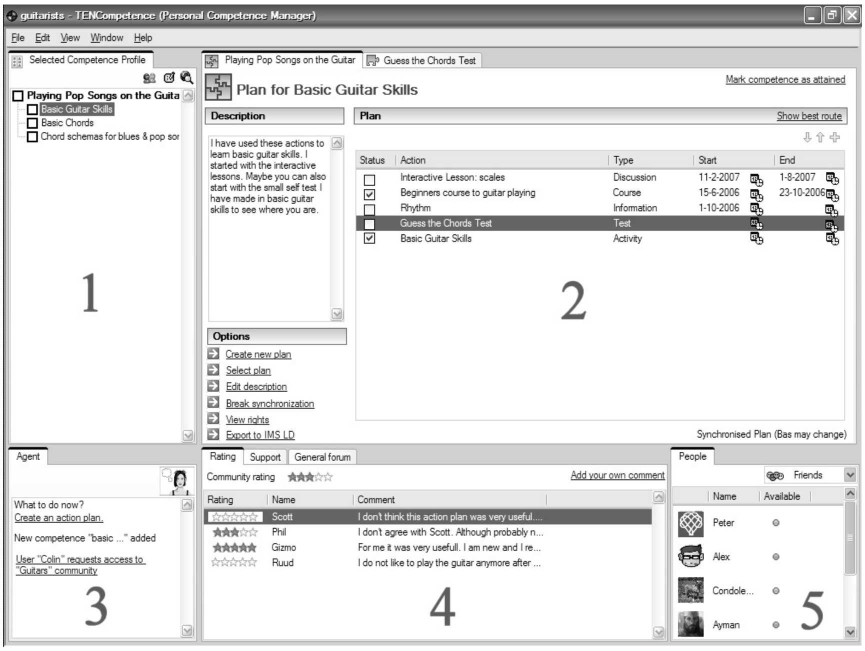

Personal Competence Manager
<<artifact>> Eclipse RCP
<<artifact>> Personal Competence Manager
<<artifact>> Client Data Model
<<artifact>> Core GUI Components
<<artifact>> ReST Conduit
<<artifact>> Eclipse Plugin Framework
<<artifact>> Forum Service Plugin
<<artifact>> Rating Service Plugin
<<artifact>> Message Service Plugin
REST
Tomcat Application Server
<<artifact>> Authentication
<<artifact>> Servlet Handler
<<artifact>> Authorization
<<artifact>> Core Provisioning Service
<<artifact>> Server Data Model
<<artifact>> Query Service
<<artifact>> Forum Service
<<artifact>> Rating Service
<<artifact>> Message Service
<<artifact>> ObjectRelational Mapping (Hibernate)
JDBC
MySQL Database Server

Figure 7.2 PCM Architecture.

Besides the core service, a number of additional, more autonomous services are provided by the server such as the forum, rating and message services. The idea is that these services will be extensible in future releases. Access to these services is governed by an authorization module. Finally, data persistence is managed through an object relational mapping using Hibernate.

The core functionality of the PCM will be discussed using detailed screen designs that were available at the time of writing of this article.


guitarists - TENCompetence (Personal Competence Manager)
File Edit View Window Help
Selected Competence Profile
Playing Pop Songs on the Guita
Basic Guitar Skills
Basic Chords
Chord schemas for blues & pop sor
1
Playing Pop Songs on the Guitar
Guess the Chords Test
Plan for Basic Guitar Skills
Mark competence as attained
Description
Plan
Show best route
Options
Create new plan
Select plan
Edit description
Break synchronization
View rights
Export to IMS LD
Synchronised Plan (Bas may change)
2
Agent
3
Rating
Support
General forum
Community rating
Add your own comment
4
People
Friends
5

I have used these actions to learn basic guitar skills. I started with the interactive lessons. Maybe you can also start with the small self test I have made in basic guitar skills to see where you are.

Status	Action	Type	Start	End
☐	Interactive Lesson: scales	Discussion	11-2-2007	1-8-2007
☑	Beginners course to guitar playing	Course	15-6-2006	23-10-2006
☐	Rhythm	Information	1-10-2006	
☐	Guess the Chords Test	Test		
☑	Basic Guitar Skills	Activity		

What to do now?
Create an action plan.

New competence "basic ..." added

User "Colin" requests access to "Guitars" community

Rating	Name	Comment
☆☆☆☆☆	Scott	I don't think this action plan was very useful....
★★★☆☆	Phil	I don't agree with Scott. Although probably n...
★★★★★	Gizmo	For me it was very usefull. I am new and I re...
☆☆☆☆☆	Ruud	I do not like to play the guitar anymore after ...

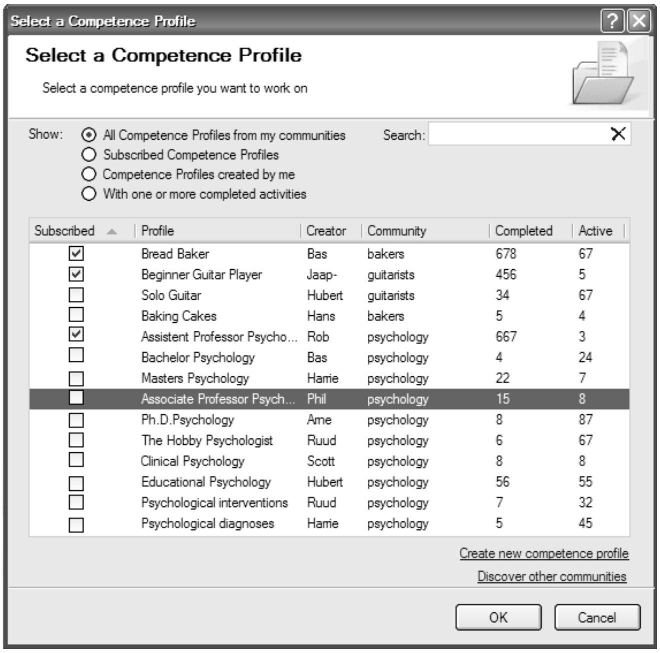
Name	Available
Peter	●
Alex	●
Condole...	●
Ayman	●

Figure 7.3 Screenshot of the PCM user Interface design

Figure 7.3 depicts the main application window of the PCM. The PCM user interface can be roughly divided into two areas. The top half area (1 and 2) contains views and editors intended for viewing and editing CPs, competences, and actions. The lower half of the main window (3, 4 and 5) contains views that help and support the users in their task performed in the upper half. In figure 7.3 the 'Plan for Basic Guitar Skills' is the active editor (area 2) and therefore provides the context for all views in the lower part of the screen.

Figure 7.3 represents a snapshot of a situation where a learning network, in the PCM represented by its synonymous term community, already has been created and some content has been added to this network. Furthermore, any user may decide to start a new learning network at any moment in time. Learning networks are not governed by any central authority and can be set up by anyone. The creator of a learning network is also the owner of the community and determines policies for the learning network access. This principle of an entity owner controlling its access rights applies for almost all entities. The general idea is that the PCM should tend to openness whenever possible in order to stimulate active participation and contributions of all community members. The PCM relies on the principles of self-organization to regulate this process (Hadeli, Zamifirescu, van Brussel, Holvoet, & Steegmans, 2003).

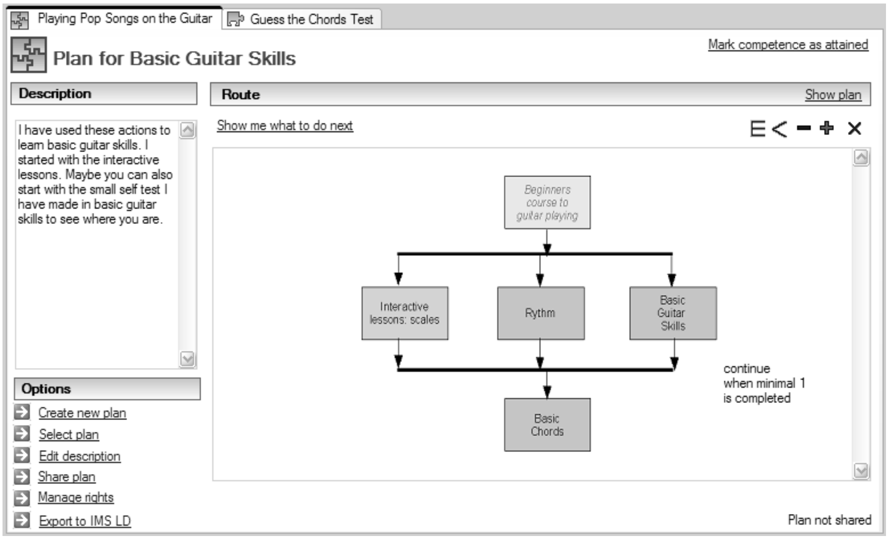
View 1 of figure 7.3 shows the CP selected by the user. A user can select CPs via the competence selection dialog shown in figure 7.4.


Select a Competence Profile
Select a Competence Profile
Select a competence profile you want to work on
Show:
(●) All Competence Profiles from my communities
( ) Subscribed Competence Profiles
( ) Competence Profiles created by me
( ) With one or more completed activities
Search:
Create new competence profile
Discover other communities
OK
Cancel

Subscribed	Profile	Creator	Community	Completed	Active
☑	Bread Baker	Bas	bakers	678	67
☑	Beginner Guitar Player	Jaap-	guitarists	456	5
☐	Solo Guitar	Hubert	guitarists	34	67
☐	Baking Cakes	Hans	bakers	5	4
☑	Assistent Professor Psycho...	Rob	psychology	667	3
☐	Bachelor Psychology	Bas	psychology	4	24
☐	Masters Psychology	Harrie	psychology	22	7
☐	Associate Professor Psych...	Phil	psychology	15	8
☐	Ph.D.Psychology	Arne	psychology	8	87
☐	The Hobby Psychologist	Ruud	psychology	6	67
☐	Clinical Psychology	Scott	psychology	8	8
☐	Educational Psychology	Hubert	psychology	56	55
☐	Psychological interventions	Ruud	psychology	7	32
☐	Psychological diagnoses	Harrie	psychology	5	45

Figure 7.4 CP selection dialog

Once the profile has been selected, the user may access the competence development plans for these competences. These will be opened in the CDP editor depicted in figure 7.3 (area 2). For any competence many CDPs may exist. The CDP is a container for a number of actions that represent a learning design targeted at the associated competence. A user may decide to simply start performing one of these actions by selecting them from the list, but can alternatively also decide to get some advice about the best next actions to take by clicking the ‘Show best route’ link. The PCM will now show a flow chart like navigational view of the CDP revealing the relations between the actions of the CDP.



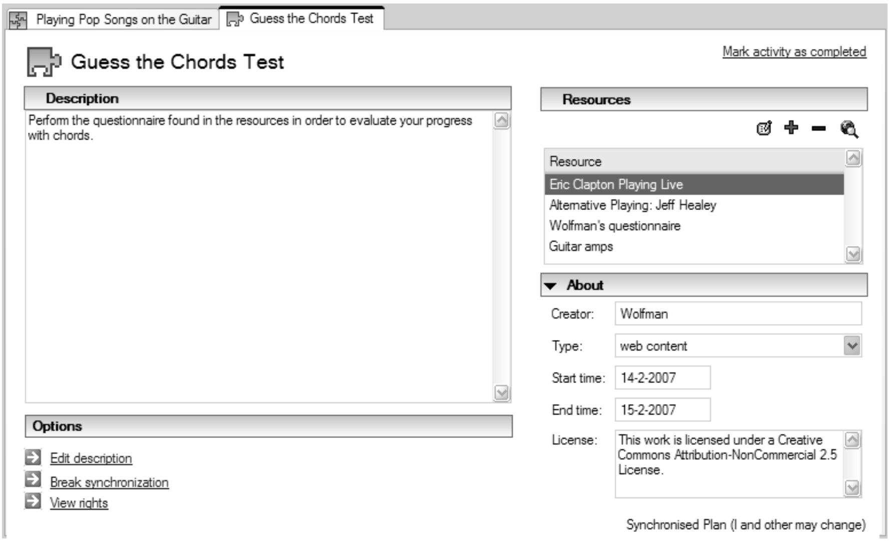
I have used these actions to learn basic guitar skills. I started with the interactive lessons. Maybe you can also start with the small self test I have made in basic guitar skills to see where you are.

Figure 7.5 Navigation view.

Figure 7.5 depicts this navigation view of the CDP. Actions in the CDP can be structured into sequences and selections. These concepts are very much informed by LD. By clicking ‘Show me what to do next’ the user activates the navigation service to receive help in selecting the best next action. In the first release of the PCM this navigation service will be implemented using the simplest of algorithms possible: suggest the next action which is not yet completed, but needs completing. In the future advanced navigation services will be available that also take personal preferences, learning styles and past performances of others into account.

Users can actively contribute to a CDP by adding new actions or modifying the detailed learning path as shown in figure 7.5. By applying these changes to a CDP the user is sharing the changes with others. A shared CDP is behaving like a Wiki with regard to this sharing aspect. Alternatively, a user may decide to create a different CDP for the selected competence all together. This CDP will show up as alternative when another user is selecting a CDP in order to acquire this competence.

When a user decides to perform an action from the CDP the action editor depicted in figure 7.6 is opened.



Guess the Chords Test

Description: Perform the questionnaire found in the resources in order to evaluate your progress with chords.

Resource:
- Eric Clapton Playing Live
- Alternative Playing: Jeff Healey
- Wolfman's questionnaire
- Guitar amps

Creator: Wolfman
Type: web content
Start time: 14-2-2007
End time: 15-2-2007
License: This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License.

Synchronised Plan (I and other may change)

Figure 7.6 Action editor

The concept of an action was also informed by LD. Actions can take two forms: a link to an external implementation like for instance a link to a run of a UOL, or an action that is managed by the PCM itself. An action has a description instructing the learner what he is expected to do. Furthermore there are resources available helping the learner to perform this action. An action can be modified by changing the description and/or by modifying the resource associated with it.

The bottom half of figure 7.3 that is composed out of area 3, 4 and 5 contain services that will help the users in performing their tasks. The agent view (area 3), informs the user about events occurring in the community. Next (area 4) there is a group of services that are helping the user to perform the selected action (area 2) which consists of a rating service, a support forum, and a general discussion forum. Finally, there is a member services showing all the members of the community. The PCM will support FOAF (FOAF project, 2007) to support the creation of *ad hoc* user communities. The PCM may be extended with additional services via the standard plug-in mechanism provided by Eclipse.

Capturing the Competence Development Plan using LD

We have seen that the TENCompetence domain model and therefore also the PCM are informed by LD, especially the part dealing with learning materials. Concepts such as learning activities, support activities, learning resources and UOLs can be directly mapped onto concepts defined in LD. Furthermore, competences themselves can be mapped through LD prerequisites and objectives. Although this mapping may seem not that obvious at first, LD started out as a specification for modelling competence based learning (Tattersall et al., 2005b). In the LD specification references are made to the 'IMS Reusable Definition of Competency or Educational Objective' specification (IMS RDCEO, 2002) for both the prerequisites and objectives sections. Finally, the CDP brings all these components together and can be mapped onto the method section of LD. The CDP consists of a simple list of actions that may be performed by the user. This list can be mapped directly to a selection in LD. In more advanced designs of the learning path within the CDP there can be a mix of selections and sequences of actions. These constructs map directly onto the selections and sequences as defined in LD. So all CDPs main constructs can be mapped to equivalent LD constructs. Table 7.1 depicts the global mapping of the main entities found within the TENCompetence domain model onto the LD elements. Note that most elements have a direct one-to-one mapping with the exception of the CDP, which requires a more elaborate mapping because it provides the container for all other elements.

Table 7.1 Translation of main TENCompetence Domain Model entities

TENCompetence domain model entity	IMS Learning Design element(s)
knowledge resource	learning-object
learning activity	learning-activity
support-activity	support-activity
assessment-activity	learning-activity with IMS Question and Test Interoperability content.
unit-of-learning	No mapping required because this is a place holder for the UOL itself. This allows a UOL to be fed back into the PCM.
Competence	prerequisite or learning-objective
CDP	unit of learning containing one learner role, the competences addressed by the associated CP expressed as objectives, selections and sequences as defined by the learning path of the CDP and a play for wrapping the activities.

The user can initiate the transformation by clicking the 'Export to LD' option. The resulting UOL can be stored for publication or if needed, for further refinements and enhancements.

Just as important as the data model entities themselves, is how the manner in which they are created. We have shown via the wire frames that editing a CDP and its components can be done without any knowledge or awareness of LD whatsoever. The PCM does not presume any particular workflow and allows a bottom-up approach because no distinction is made between design time and runtime. Via the principle of "what you see is what you get", the PCM allows the active participation of learners in the creation of educational materials and scenarios. A learning design can become an emergent property of the work of a whole community. At any point in time a user may decide to capture the outcomes of this process in the form of a UOL by performing an export. The reasons for doing so can be numerous like being able to:

- reflect on the quality of the learning design which can be achieved more easily now because the design is made explicit and formal;
- reuse the same materials for another group of learners making the learning experience reproducible;
- improve the design by adding more sophisticated features adding a great deal of extensibility and flexibility to the PCM;
- share the design with other practitioners who could be using other e-learning environments (LD provides this interoperability);
- capture a design as a permanent record for the learning experience provided, this record in the form of a UOL can provide accountability independent of a particular version of particular software.

The PCM uses LD as an export format for its CDPs. The exported UOL only captures parts of the functionality offered by the PCM because it is merely a snapshot of the design modelled through the CDP, not of the process that has lead to it. The context in which the CDP has been created, like the groups discussion, ratings of alternative CDPs, creation of *ad hoc* communities working together on the topic, building of reputations of users within the community, and so forth is not captured by the resulting UOL. Also personalized data such as the planned start and end dates for activities are not captured in the UOL because LD specifies a learning design at the level of user roles rather than at the level of an individual. This is also the reason that a UOL needs to be populated through the run mechanism before it can be deployed: the personal information has to be added by the runtime engine in order to deliver the design.

The example depicted by figure 7.3 and figure 7.5 would result in the following LD fragment, which has been greatly simplified for readability purposes.

```
<learning-design>
  <title>Plan for Basic Guitar Skills</title>
  <learning-objectives>
    <item identifierref="basic_guitar_skills"/>
  </learning-objectives>
  <components>
    <roles>
      <learner identifier="learner"><title>Learner</title></learner>
    </roles>
    <activities>
      <learning-activity identifier="a_beginners_course_guitar">
        <title>Beginners course guitar playing</title>
      </learning-activity>
      <learning-activity identifier="a_interactive_lessons:_scales">
        <title>Interactive lessons: scales </title>
      </learning-activity>
      <learning-activity identifier="a_rhythm">
        <title>Rhythm</title>
      </learning-activity>
      <learning-activity identifier="a_basic_guitar_skills">
        <title>Basic Guitar Skills</title>
      </learning-activity>
```

```
        <learning-activity identifier="a_basic_chords">
          <title>Beginners course guitar playing</title>
        </learning-activity>
        <activity-structure identifier="seq_1"
                            structure-type="sequence">
          <learning-activity-ref ref="a_beginners_course_guitar" />
          <activity-structure-ref ref="sel_1"/>
          <learning-activity-ref ref=" a_basic_chords "/>
        </activity-structure>
        <activity-structure identifier="sel_1"
                            structure-type="selection">
          <learning-activity-ref ref="a_interactive_lessons:_scales" />
          <learning-activity-ref ref="a_rhythm"/>
          <learning-activity-ref ref="a_basic_guitar_skills"/>
        </activity-structure>
      </activities>
  ..</components>
    <method>
      <play>
        <act>
          <role-part>
            <role-ref ref="learner"/>
            <activity-structure-ref ref="seq_1"/>
          </role-part>
        </act>
      </play>
    </method>
  <learning-design>
```

The translation of the constructs in the PCM has been fairly straightforward according to the rules described in table 7.1 All exported CDPs have such a fairly basic learning design because the possibilities to vary this design are relatively limited compared to the modelling possibilities and freedom offered by LD.

The exported UOL can be edited with all available LD authoring tools, enhancing the design where needed. These tools allow more sophisticated editing of the UOL because they make all constructs of LD available to the user. However, this also implies that from this point onwards ample LD expertise is required to maintain the UOL. An enhanced design can be fed back into the PCM by creating a new action that wraps this UOL. The PCM integrates the CopperCore (Martens et al., 2004) LD runtime environment in order to deploy the modified UOL. Without this integration reuse of the enhanced UOL with in the PCM would not be impossible because the PCM would not be capable of interpreting the enhanced design itself. This also implies that once a UOL has been enhanced it can only be re-edited via the regular LD tools.

This action that wraps the exported UOL, can replace the original CDP because its learning objectives are targeted towards the same competence as the CDP it was derived from. The action containing referring to the UOL could also be included into a bigger CDP which in turn could be exported to another UOL resembling the Russian dolls model. This way the bottom-up authoring approach provided by the PCM can be integrated with the more formal top-down design approach associated with current LD authoring environments, providing the best of both worlds. Figure 7.7 depicts this editing cycle.

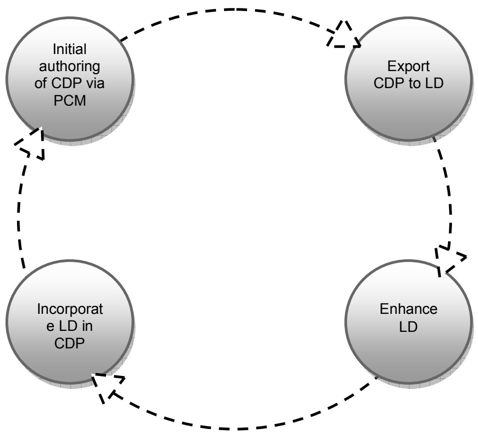


Figure 7.7 The editing cycle

In order for the round-trip editing cycle to succeed, a specific deployment approach for the exported UOL has to be chosen. Because the PCM relies on the *ad hoc* formation of communities per CDP, the resulting runtime delivery of the UOL should adhere to these communities as well. The proper integration of the PCM and the CopperCore runtime engine is crucial because the CDP membership and the UOL run subscriptions have to be kept synchronized at all time. Therefore exactly one run will be created of a UOL for every CDP containing that UOL. Users are added and removed from a run in accordance to their registration for the containing CDP. So *de facto*, the CDP population and the run population are kept in sync. For this first release of the PCM it is assumed that the UOL will allow users to be “rolled on” and “rolled off”. It is however possible to use LD constructs that forbid this type of continues registration by forcing users to be added in cohorts. These restrictions will simply be ignored in the first release of the PCM and need further investigation in the future.

Because the exported UOL is wrapped with its own action when it is imported in the PCM, all regular support tools such as ratings and forums and self assessments are available when executing the UOL. Therefore, there is no need to synchronize outcomes of the CopperCore runtime engine with the PCM. However, in future releases, this could be the case. The CopperCore Service Integration framework (Vogten & Martens, 2006) provides a first direction towards a closer integration when the need should arise in future release of the PCM.

The assignment of roles is another issue that needs to be resolved for the editing cycle of figure 7.7 to work. In LD, users can fulfill multiple roles in one design. A user needs to be assigned to one or more of such roles before the user can actively participate in a run. In those cases where a UOL is merely exported and not modified, this assignment is simple and can be done without any additional actions because there will be only one role defined in the exported UOL as we have seen a few paragraphs ago in the simplified example. However, when the generated UOL is enhanced it is perfectly reasonable to have a more complex role structure. When the role mappings are the same for every user this is no real problem because the role assignments can still be handled automatically. However, when the design assumes users to take on different roles, the mapping is not that straightforward anymore. Intervention by a user or intelligent role mapping services may be required in those cases. For the first release of the software, simple mappings are assumed by default and user interaction is required for these more complex situations. For future releases this is an issue that needs further exploration.

Conclusion and future work

In this paper we have argued that LD is a very generic, complete and therefore also a complex specification. For a non specialist the use of LD in the daily teaching practice is only feasible with the help of sophisticated and probably specialized tools. The current state-of-the-art LD tools can be categorized as generic and LD aware, requiring a specialist's expertise. Furthermore, an external data representation such as LD, leads to a natural separation of design time and runtime tooling. This in turn introduces a top-down workflow approach to provisioning of learning through consecutive stages of design, authoring, publication, user management, and finally delivery.

Although this is a perfectly sound approach, it can also be problematic in cases where practitioners prefer a more bottom-up approach without having a very elicit view on the design. These practitioners will probably prefer an environment where there is no strict separation between design time and runtime. This approach is often more appealing, intuitive, and suitable for the initial stages of a design. The PCM provides this type of editing. Especially the CDP editor provides an easy means for creating a learning design that is built up from actions which in turn can be organized into sequences and selections. In a later stage, especially when a design has matured and proven to be particularly successful, there may be a need to redeploy the same design for a different group of learners. The *ad hoc* design can be exported to a UOL making the design formal. Other reasons for exporting the design could be the need to reuse the same design with other resources. It could also be the case that it would be worthwhile to redeploy the same design in a totally different e-learning environment. Quality assurance could be another reason for formalizing an *ad hoc* design into a UOL. The exported basic designs can be improved upon with the normal LD tools and then be reused in the context of the PCM itself or by any other LD compliant environment. The PCM integrates LD tools such as CopperCore for this purpose.

The approach presented in this paper allows for an easy introduction of users to LD in cases when there are clear benefits for the user to do so. The generated LD can be used as it is, but can also be improved upon. Whatever is the case, the user will need ample LD knowledge from that point onwards. Nevertheless, the user has a clear motivation, one of the aforementioned reasons, to make the additional effort needed to become familiar with LD. Although the user can feed back the altered UOL into the PCM, once exported and modified, the point of no return has been passed. The PCM will not be able to help the user maintain the UOL. The reason for this is that advanced LD concepts have no equivalent in the PCM such as, for example, support for advanced personalization, support for different pedagogies, support for multiple roles and support for advanced role based workflow. Therefore, the PCM will never be able to really replace the existing LD tools but must rather be considered to offer a gentle introduction to LD for those practitioners who are new to the tools and concepts and do not have or see a need to invest in them right from the start.

When exporting the CDP to a UOL two distinct approaches can be defined. First, the one discussed so far, where the produced UOL is reused in the context of the CDP. This export may assume that the services offered by the CDP will be available to the UOL as well because the UOL is reused in the same context. However, if the UOL is reused in a totally different context from the CDP, another type of export may be required because referenced and implicit services have to be defined and bundled in some form into the UOL. Although initial steps have been taken in this direction with, for example, the integration of assessment services through IMS Question and Test Interoperability (IMSQTI, 2006; Vogten et al., 2007), there is still further work to be done in this area especially regarding the standardization of service interfaces. For now the PCM will only support the first type of LD export requiring the PCM to run the constructed UOL.

At the moment of writing several initiatives are improving on the available LD tools. In fact some of these initiatives have been bundled in the TENCompetence integrated project (TENCompetence consortium, 2007). It will be interesting to see how these tools develop and what this means for the integration in the PCM. A first step towards this integration is the harmonization of the look and feel of both the CDP editor and the Reload based LD editor. Work towards this direction has recently started and although at the time of writing this development is still very much in its early stages it looks like a promising step towards a more seamless integration of the PCM and LD.

Until that time, the approach presented in this article combining the implicit bottom-up design method provided by the PCM and the more formal elicit top-down design favoured by the current LD toolset offers a practical alternative.

Chapter 8

Review of Results and further Research and Development

Introduction

When the LD specification was officially released there was a real need for a reference implementation to help practitioners to better understand the specification. System integrators would be able to experiment with the integration of LD in their systems, and system developers would benefit from a reference design that demonstrated how an LD runtime environment can be built.

Designing and implementing a runtime environment of this sort is in no way straightforward. LD combines characteristics from different languages; for example, declarative languages. We have seen that this is especially true for the LD conditions, which have similarities with production rules. It is also declarative in a more semantic sense: it requires much scaffolding from the runtime, as is the case, for example, with some of the services defined by LD. It is also a persistent language, meaning that the runtime is expected to automatically take care of persistence. LD shares some characteristics, too, of an imperative programming language in which statements are given in the order they are to be executed in. Finally, it resembles a workflow language, orchestrating the learning processes between the different learning and support roles.

Given these combined characteristics, implementing a runtime environment requires considerable resources and effort, even when provided with a solid design. To give an idea of the magnitude of such an investment, the number of code lines can be used as a reference (Albrecht & Gaffney, 1983; Rosenberg, 1997): for the CopperCore implementation this results in 30,000 lines of Java code. Although not a huge amount it does represent considerable effort, particularly as this is only the code for the engine. By comparison, the Apache HTTP 2.0.x involves 90.000 lines of code (M Squared Technologies, 2008). Any implementation, therefore, should be reusable in many different situations to make costly rebuilds less necessary. This may ease the uptake of LD because it can be used without having to invest in engine development.

LD itself relies on other specifications and learning services. Although it comes with a fairly detailed description of how to incorporate these specifications and services in the learning design at a lexical level, very little is stated about the runtime implications. This situation led to the two research and development questions addressed in this thesis.

i) *How can a fully compliant reusable reference runtime environment for the IMS Learning Design specification be designed and implemented?*
ii) *How, given a reference implementation for the IMS Learning Design specification, can implementations for other e-learning*

specifications and learning support services be integrated generically at runtime level?

Review of the results

In chapter 2 we defined the LD engine as a software component for processing LD's business rules for any UOL. The engine supplies input for an LD player responsible for rendering the results of the engine in a form presentable to a user. The first question of this thesis addresses a reusable reference design and implementation of an LD engine. Starting with the more in-depth analysis of LD in chapter 2, we identified seven categories of requirements that must be met by an LD engine: validation, parsing, publishing, provisioning, population, personalization and integration. We now review our results by discussing how CopperCore has dealt with these requirements.

VALIDATION

CopperCore provides validation in two distinct stages. The first is validation of the structural soundness of a UOL package and the resources within it; in chapter 2, we saw how such a package is constructed. The second stage involves more semantic validation of the package by checking the constraints imposed on the learning design as defined by LD. CopperCore provides this validation as an integral part of the parsing process.

CopperCore validates the UOL's structural soundness in four consecutive steps. In the first step, the UOL content is unzipped while ensuring that the UOL contains an 'imsmanifest.xml' file. The second step involves validating this manifest file against the appropriate XML LD schema, which can be level A, B or C. If this validation is successful, the learning design contained in the manifest is parsed and all references checked, ensuring that each reference refers to a valid LD entity. This step is important because XML schema does not support this type of validation. Furthermore, any referential recursion could lead to unexpected results later on in the process. The third step comprises the validation of files of type 'imsldcontent'. These can contain references to the properties and activities in the learning design which must be checked. Finally, the fourth step consists in cross checking all referenced resources in the manifest against the files contained in the UOL. This results in errors whenever files are missing from the UOL but referenced in the manifest and vice versa this check will result in warnings whenever files are included in the UOL that are never referenced from the manifest. All validation results are collected in a log which is returned to the client. Validation continues even if errors occur, allowing the user to correct as many errors as possible before the next validation attempt. After the first validation stage, CopperCore has ensured the correctness of the UOL package, and is ready to start the parsing stage.

PARSING

CopperCore parses the UOL as part of the publication of the UOL. This parsing – and more specifically, the learning design within the UOL – starts by building up an internal data model of the learning design in-memory. Various data model classes represent equivalent LD elements. After the construction of this in-memory data model, the second, more semantic validation is performed. Most noteworthy is the validation of data types during this stage. CopperCore implements static, parse-time coercions checks for data types, and validation picks up on any illegal type conversions. We determined in chapter 2 that LD is unclear on this matter; we decided to let runtime predictability prevail over expressiveness in this case.

If no validation errors occur, the parsing process proceeds with generating property definitions for the FSM properties as described in chapter 3. Each data model class is capable of generating the appropriate and required definitions associated with that class. For example, a learning object will create an implicit, local personal property definition containing the content of the learning object as part of its definition. In similar fashion, property definitions are generated for explicit properties. Data model objects with a visibility or completion state will generate property definitions for these attributes as well. In addition, specific property definitions will be created for the activity tree and environment trees discussed in chapter 4. After this process is complete, all necessary property definitions to capture the state of a single FSM have been created. Furthermore, the learning design has been dissected into XML snippets stored as part of the property definitions. These snippets form the source for the content personalization during runtime.

After creating the property definitions, all explicit and implicit conditions are generated. These will be interpreted by the event handling mechanism discussed in chapter 3. Explicit conditions are simply defined in the learning design itself. Implicit conditions represent the business logic defined by LD, such as completion rules. The data model objects generate these implicit conditions using a slightly modified version of the LD expression language. All conditions are stored in a special property definition. Based on these conditions, an event table is built linking all potential triggers with the conditions to be evaluated. This table allows the event dispatcher described in chapter 3 to quickly decide which conditions should be evaluated after an event has occurred. If the antecedent of such a condition evaluates to true, the associated event handler will be launched and process the consequence of this condition.

PUBLISHING

CopperCore does not provide direct access to the validation and parsing stages through its API. Rather, this is done indirectly through the publication process. CopperCore first validates the UOL, making sure it is structurally sound. Next, the parser starts. If the publication is started in validation-only mode or if there have been validation errors, the publication process stops; in all other cases, it continues by persisting the outcomes of the parsing stage in a relational database. CopperCore will overwrite any existing property definitions, thereby effectively allowing republications of existing UOLs.

Although republication is a powerful feature of the engine, it must also be used with care, especially once a UOL has been populated with users. The type and restrictions of explicit properties could have been changed between UOL versions. If these properties have already been instantiated, their values could conflict with the new property type or the restrictions imposed on them. In these cases CopperCore will reset their values automatically to the defined default value or to ‘null’ if no default value has been defined. In this way, users could lose data as result of republication.

More important, existing property values can leave the engine in an unexpected state if the designer failed to consider them when changing the learning design. This state occurs, for example, when a learning design defines that an act is only completed if a property has the value 2. Suppose the same learning design only increments the value of this property when the value is 1, which also happens to be the default value: this design will not cause problems when populated with new users. However, suppose 0 was the default value in an older version of the same learning design, and that several users already have a 0 value for this property. These users would get stuck in their learning process once the new learning design was published, because the property value would never get to 2. Therefore, learning design authors have to be particularly careful when republishing a UOL, especially if they have modified the property declarations and/or conditions in their designs.

PROVISIONING

After successful publication, CopperCore provisions the resources contained in the UOL by creating a separate folder for each UOL in the root folder of a web server. All resources contained in the UOL are copied to this location. Republication simply overwrites the resources in this folder. The UOL itself is now ready to be provisioned through the creation of runs. In chapter 4 we described the CopperCore’s LDCourseManager API, which provides the calls necessary to create multiple runs for each UOL. CopperCore forces an evaluation of all defined conditions in the UOL once a user has been assigned to a run. This results in an initial seed of some of the properties, and allows CopperCore to process any tautologies that do not have a trigger. The following XML snippet is an example of such tautology.

```
<if>
  <is>
    <property-value>1</property-value>
    <property-value>1</property-value>
  </is>
  <then>
    <hide>
      <class class="answers" title="Show answers" with-control="true"/>
    </hide>
  </then>
</if>
```

This snippet will result in the hiding of all content with the class attribute 'answer' for each user when users are assigned to the run. However, the expected runtime behavior for this type of tautology is not clearly documented in the LD information model. Because the condition has no trigger, it will no longer be evaluated. We have therefore chosen to force evaluation of these conditions during run population.

POPULATION

Population involves the assignment of users to specific roles within the context of a run. CopperCore has an active role for each user in a run to establish which FSM to use when personalizing the UOL; it is therefore mandatory to set this active role. This makes it possible for passing merely two parameters, the run-id and user-id, to suffice for most calls to the LDEngine API (as shown in chapter 4).

After population, the CopperCore engine creates instances for each property according to the associated property definition. As we saw in chapter 3, this definition determines the instantiation scope and default value. Properties are created just in time, meaning that they are instantiated when referenced for the first time. This late binding ensures that the aforementioned republication issues are kept to a minimum. Properties are automatically persisted by the CopperCore engine. This persistence is controlled by a transaction manager, and new event loops begin by setting up new transactions. This ensures that the engine state remains valid even when an error occurs during event handling. In such cases all changes are reverted, effectively reinstating the last known valid state of the FSMs. In chapter 3, we concluded that the engine can be thought of as a collection of FSMs, and therefore automatically ensures that CopperCore meets all multi-user and multi-role LD requirements.

PERSONALIZATION

We argued in chapter 3 that personalization is simply a matter of transforming the generic XML snippets into personalized content by replacing references to implicit and explicit properties with their actual values as defined in the FSMs. This also implies that the FSMs have to be up to date at any given moment. We described the event handler mechanism and stated how each FSM is kept up to date by reacting to triggers and launching the corresponding event handlers. Therefore, using the concept of the event handler, CopperCore merely has to respond to triggers occurring during runtime to keep the FSMs up to date. In most cases these triggers are properties changing their value, but the elapsing of time and changing of roles can also be triggers. CopperCore has a special timer that raises a trigger after a predefined time. The period has to be fine enough to handle any reasonable learning design, but at the same time course enough to avoid overloading the engine with events. The timer is configurable by the system administrator, and set to 30 minutes by default. Each trigger can cause one or more rules to be fired, which in turn launches specific event handlers. These event handlers could result in the change of property values, which in turn could trigger other events.

CopperCore pays extra attention to recursions, as they could freeze up the engine. It does so in two ways. First of all, some types of recursion can be detected during the validation stage. Recursion may occur when resolving references in the UOL, as discussed in chapter 3, or when triggering events (a more subtle form of recursion). Consider the following LD snippet:

```
<if>
  <greater-than>
    <property-ref ref="int-prop" />
    <property-value>1</property-value>
  </greater-than>
  <then>
    <change-property-value>
      <property-ref ref="int-prop" />
      <property-value>
        <calculate>
          <sum>
            <property-ref ref="int-prop" />
            <property-value>1</property-value>
          </sum>
        </calculate>
      </property-value>
    </change-property-value>
  </then>
</if>
```

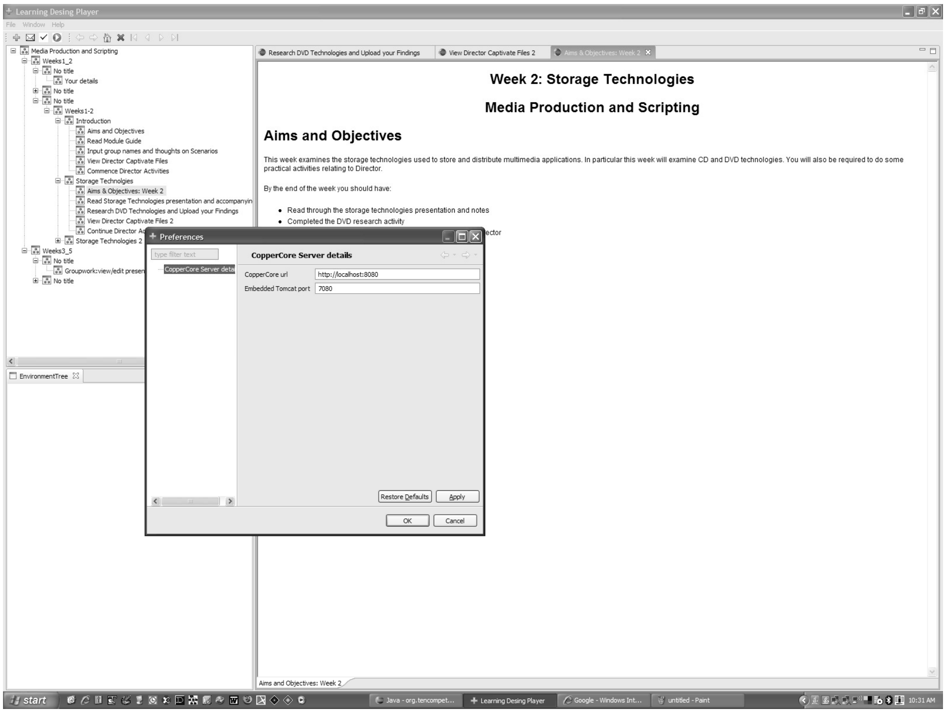
Once the value for property ‘int-prop’ is set to 1 or higher it increases the value of property ‘int-prop’ by 1, which in turn satisfies the condition again. If the engine did not intervene, it would get stuck in an infinite loop. CopperCore counters this by allowing each rule, and therefore also each condition, to be fired once in the lifetime of an event chain. Setting the value of property ‘int-prop’ to 1 would thus result in a value of 2 after completion of the event chain. The expected behavior, however, is not clearly documented in the LD information model, and our interpretation is mostly informed by practicality. Of course, this is not ideal and could result in interoperability issues.

The event handling mechanism with all its safeguards such as recursion detection and transaction management ensures that CopperCore contains only valid states for each of its FSMs. The next step of personalization is the transformation of the generic XML snippets stored in the property definition. This process is fairly simple and consists in merging property values from an FSM into the generic XML snippet by replacing the property references with their actual value. CopperCore performs the transformation of these XML snippets when they are fetched through the LDEngine API.

INTEGRATION

CopperCore was designed to be reusable from its very conception. This was achieved by clearly separating the engine from the player and providing two APIs, which give access to the engine. The APIs presented in chapter 4 have proven quite stable. Chapter 4 was based on work done in the early stages of CopperCore's development, at a time when it consisted in merely level A. Nevertheless, the API itself did not change at all when CopperCore progressed to become level-B and -C compliant.

The same APIs were extended with SOAP-compliant equivalents; the SOAP protocol effectively opened up use of the engine beyond the Java-based developments. We succeeded in building an experimental 'fat player client' for the engine with the eclipse RCP framework using the SOAP calls. Figure 8.1 shows a screenshot of this prototype. The ELeGI (ELeGI, 2007) project incorporated the engine in its .Net developments. Taken together, the fat client and ELeGI project demonstrate that the same engine can be used in different deployments and environments.


Learning Desing Player
File Window Help
Media Production and Scripting
Weeks1_2
No title
Your details
No title
No title
Weeks1-2
Introduction
Aims and Objectives
Read Module Guide
Input group names and thoughts on Scenarios
View Director Captivate Files
Commence Director Activities
Storage Technolgies
Aims & Objectives: Week 2
Read Storage Technologies presentation and accompanyin
Research DVD Technologies and Upload your Findings
View Director Captivate Files 2
Continue Director Ac
Storage Technologies 2
Weeks3_5
No title
Groupwork:view/edit presen
No title
EnvironmentTree
Research DVD Technologies and Upload your Findings
View Director Captivate Files 2
Aims & Objectives: Week 2
Preferences
type filter text
CopperCore Server detai
Restore Defaults
Apply
OK
Cancel
Aims and Objectives: Week 2
start

Week 2: Storage Technologies

Media Production and Scripting

Aims and Objectives

This week examines the storage technologies used to store and distribute multimedia applications. In particular this week will examine CD and DVD technologies. You will also be required to do some practical activities relating to Director.

By the end of the week you should have:

- Read through the storage technologies presentation and notes
- Completed the DVD research activity
- ector

CopperCore Server details

CopperCore url: http://localhost:8080
Embedded Tomcat port: 7080

Figure 8.1 Screenshot of CopperCore client using eclipse RCP and SOAP API

In chapter 1 we provided several other examples of how the engine has been integrated and reused in different situations. When the development of CopperCore started, we decided to make the software available through an open-source license. This allowed another kind of reuse: developers can study and modify the source code of CopperCore and CCSI when needed for their own specific purposes. In chapter 1 we also presented some examples of a modified CopperCore source code.

The CCSI architecture introduced adapters that stubbed the original CopperCore APIs with equivalents for both the Java RMI and SOAP protocols. However, the provided methods and passed parameters of these APIs remained the same. Therefore, only minor modifications to client code are required to migrate from the original CopperCore API towards these CCSI stub APIs. One design decision we took when defining the original APIs was to limit the number of available methods as far as possible. We also tried to make the correct sequence of required calls to the API as logical and intuitive as possible by creating chains of method calls where each call requires data obtained from a previous call to be passed as a parameter in the next call. This approach guides API users on invisible rails, as it were. However, it also limits the possibilities to use the engine in different ways than originally perceived. In the 'Further research and development' section, we propose an alternative service design that would also offer a more granular API.

We have now reviewed how CopperCore meets the runtime requirements of validation, parsing, publishing, provisioning, population, personalization and integration by applying the design approach presented in chapter 3 and chapter 4. All three LD levels are supported by the current CopperCore version. Compliancy has been demonstrated in practice by various users' applications of the engine in different projects. CopperCore itself has been downloaded about 10,000 times, which is an indication of its impact. In addition, none of the published UOLs revealed fundamental problems with the engine design or compliancy, though we concluded in chapter 1 that performance tests revealed CopperCore to not yet be suitable for enterprise-scale deployments. This is disappointing, even if this kind of use was not the focus of our research and development; later in this chapter we discuss some proposals that could optimise performance. And by reviewing in chapter 1 the projects that have used CopperCore, we can nevertheless conclude that it has had a considerable impact on the LD community and established itself as the de facto reference runtime implementation for LD. It can thus provide guidance for those issues where the LD specification lacks clarity. Some of those issues we have already encountered, such as the processing of coercions, recursions and trigger-free tautologies.

Our second research and development question addressed the generic integration of learning support services and other e-learning specifications into the CopperCore engine. In chapter 1 we gave an example of an asynchronous conference declaration in LD; the specification also allows other e-learning services to be referenced through declaration in similar fashion. An engine could provision all these services and specifications by simply implementing them as part of the engine core. However, this approach may not be overly sensible given that a number of implementations for these services are already available. Furthermore, LD allows the services to be extended in the future; other specifications, too, could become available. A more agile approach is therefore required to allow the integration of these services and specifications in the context of an LD runtime.

Such integration involves more than just merely provisioning services. The LD runtime must be informed about the service outcomes to be able to adequately respond. chapter 5 presents the CCSI architecture and implementation as a generic solution for the integration of these services within CopperCore.

CopperCore had already established an installed base when CCSI development started. Therefore, the CCSI architecture in chapter 5 adds a new layer of service adapters wedged between a CopperCore client and the service implementation to be integrated. These service adapters replicate the original interfaces of the service implementations. This means that only minimal code modifications need be applied to the existing clients in order to use CCSI. The service adapters enable the underlying service to listen for events from other service adapters, while simultaneously allowing events generated by the underlying service to be sent to a dispatcher. This dispatcher acts as a service bus, and at the same time is the central registration point for the service adapters. We saw that having this service registry allowed the seamless replacement of one adapter implementation by another as long as the service API was honored. Furthermore, new adapters can be added with the needed flexibility as new services become available.

This CCSI architecture was elaborated by taking the example of the IMS QTI and LD integration. We showed how we used lexical similarities between properties, defined in both LD and IMS QTI, to synchronize assessment outcomes with a learning design. This integration was informed by IMS interoperability guidelines. Although not ideal, it provides a practical solution to the integration and interoperability problems between specifications. This was also the approach we took with CCSI in general: instead of having fairly heavy solutions for synchronizing services, it provides a lightweight, flexible and practical solution for service integration.

The concept of property synchronization based on lexical similarity can also be applied to other services. Furthermore, events triggered as a result of administrative tasks such as run provisioning and run population can be used to instantiate some services. In chapter 1 we described some initiatives that extended CCSI with new service types, including new areas such as gaming (Moreno-Ger et al., 2007) and specifications like IMS SCORM (Tattersall et al., 2006). Interesting, too, is the development of a widget adapter (Wilson et al., 2007) which promises to provide much flexibility.

With CCSI we demonstrated that it is possible to integrate e-learning services and other specifications with CopperCore. Our approach is both lightweight and generic while intruding little on existing developments. In chapter 1 we concluded that we successfully addressed both research and development questions with CopperCore and CCSI. However, we also concluded that the uptake of LD remained problematic even after the first range of applications such as CopperCore/CCSI, CCRT, Reload and SLeD became available. In ‘Creating an 28 weeks course with LD and CopperCore’ (Spang Bovey & Dunand, 2006) and ‘Panning for Gold’ (Bailey et al., 2006), the authors argue that the maturity of the LD toolset is a major cause of this disappointing uptake.

The current LD authoring environments in particular are identified as a major obstacle due the amount of expertise they require and the top-down approach they tend to impose.

Therefore, we presented in chapter 7 a complementary approach to LD authoring that lowers the threshold for practitioners in adopting LD. This combines bottom-up authoring with easy-to-use tools with the more formal, top-down approach currently favored by the LD toolset. We used the Personal Competence Manager (PCM) to illustrate this integration. The PCM allows users to develop their personal competences within a competence profile. Each competence may have one or more associated competence developments plans (CDP). These CDPs contain a number of activities that support a user in acquiring the competence. The PCM allows easy-to-use editors for these constructs. Effectively, this allows the creation of very simple learning designs, albeit not LD compliant. Because no distinction is made between design time and run time, authors can constantly modify their learning designs with instantaneous effect.

Although the low threshold of this kind of editing is beneficial to most authors, we identified several arguments for investing in a more formal description of the learning design in LD: accountability, reproducibility, extensibility and quality control. We also explained how the PCM concepts can be mapped onto LD. Each CDP can be expressed and exported as a UOL, and enhanced using the regular LD authoring tools. Our approach is complementary because the LD authoring tools are still required to enhance, modify, or extend the UOL. The authoring cycle is completed by feeding the produced UOL back into the PCM as an alternative to the original CDP. This cycle makes it possible to use the most appropriate authoring tools for the situation at hand. Authors can benefit from the ease of use of the PCM's simple authoring environment in situations where having a formal specification is not valuable. They may also decide to export their learning design to LD when, for example, it has matured into a stable state and requires further refinement not offered by the PCM.

This authoring approach requires that CopperCore and the PCM be closely, seamlessly integrated; we have discussed some of the issues involved in achieving this including rolling-on and rolling-off users, inclusion or exclusion of services, and role assignments. The latter can be especially awkward because the PCM does not distinguish formal roles. How users should be assigned their corresponding roles when importing the UOL is a challenge still to be solved, and might require human intervention. In the 'Further research and development' section we discuss a design that could help solve some of these provisioning and population issues.

We conclude that both research and development questions were successfully addressed by the work presented in this thesis. Both the CopperCore and CCSI developments have generated considerable interest and impact within the IMS LD community, and played important roles in the uptake of LD after its release. However, we must also conclude that the current LD toolset has not yet reached the necessary maturity for large-scale deployments. In 'Learning design: concepts' (Koper & Bennett, in press) the authors provide an overview of these issues. They can be technical – system performance, for example – but mostly concern usability, with LD authoring remaining particularly problematic. We have provided some solutions for these issues in this thesis. In the next section we identify some areas for future research and development.

Further research and development

We have established in this thesis how a reference runtime environment for LD can be designed and implemented. Our design and the resulting implementation were tested by their use in other research and developments and through small-scale experimental deployments. We produced an implementation that uses a collection of FSMs to deal with the challenge of processing the complex LD specification. The event managers were implemented elegantly by reusing the LD condition language to define their behavior. This allowed us to express LD's 'implicit' business rules, as described in the IMS Learning Design Information Model, explicitly as LD conditions. This not only contributed to the overall elegance of our implementation, but also helps to understand and correctly interpret the specification; the lack of expressiveness of XML schema was one of its criticisms (Amorim, Lama, & Sánchez, 2006a; Amorim, Lama, Sánchez, Riera, & Vila, 2006b).

The TELCERT project also addressed some of these issues. Additional descriptions in the form of natural language such as that provided in the IMS Learning Design Information Model are required to compensate for the limitations of the XML schema language. This also applies to the description of the LD runtime behavior, which is entirely in natural language. The use of natural language can lead to ambiguous interpretations, which in turn can lead to interoperability issues. In the previous sections we have already encountered ambiguities in the correct processing of coercions, recursions and tautologies.

Because CopperCore has established itself as the de facto reference runtime implementation, it can help guide users and developers when the correct or desired interpretation of LD is unclear. However, it would be better to have a more precise and formal representation for those parts of the specification which could lead to ambiguity. In their articles 'A learning design ontology based on the IMS specification' (Amorim et al., 2006b) and 'Semantic modelling of the IMS LD level B specification' (Amorim et al., 2006a), the authors propose the use of first-order logic as a more formal notation. They distinguish two kinds of axioms: design time and runtime axioms. They focus on the former; we will deal with the latter. The event handlers described in chapter 3 responsible for processing LD's business logic could be considered implementations of such runtime axioms.Table 8.1 shows the completion rules for an LD act; how part of the LD specification expressed in natural language is converted into a rule expressed by an LD condition. The first row contains the text of the IMS Learning Design Information Model dealing with the completion of an act. The next row describes the same completion rule but this time expressed by two formal, first-order logic axioms. This first-order logic formalization can be applied to a UOL. The third row depicts shows an XML snippet of a UOL dealing with the completion of an act, with the resulting LD condition represented in the fourth row. Note that all quantifiers have been resolved. The condition will be processed by an event handler as described in chapter 3, ensuring that the FSMs are kept up to date.

Table 8.1 LD axiom in natural language and first-order logic, followed by its implementation as an event handler condition

Information model description	Page 42 (item 0.4.1): “This element states that an act is completed when the referenced role-part(s) is (are) completed. More than one role-part can be selected, meaning that all the referenced role-parts must be completed before the act is completed.”
Formal first-order logic axioms	∀ a, ca, rp | a ∈ Act ∧ ca ∈ Complete-Act ∧ complete-act-ref(ca, a) ∧ rp ∈ Role-Part ∧ when-role-part-completed(rp,ca) → role-part-ref(rp, a) ∀ a, rp | a ∈ Act ∧ rp ∈ Role-Part ∧ role-part-ref(rp,a) ∧ complete(rp) → complete(a)
Example XML snippet of UOL	`<act id="act2">` `<title>ACT2: INTRODUCTION TO PREPARATORY PHASE</title>` `<role-part identifier="RolePart3">` `<role-ref ref="Learner"/>` `<learning-activity-ref ref="Preparation_Intro"/>` `</role-part>` `<role-part identifier="lastrolepartact2">` `<role-ref ref="Support_Staff"/>` `<support-activity-ref ref="Preparation_Intro"/>` `</role-part>` `<complete-act>` `<when-role-part-completed ref="lastrolepartact2"/>` `</complete-act>` `</act>`
Generated condition to be interpreted by event handler	`<if>` `<complete>` `<role-part-ref=="lastrolepartact2"/>` `</complete>` `</if>` `<then>` `<complete>` `<act-ref ref="act2" />` `</complete>` `</then>`

Having this formalization of the LD specification will help developers and authors correctly and unambiguously interpret the specification. But it could also help to generate the conditions processed by the event handlers. The logic for creating these conditions is currently hard coded in the engine. If the translation of the first-order logic axioms could be automated, parts of this logic could be externalized. This is not just a good idea in general, but would also improve the robustness of the code and make it more flexible and adaptable for future changes.Table 8.1, however, shows only a single example; further investigation is needed to determine whether all LD runtime rules can be expressed this way.

CopperCore struggled with performance issues. Most were effectively addressed in the latest release of CopperCore, making the engine suitable for deployments of up to 150 simultaneous users. Nevertheless, our implementation of the property persistence requires further attention. CopperCore implements the proposed design of chapter 3 quite literally, meaning that properties are retrieved whenever needed to evaluate expressions, personalize the activity tree or monitor progress. Although property retrieval is fast, doing so can still slow things down simply because there are so many properties, each referenced and fetched separately. Most time is lost in the overhead when establishing a connection with the EJBs. Closer inspection revealed that fetching one property has more or less the same performance impact as fetching 100 properties simultaneously, because the overhead is the predominant factor. Therefore, the solution for increasing overall performance lies in clustering the property access. The challenge here is to find a balance between the current situation, where each property is retrieved separately, and retrieving all properties for all users simultaneously. Several grouping mechanism can be envisaged for collective retrieval.

Two major arguments withheld us from implementing any of the suggested performance optimizations in the last update. First, the current performance appears acceptable and adequate for the typical use of the CopperCore engine at this moment in time. Second, the suggested adaptations would ultimately clutter up the clean design we described in chapter 3. In other words, we let readability and cleaner implementation prevail over performance optimization.

Our CCSI approach showed how specifications such as IMS QTI and learning services like forums can be integrated with CopperCore. We have seen that CopperCore behaves similarly to any other service: an adapter has to be provided and communication is handled through the dispatcher. The CCSI approach is generic, to the extent that it could be used in absence of an LD service. What makes the LD adapter and therefore CopperCore stand out is the fact that it is also the integrative container for all other specifications. A UOL contains the learning design, the IMS QTI items and all the resources. CopperCore therefore provisions the content for the other services.

However, a different approach for a service-oriented architecture using more fine-grained services is also conceivable. Because our CCSI approach is generic, one could argue that the CCSI architecture could have been used for a complete engine redesign. There are indeed some similarities between the CopperCore and CCSI designs. The CCSI dispatcher and the engine’s event dispatcher have similar purposes, and the adapters and event handlers also show commonalities. The merger of CCSI and CopperCore could have resulted in architecture similar to that presented in figure 8.2.

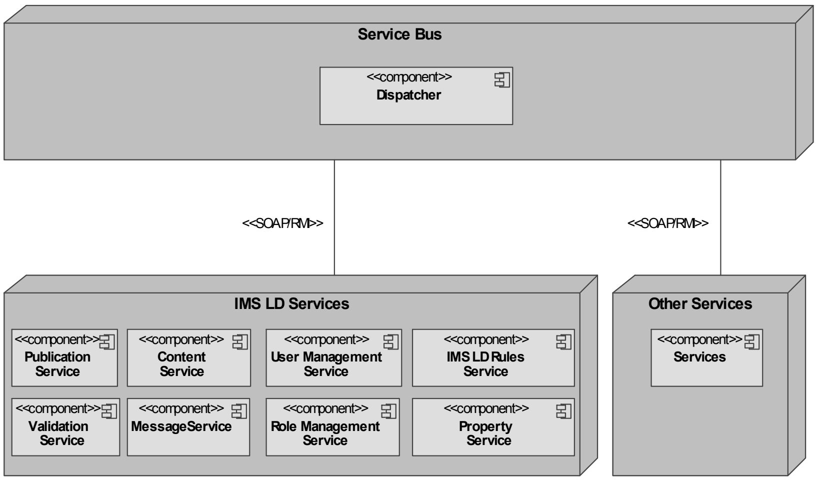

Service Bus
<<component>> Dispatcher
<<SOAP/RMI>>
<<SOAP/RMI>>
IMS LD Services
<<component>> Publication Service
<<component>> Content Service
<<component>> User Management Service
<<component>> IMS LD Rules Service
<<component>> Validation Service
<<component>> MessageService
<<component>> Role Management Service
<<component>> Property Service
Other Services
<<component>> Services

Figure 8.2 Integrated CopperCore CCSI architecture

The architecture of figure 8.2 is probably not fully accurate and some services are likely missing, but it does show how the CopperCore engine could be broken up into more independent, finer grained services. These resemble services as defined by CCSI, and also communicate with each other through a dispatcher. Each service would be accessible through its own API, not shown in the diagram. An high-level public API such as that provided by CopperCore could be added to the service bus. The services themselves would expose a more technically oriented, finer grained API.

We considered using this architecture because it is cleaner and more modular, but there were practical and technical considerations not to do so. We anticipated problems with the number of events being sent via the bus and the impact this would have on performance. These events are so numerous because each property value change triggers an event, and our design relies on having many properties represent a single FSM. The performance of the bus is therefore critical. Because we do not want to make assumptions about how the underlying services are deployed, this architecture must support remote message invocation protocols such as SOAP and RMI to communicate with the services. However, these protocols have considerable performance overhead. In our CCSI approach such performance penalties are limited because only a limited number of events are dispatched. Nevertheless, the proposed architecture of figure 8.2 requires further research into solutions to the expected performance issues. Peter and Vantroys (2005) looked into using a workflow engine to build an LD runtime, and it would be interesting to also explore if and how orchestration and workflow standards such as BPEL4WS and XPDL could play a role in our revised architecture. Another consideration for not revising our architecture was a practical one. CopperCore by that time already had an installed base, and implementing this architecture would likely have resulted in major code and API changes.

In chapter 7 we concluded that there are still unresolved issues regarding rolling-on and rolling-off users and their role assignments later in the process. This provisioning problem is not unique to CopperCore, although its specific implementation is. We would face similar problems if we wanted to enrol users in other e-learning environments. Therefore, a generic solution for this type of problem could be very useful.

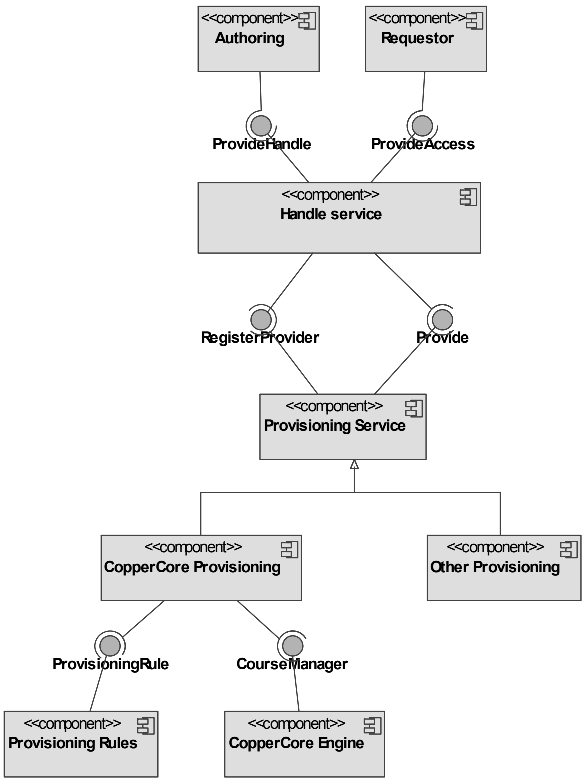

<<component>> Authoring
<<component>> Requestor
ProvideHandle
ProvideAccess
<<component>> Handle service
RegisterProvider
Provide
<<component>> Provisioning Service
<<component>> CopperCore Provisioning
<<component>> Other Provisioning
ProvisioningRule
CourseManager
<<component>> Provisioning Rules
<<component>> CopperCore Engine

Figure 8.3 Proposed architecture for generic provisioning service

Figure 8.3 depicts a draft architecture for generic provisioning with a handle service at its core. This handle service has two purposes. First, it will provide a unique handle for each learning unit. We use the term 'learning unit' deliberately here because it can represent not only a UOL but also any other learning artifact that can be provisioned. The handle represents the learning unit and not an instantiation of it; in the case of LD, then, it represents the UOL and not a run, which could be implemented by using the URI of the UOL. This handle should be used when referring to this learning unit.

The second purpose of the handle service is to provision a learning unit instance for a particular user. We foresee at least three responses to such a provisioning request: *request denied*, meaning that the user will not be granted access to learning unit; *request granted*, which will return a URL that gives access to a particular learning unit instance; and *request postponed*, meaning that the decision whether the user will be granted access to the learning unit is postponed. The requestor must be able to deal with all three responses, and inform the user about the access status. For the handle service to process these requests, at least one provisioning service should have registered with the handle service.

Figure 8.3 shows a CopperCore provisioning service. This particular service would have a database with rules providing information about the roll-on and roll-off behavior for each of the provided UOLs, and even for defining assignments to the UOL roles. This database could be filled by the UOL author in an authoring environment; these rules themselves could be very simple, as described in chapter 7, or very complex, taking complicated planning issues into consideration. It should also be possible to define manual provisioning, leaving the roll-on and roll-off up to human operators like tutors. Once the correct rules have been defined, the actual provisioning happens through the CourseManager interface described in chapter 4.

This architecture provides merely a first step towards a working solution for the provisioning problems of chapter 7. Because it is a generic approach, it would provision access to other learning services in a similar fashion. However, several issues need further investigation, such as authentication issues (single sign-on) and definition of the provisioning rules. But the provisioning service would certainly allow exciting new possibilities for combining environments: for example, a Moodle course could be referenced from a UOL or vice versa. It would also be interesting to investigate whether CCSI could be used as an implementation framework for this architecture, in which case the handler service would be one of the CCSI adapters.

The uptake of LD has been hampered by the maturity of the available toolset. This can largely be explained by the fact that LD covers so many concepts: defining roles and groups, learning content, sequencing of learning content, and personalization of learning. LD’s strength is that it ties all these concepts together into a single package. However, this is also a weakness. Building tools that support the complete specification is complex and requires considerable effort. We also saw in the Moodle example in chapter 1 that integration with other environments is possible but not straightforward, given that concepts likely overlap (Burgos, Tattersall, Dougiamas, Vogten, & Koper, 2006). In addition, successful Web 2.0 (O'Reilly, 2006) developments are currently taking a very different direction. Although the term Web 2.0 can mean radically different things to different people, some characteristics are generally attributed to it, such as the preference for micro formats rather than heavyweight specifications, and the trend to use REST- and JSON-based APIs. Syndication and aggregation of data using RSS, Atom and mash-ups also belong to the current practice attributed to Web 2.0-based applications, giving users control over the environment.

These technological aspects only partly contribute to the current success and hype of Web 2.0. However, they are enabling technologies allowing the easy uptake of new Web 2.0 developments. It would be interesting to investigate whether there are lessons to be learned from the Web 2.0 approach for the work presented in this thesis. Table 8.2 compares some of the aforementioned Web 2.0 characteristics with those currently found in the CopperCore and CCSI toolset.

Table 8.2 Comparison of Web 2.0 and CopperCore/CCSI

	Typical for Web 2.0	CopperCore/CCSI
API interfacing	REST and JSON: lightweight protocols easy to use in browsers	RMI and SOAP: enterprise solutions, relative heavyweight, difficult to use
Formats	Micro formats extending XHTML	XML format partly extending XHTML but also closely related to LD specification
Syndication/ aggregation	RSS and Atom	Syndication through integration of services with CCSI. Content itself cannot be syndicated
Mash-ups	Merging of content from different sources	Mostly merging content from UOL zip file, although reference to external resources is possible
Content	Contributions by individual users, modified continuously	Contributions by authors of learning design and only updated with new releases of UOL
Communities	Ad hoc transient, often self-organized	Predefined by membership of runs
Locus of Control	User	Designer

We must keep in mind the considerable tensions and contradictions between developing a runtime for LD and a typical Web 2.0 application. For example, Web 2.0 focuses on control by the individual user and the user's position in the wider community. LD, in contrast, is directed at orchestration of users where the designer is in control. Communities are often ad hoc and transient in Web 2.0 environments, while they are defined formally through runs and roles in LD. This has led to some of the issues already discussed in chapter 7 regarding run and role assignments. The provisioning architecture depicted in figure 8.3 may also provide a solution for these situations.

Further research is needed to investigate how an LD runtime can be implemented such that it manifests some Web 2.0 features like agility, simplicity and syndication, while still being fully LD compliant. Whether this can be achieved without modification to the specification itself is uncertain. A starting point could be the definition of more granular services such as those presented in figure 8.2.

References

References

Albrecht, A. J., & Gaffney, J. E. (1983). Software Function, Source Lines of Code, and Development Effort Pediction: a Software Science Validation. *IEEE Transactions on Software Engineering, 9*(6), 639-648.

ALFANET (2004). *The ALFANET Project.* Retrieved January 10, 2004, from The Website of the ALFANET Project: http://alfanet.ia.uned.es/

Amorim, R., Lama, M., & Sánchez, E. (2006a). Semantic Modeling of the IMS LD Level B Specification. *The 6th IEEE Conference on Advanced Learning Technologies*, 880-882.

Amorim, R., Lama, M., Sánchez, E., Riera, A., & Vila, X. (2006b). A Learning Design Ontology based on the IMS Specification. *Educational Technology & Society, 9*(1), 38-57.from http://www.ifets.info/journals/9_1/5.pdf

Atkinson, M. P., Bayley, P. J., Chilsom, K. J., Cockshott, W. P., & Morrison, R. (1990). An approach to persistent programming. In S. B. Zdonik & D. Maiers (Eds.). (26 ed., pp. 141-146) (chap. 4).Morgan Kaufmann.

Bailey, C., Zalfan, M. T., & Davis, H. C. (2006, January 1). Panning for Gold: Designing Pedagogically-inspired Learning Nuggets. *Educational Technology & Society, 9*(1), 113-122.

Barr, N. (2000). *Assessment Provision through Interoperable Segments.* Retrieved January 15, 2006, from The website of the APIS project: http://sourceforge.net/projects/apis/

Barrera, C., Boticario, J., Gaudioso, E., Rodriguez, A., Hoke, I., Boy, J., et al. (2005). *D66 - Evaluation Results. ALFANET Project Deliverable*. Retrieved http://rtd.softwareag.es/alfanet/PublicDocs/ALFANET_D66_v1.zip

Barret-Baxendale, M., Hazlewood, P., Oddie, A., & Anderson, M. (2005, December 16). *SliDe final report.* Retrieved January 02, 2008, from website of SliDe: http://www.hope.ac.uk/slide/documents/slide_final.doc

Berlanga, A., & García, F. (2007, August 17). IMS LD reusable elements for adaptive learning designs. *Journal of Interactive Media in Education,*from http://jime.open.ac.uk/2005/11

Berners-Lee, T. (1994, June 1). *Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web.* Retrieved July 03, 2004, from http://www.ietf.org/rfc/rfc1630.txt

Berners-Lee, T., Masinter, L., & McCahill, M. (1994, December 1). *Uniform Resource Locators (URL).* Retrieved from http://www.ietf.org/rfc/rfc1738.txt

Boticario, J., & Santos, O. (2007, September 28). *An open IMS-based user modelling approach for developing adaptive learning management* *systems.* Retrieved January 03, 2008, from Journal of Interactive Media in Education: http://www-jime.open.ac.uk/2007/02/

Brownston, L., Farrel, R., Kant, E., & Martin, N. (1985). *Programming expert systems in OPS5: an introduction to rule-based programming*. Addison-Wesley Longman Publishing Co., Inc.

Burgos, D., Tattersall, C., Dougiamas, M., Vogten, H., & Koper, R. (2006, September 12). Mapping IMS Learning Design and Moodle. A firstunderstanding. *IEEE Technical Committee on Learning Technology*, Proceedings of Simposo Internacional de Informática Educativa (SIIE06) *.* León, Spain.

Burgos, D., Tattersall, C., & Koper, R. (2007, August 2). How to represent adaptation in e-learning with IMS learning design. *Interactive Learning Environments, 15*(2), 161-170.

CETIS (2007). *PLE Report.* Retrieved August 01, 2007, from the website of CETIS: http://wiki.cetis.ac.uk/Ple

D4LD (2006, December 31). *Design for Learning Design.* Retrieved 2007, from website of JISC: http://www.jisc.ac.uk/whatwedo/programmes/elearning_pedagogy/elp_designlearn.aspx

Deursen, A. v., Klint, P., & Visser, J. (2000, June). Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices*, 35*, 26-36, 6 . New York, NY, USA: ACM.

DLD (2005). *Demonstrating learning design.* Retrieved January 17, 2008, from website of JISC: http://www.jisc.ac.uk/whatwedo/programmes/elearning_framework/elfdemo_dld.aspx

dotLRN (2008, January 3). *dotLRN.* Retrieved January 18, 2008, from website of dotLRN: http://dotlrn.org/

Eclipse (2007, May 1). *Eclipse.* Retrieved May 07, 2007, from Website of Eclipse Consortium: http://www.eclipse.org

EFSCE (2007). *E-Framework Services for Course Evaluation.* Retrieved January 02, 2008, from website of EFSCE: http://www.efsce.ecs.soton.ac.uk/index.htm

EJB 3.0 software expert group (2008). *Java Persistence API.* Retrieved November 25, 2007, from website of Sun: http://java.sun.com/javaee/technologies/persistence.jsp

ELeGI. (2007, June 20). *Publishable Final Activity Report*. Retrieved http://213.27.211.106/elegi/wp-content/uploads/2007/06/elegi-publishable-final-activity-report-10.pdf

Elrad, T., Filman, R. E., & Bader, A. (2001). Aspect-oriented programming: Introduction. *Communications of the ACM, 44*(10), 29-32.

EML 1.0 (2000, December 20). *EML 1.0 specification.* Retrieved May 21, 2008, from Dspace site of the Open University of the Netherlands: http://hdl.handle.net/1820/81

EML 1.1 (2002, July 18). *EML 1.1 specification.* Retrieved May 21, 2008, from Dspace site of the Open University of the Netherlands: http://hdl.handle.net/1820/80

Escobedo de Cid, J. P., Fuente Valentín, L. d. l., & Guitérrez, S. (2007, September 28). *Implementation of a Learning Design Run-Time Environment for the .LRN Learning Management System.* Retrieved November 29, 2007, from Journal of Interactive Media in Education: http://www-jime.open.ac.uk/2007/07/

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Retrieved June 12, 2007, from Website of Roy Tomas Fielding at UC Irvine: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

FOAF project (2007). *FOAF.* Retrieved from Website of the Friend of a Friend project: http://www.foaf-project.org/

Fuentes, C., Carrión, J., Arana, C., Boticario, J., Barrera, C., Santos, O., et al. (2005, April 30). *D82 - Public Final Report*. Retrieved http://rtd.softwareag.es/alfanet/PublicDocs/ALFANET_D82_Public.zip

Gaeta, A., Gaeta, M., & Ritrovato, P. (2007, September 29). Gaeta, A.; Gaeta, M.; Ritrovato, P.. *Personal and Ubiquitous Computing,*

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Griffiths, D., Beauvoir, P., Barret-Baxendale, M., Hazlewood, P., & Oddie, A. (2007, November 19). *Development and evaluation of the Reload Learning Design Editor.* Retrieved January 21, 2008, from Paper presented at TENCompetence Open Workshop on Current research on IMS Learning Design and Lifelong Competence Development Infrastructures: http://hdl.handle.net/1820/1135

Griffiths, D., & Blat, J. (2005). The Role of Teachers in Editing and Authoring Units of Learning Using IMS Learning Design. *International Journal on Advanced Technology for Learning, 2*(4), 243-251.

Griffiths, D., Blat, J., Elferink, R., & Zondergeld, S. (2005a). Open Source and IMS Learning Design: Building the Infrastructure for eLearning. *Proceedings of the First International Conference on Open Source Systems (OSS 2005)*, *Proceedings of the First International Conference on Open Source Systems (OSS 2005).* Genova, Italy.

Griffiths, D., Blat, J., García, R., Vogten, H., & Kwong, K.-L. (2005b). Learning Design Tools. In R. Koper & C. Tattersall (Eds.)*. Learning Design, a Handbook on Modelling and Delivering Networked Education and Training* (pp. 109-135) (chap. 7). Heidelberg: Springer.

Hadeli, P., Zamifirescu, C., van Brussel, S.-G. B., Holvoet, T., & Steegmans, E. (2003). *Self-Organising in Multi-Agent Coordination and Control Using Stigmergy.* Retrieved July 19, 2007, from Paper presented at The First Workshop on Self-Organising Engineering Applications (ESOA 2003).Melbourne Australia: http://esoa.unige.ch/esoa03/papers/esoa03_7c.pdf

Harrer, A., Malzahn, N., Hoeksema, K., & Hoppe, U. (2005, August 25). *Learning Design Engines as Remote Control to Learning Support Environments.* Retrieved November 29, 2007, from Journal of Interactive Media in Education: http://jime.open.ac.uk/2005/05/

Hermans, H., Manderveld, J., & Vogten, H. (2004). Educational Modelling Language. In W. Jochems, J. van Merriënboer, & R. Koper (Eds.), *Open and Flexible Learning. integrated E-LEARNING implications for pedagogy, technology & organization* (pp. 80-99) (chap. 6). London, New York: RoutledgeFalmer.

Hernández-Leo, D., Villasclaras-Fernández, E., Asensio-Pérez, J., Dimitriadis, Y., Jorrín-Abellán, I., Ruiz-Requies, I., et al. (2006a). COLLAGE: A collaborative Learning Design editor based on patterns. *Educational Technology & Society, 9*(1), 58-71.

Hernández-Leo, D., Villasclaras-Fernández, E. D., Asensio-Pérez, J. I., Dimitriadis, Y. B.-L. M. L., & Marcos-García, J. A. (2006b). Tuning IMS LD for implementing a collaborative lifelong learning scenario. IEEE International Conference on Advanced Learning Technologies*, Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies*, 1160-1161. Kerkrade, the Netherlands: IEEE.

Hibernate (2008). *Hibernate.* Retrieved November 25, 2007, from Hibernate website: http://www.hibernate.org/

Hutchinson, S. (2007, April 17). *Performance Test Results Report for the Sled player.* Retrieved from website of SLeD: http://sled.open.ac.uk/sledweb/perf/Sled%20Performance%20Testing%20Results.pdf

IBM, BEA Systems, Microsoft, SAP AG, & Siebel Systems (2006). *Business Process Execution Language for Web Services.* Retrieved February 16, 2006, from Website of IBM: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

IEEE (2003). *IEEE Learning Technology Standards Committee.* Retrieved from Website of Learning Technology Standards Committee: http://ltsc.ieee.org

IMS (2003). *IMS Global Learning Consortium.* Retrieved November 11, 2003, from Website of IMS Global Learning Consortium: http://www.imsglobal.org

IMS RDCEO (2002, October 1). *IMS Reusable Definition of Competency or Educational Objective.* Retrieved November 28, 2007, from Website of IMS Global Learning Consortium: http://www.imsglobal.org/competencies/index.html

IMS Simple Sequencing (2006). *IMS Simple Sequencing.* Retrieved from Website of IMS Global Learning Consortium: http://www.imsglobal.org/simplesequencing/index.html

IMS-TIG. (2006). *IMS Tools Interoperability Guidelines*. Retrieved January 12, 2006, from http://www.imsglobal.org/ti/index.html

IMSCP-IM (2003). *IMS Content Packaging Information Model.* Retrieved 2004, from Website of IMS Global Learning Consortium: http://www.imsglobal.org/content/packaging/cpv1p1p2/imscp_infov1p1p2.html

IMSLD (2003). *IMS Learning Design Specification.* Retrieved July 03, 2003, from Website of IMS Global Learning Consortium: http://www.imsglobal.org/learningdesign/index.cfm

IMSLD-BPG. (2003, January 20). *IMS Learning Design Best Practice Guide*. Retrieved June 10, 2003, from http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html

IMSLD-IM (2003, January 20). *IMS Learning Design Information Model. Version 1.0 Final Specification.* Retrieved June 10, 2003, from Website of IMS Global Learning Consortium: http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html

IMSLD-XB. (2003, January 20). *IMS Learning Design XML Binding*. Retrieved June 10, 2003, from http://www.imsglobal.org/learningdesign/ldv1p0/imsld_bindv1p0.html

IMSQTI (2006). *IMS Question and Test Interoperability.* Retrieved January 12, 2006, from Website of IMS Global Learning Consortium: http://www.imsglobal.org/question/index.html

IMSQTI-IG. (2006). *IMS Question and Test Interoperability Integration Guide*. Retrieved http://www.imsglobal.org/question/qti_v2p0/imsqti_intgv2p0.html

J2EE (2007, July 19). *Java Platform, Enterprise Edition.* Retrieved July 19, 2007, from http://java.sun.com/javaee/:

JBoss (2004, July 19). *JBoss Application Server.* Retrieved from http://www.jboss.org:

JISC (2006). *JISC E-Learning Framework: Technical Framework and Tools Strand.* Retrieved February 20, 2006, from Website of the JISC E-Learning Framework, technical framework and tools strand: http://www.jisc.ac.uk/index.cfm?name=elearning_framework

Koper, R. (2001, November 1). *Modeling units of study from a pedagogical perspective: the pedagogical meta-model behind EML.* Retrieved January 14, 2008, from OTEC working paper: http://hdl.handle.net/1820/36

Koper, E. J. R. (2005a). An Introduction to Learning Design. In E. J. R. Koper & C. Tattersall (Eds.)*. Learning Design: A Handbook on Modelling and Delivering Networked Education and Training* (pp. 3-20) (chap. 1).Springer Verlag.

Koper, R. (2005b). Designing Learning Network for Lifelong Learners. In R. Koper & C. Tattersall (Eds.). *A Handbook on Modelling and Delivering Networked Education and Training* (pp. 239-252) (chap. 14).Springer.

Koper, R. (2006). *TENCompetence Domain Model*. Retrieved May 12, 2007, from http://hdl.handle.net/1820/649

Koper, R., & Bennett, S. (in press). Learning Design: Concepts. In P. Kinshuk, D. Sampson, H. H. Adelsberger, & J. M. Pawslowski (Eds.), *International Handbooks on Information Systems. Handbook on Information Technologies for Education and Training*.from http://hdl.handle.net/1820/831

Koper, E. J. R., & Van Es, R. (2004). Modeling units of learning from a pedagogical perspective. In R. McGreal (Ed.). *Accessible eduction using learning objects*. London: RoutledgeFalmer.

Koper, R., Hermans, H., Vogten, H., & Brouns, F. (2007, October 2). *Educational Modelling Language.* Retrieved January 05, 2008, from http://eml.ou.nl: http://eml.ou.nl

Koper, R., & Manderveld, J. (2004, September 1). Educational Modelling Language: Modelling reusable, interoperable, rich and personalised units of learning. *British Journal of Educational Technology, 35*(5), 537-552.

Koper, R., & Olivier, B. (2004). Representing the Learning Design of Units of learning. *Educational Technology and Society, 7*(3), 97-111.

Koper, R., & Specht, M. (2007). TenCompetence: Lifelong Competence Development and Learning. In M. Sicilia (Ed.). *Competencies in Organizational E-Learning: Concepts and Tools* (pp. 230-247) (chap. 11). Idea Group Inc.

Koper, R., & Tattersall, C. (2005). *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*. Berlin Heidelberg New York: Springer Verlag.

LAMS (2008). *LAMS.* Retrieved January 16, 2008, from website of LAMS International: http://www.lamsinternational.com/

Learning Networks (2008, May 23). *Learning Networks Repository.* Retrieved May 23, 2008, from Learning Networks repository: http://dspace.learningnetworks.org/handle/1820/16

M Squared Technologies (2008, May 21). *Effective Lines of Code eLOC Metrics for popular Open Source Software.* Retrieved May 21, 2008, from website of M Squared Technologies: http://msquaredtechnologies.com/m2rsm/rsm_software_project_metrics.htm

Martens, H., Vogten, H., Van Rosmalen, P., & Koper, E. J. R. (2004). *CopperCore.* Retrieved January 14, 2005, from SourceForge: http://coppercore.org

McAndrew, P., Nadolski, R., & Little, A. (2005). Developing an approach for Learning Design Players. *Journal of Interactive Media in Education,*

McAndrew, P., Woods, W., Little, L., Weller, M., Koper, E. J. R., & Vogten, H. (2004, June 3). *Implementing Learning Desing to support web-based learning.* Retrieved January 03, 2008, from AusWeb04.The Tenth Australian World Wide Web Conference: http://ausweb.scu.edu.au/aw04/papers/refereed/mcandrew/

Milligan, C. D., Beauvoir, P., & Sharples, P. (2005, July 1). *The Reload Learning Design Tools.* Retrieved November 29, 2007, from Journal of Interactive Media in Education: http://jime.open.ac.uk/2005/06/

Moodle (2006). *Moodle.* Retrieved from Moodle website: http://moodle.org/

Moreno-Ger, P., Martínez-Ortiz, I., Luis Sierra, J., & Fernández/Manjón, B. (2007, September 28). *Adaptive Units of Learning and Educational Videogames.* Retrieved November 29, 2007, from Journal of Interactive Media in Education: http://jime.open.ac.uk/2007/05/

Nadolski, R., ONeill, W., Vegt, W. v. d., & Koper, R. (2006, February 13). *Conformance Testing, the Elixir within the Chain for Learning Scenarios and Objects.* Retrieved December 03, 2007, from Dspace site of the Open University of the Netherlands: http://hdl.handle.net/1820/581

Navarro, L., Diaz, A., Such, M., Martín, D., & Peco, P. (2007, September 28). *Learning Units Design based in Grid Computing.* Retrieved November 29, 2007, from Journal of Interactive Media in Education: http://jime.open.ac.uk/2007/10/

O'Reilly, T. (2006, September 30). *What Is Web 2.0.*, O'Reilly. Retrieved October 05, 2007, from O'Reilly web site: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

Olivier, B. (2004). *Learning Design Update*. Retrieved August 4, 2006, from http://www.jisc.ac.uk/uploaded_documents/Learning_Design_State_of_Play.pdf

Olivier, B., & Tattersall, C. (2005). The Learning Design Specification. In R. Koper & C. Tattersall (Eds.). *Learning Design. A Handbook on Modelling and Delivering Networked Education and Training* (pp. 21-40) (chap. 2).

OMG (2003). *Unified Modeling Language (UML).* Retrieved January 21, 2006, from website of Object Management Group: http://www.omg.org

OpenACS (2008, January 4). *OpenACS.* Retrieved January 17, 2008, from website of OpenACS: http://openacs.org/

Paquette, G., De la Teja, I., Léonard, M., Lundgren-Cayrol, K., & Marino, O. (2005). An Instructional Engineering Method and Tool for the Design of Units of Learning. In R. Koper & C. Tattersall (Eds.). *Learning Design. A Handbook on Modelling and Delivering Networked Education and Training* (pp. 161-184) (chap. 9).Springer.

Peter, Y., & Vantroys, T. (2005). Platform Support for Pedagogical Scenarios. *Educational Technology & Society, 8*(3), 122-137.

Reload (2007). *Reload Learning Design Editor.* Retrieved June 12, 2006, from the website of the Reload Project: http://www.reload.ac.uk/

Rosenberg, J. (1997). Some Misconceptions About Lines of Code. *Fourth International Software Metrics Symposium (METRICS'97)*, 137-142.

Santos, O., Boticario, J., & Barrera, C. (2008, May 1). *ALFANET: An Adaptive and Standard-Based Learning Environment Built Ipon DOTLRN and Other Open Source Developments.* Retrieved May 01, 2008, from website of Jesús G.Boticario: http://www.ia.uned.es/~jgb/publica/dotrln-ocsjgbcb-final.pdf

SBLDS (2004, April 10). *SBLDS: Service Based Learning Design System.* Retrieved January 17, 2008, from website of JISC: http://www.jisc.ac.uk/whatwedo/programmes/elearning_framework/sblds.aspx

SCORM (2008, May 21). *SCORM 2004 3rd Edition.* Retrieved May 21, 2008, from website of ADL: http://www.adlnet.gov/scorm/index.aspx

Sherrat, R., & Jeyes, S. (2006). *Assis, Final Report.* Retrieved February 21, 2006, from Website of the university of Hull: http://www.hull.ac.uk/esig/downloads/Final-Report-Assis.pdf#search=%22assis%20final%20report%20jisc%22

Sipser, M. (1997). *Introduction to the Theory of Computation*. PWS Publishing Company.

SLeD (2005, February 1). *Service Based Learning Design Player.* Retrieved January 10, 2004, from http://sled.open.ac.uk:

SLeD2 (2005). *SLeD2.* Retrieved December 01, 2007, from website of JISC: http://www.jisccollections.co.uk/sitecore/content/Home/whatwedo/programmes/elearning_framework/elftoolkit_ou.aspx

SLeDID (2005). *SLeD integration demonstrator.* Retrieved December 14, 2007, from website of JISC: http://www.jisc.ac.uk/whatwedo/programmes/elearning_framework/elfdemo_livhope.aspx

Sodhi, T., Miao, Y., Brouns, F., & Koper, R. (2007, June 25). *Design Support for non-expert authors in the creation of units of learning - a first exploration.* Retrieved January 21, 2008, from Dspace site of the Open University of the Netherlands: http://hdl.handle.net/1820/984

SourceForge (2008). *CopperCore.* Retrieved May 21, 2008, from SourceForge: http://sf.net/projects/coppercore

Spang Bovey, N., & Dunand, N. (2006, September 25). Seamless production of interoperable e-Learning units: stakes and pitfalls. *Dspace site of the Open University of the Netherlands, TENCompetence Conference. March 30th-31st, Sofia, Bulgaria* TENCompetence.

Tattersall, C., Sodhi, T., Burgos, D., & Koper, R. (2007, December). Using the IMS Learning Design Notation for the Modeling and Delivery of Education. In L. Botturi & T. Stubbs (Eds.). *Handbook of Visual Languages for Instructional Design: Theories and Practices* (pp. 299-314) (chap. XV).Idea Group Inc.

Tattersall, C., Vogten, H., Brouns, F., Koper, R., Rosmalen, P. v., Sloep, P., et al. (2005a). How to create flexible runtime delivery of distance learning courses. *Educational Technology & Society, 8*(3), 226-236.

Tattersall, C., Vogten, H., & Hermans, H. (2005b). The Edubox Learning Design Player. In R. Koper & C. Tattersall (Eds.). *Learning Design, a Handbook on Modelling and Delivering Networked Education and Training* (pp. 303-310) (chap. 19). Heidelberg: Springer.

Tattersall, C., Vogten, H., & Koper, R. (2005c). An Architecture for the Delivery of E-learning Courses. In R. Koper & C. Tattersall (Eds.). *Learning Design. A Handbook on Modelling and Delivering Networked Education and Training* (pp. 63-73) (chap. 4).Springer.

Tattersall, C., Vogten, H., Martens, H., & Koper, E. J. R. (2006, January 19). How to use IMS Learning Design and SCORM 2004 together. International Conference on SCORM 2006 Tamkang University, Taipei, Taiwan.

TENCompetence (2006). *TENCompetence.* Retrieved from The website of TENCompetence: http://www.tencompetence.org

TENCompetence consortium (2007). *TENCompetence Project.* Retrieved February 01, 2006, from Website of the TENCompetence project: http://www.tencompetence.org

Towle, B., & Halm, M. (2005). Designing Adaptive Learning Environments with Learning Design. In E. J. R. Koper & C. Tattersall (Eds.). *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training* (pp. 215-226) (chap. 12).Springer Verlag.

UNFOLD (2007, December 31). *UNFOLD.* Retrieved from UNFOLD web site: http://www.unfold-project.net:8085/UNFOLD

Van der Vegt, W. (2006, February 17). *CopperAuthor.* Retrieved January 03, 2008, from SourceForge: http://sourceforge.net/projects/copperauthor/

Van Rosmalen, P., & Boticario, J. (2005). Using Learning Design to Support Design and Runtime Adoptation. In R. Koper & C. Tattersall (Eds.). *Learning Design. A Handbook on Modelling and Delivering Networked Education and Training* (pp. 291-301) (chap. 18).Springer.

Van Rosmalen, P., Vogten, H., van Es, R., Passier, H., Poelmans, P., & Koper, R. (2007). Authoring a full life cycle model in standards-based, adaptive e-learning. *Educational Technology & Society, 9*(1), 72-83.from http://www.ifets.info/journals/9_1/7.pdf

Van Rosmalen, P., Brouns, F., Tattersall, C., Vogten, H., Bruggen, J. v., Sloep, P., et al. (2004). Towards an open framework for adaptive, agent-supported e-learning. *International Journal of Continuing Engineering Education and Lifelong Learning, 15*(3-6), 261-271.

Vogten, H., & Martens, H. (2006). *CopperCore Service Integration.* Retrieved February 02, 2006, from Website of the CopperCore Service Integration framework: http://sf.net/projects/ccsi

Vogten, H., Martens, H., Nadolski, R., Tattersall, C., Rosmalen, P. v., & Koper, E. J. R. (2007). CopperCore Service Integration. *Journal on Interactive Learning Environments, 15*(2*),* 171-180.

W3C (1999, November 16). *XSL Tranformations.* Retrieved January 13, 2008, from Website of W3C: http://www.w3.org/TR/xslt

W3C (2000, May 8). *SOAP.* Retrieved from Website of W3C: http://www.w3.org/TR/soap/

W3C (2003). *XML Extensible Markup Language.* Retrieved November 05, 2003, from Website of W3C: http://www.w3.org/XML/

W3C (2007). *XML Schema.* Retrieved from Website of W3C: http://www.w3.org/XML/Schema

Westera, W., Brouns, F., Pannekeet, K., Janssen, J., & Manderveld, J. (2005). Achieving E-Learning with IMS Learning Design - Workflow Implications at the Open University of the Netherlands. *Educational Technology & Society, 8*(3), 216-225.

Wilson, S., Blinco, K., & Rehak, D. (2004, January 7). *Service-Oriented Frameworks: Modelling the infrastructure for the next generation of e-Learning Systems.* Retrieved April 14, 2005, from WebSite of the Joint Information Systems Committee: http://www.jisc.ac.uk/uploaded_documents/AltilabServiceOrientedFrameworks.pdf

Wilson, S., Sharples, P., & Griffiths, D. (2007, July 8). *Extending IMS Learning Design services using Widgets: Initial findings and proposed architecture.* Retrieved January 05, 2008, from Dspace site of the Open University of the Netherlands: http://dspace.ou.nl/handle/1820/963

Workflow Management Coalition (2005, October 3). *Workflow Management Coalition Workflow Standard - Process Definition Interface XML Process Definition Language.* Retrieved November 10, 2007, from website of WFMC:

Zarraonandia, T., Dodero, J. M., & Fernández, C. (2006). Crosscutting runtime adaptations of LD execution. *Educational Technology & Society, 9*(1), 123-137.

zur Muehlen, M., Nickerson, J. V., & Swenson, K. D. (2004). Developing Web Service Choreography Standards; The case of REST vs. SOAP. *Decision Support Systems, 37*

Summary

“*We shall not cease from our exploration, and at the end of all our exploring, we shall arrive where we started and know the place for the first time.*”

T.S. Elliot

Summary

IMS Learning Design (LD) is a formal language for describing learning designs which uses eXtensible Mark-up Language (XML) as its meta-language. A meta-language is used to make statements about another language: for example, English grammar is a meta-language for English. When the LD specification was officially released, there was a need for an associated reference runtime implementation to help practitioners better understand the specification. System integrators would be able to experiment with the integration of LD in their systems, and system developers could benefit from using it as a reference for their own developments.

However, designing and implementing a reference LD runtime environment is not straightforward. The specification combines characteristics from different languages: for example, it shares some characteristics of an imperative programming language in which statements are given in the order they are to be executed in. But it also has some characteristics of a declarative programming language. LD's conditions require constant evaluation, and resemble production rules in a production system. LD is also declarative in a different sense: it expects much scaffolding from the runtime, as is the case, for example, with services referenced by a mere declaration. In addition, LD is a persistence language, implying that the runtime is expected to automatically take care of persistence. Finally, it resembles a workflow language, orchestrating the learning processes between the different roles of different users. Therefore, implementing a runtime environment requires considerable resources and effort. Any reference implementation, therefore, should be reusable in many different situations to make costly rebuilds less necessary.

The first research question of this thesis is thus formulated as follows:

i) *How can a fully compliant reusable reference runtime environment for the IMS Learning Design specification be designed and implemented?*

LD also relies on other specifications and learning services. Although it comes with a fairly detailed description on how to incorporate these specifications and services in the learning design at a lexical level, very little is stated about the runtime implications. This situation has led to the second research question:

ii) *How, given a reference implementation for the IMS Learning Design specification, can implementations for other e-learning specifications and learning support services be integrated generically at runtime level?*

In chapter 2 we review LD in more detail by discussing its main constructs: we see that it comes in three flavours, each extending the other. LD level A defines

the core entities for the specification. It provides constructs for specifying objectives, prerequisites, roles and activities. The method construct combines these roles and activities into role-parts, which can be organized into acts, which in turn can be sequenced through plays. The method is very much inspired by the theatre. There, a play also has roles and acts and actors who each know the role they are supposed to play. Actors know when to appear on stage, for example, and when to interact with each other. Similarly, the LD play determines which activities have to be performed by what roles, and who will interact with whom at what moment.

Level B offers the means to personalize a learning design by adding properties and conditions to the specification. LD properties are similar to the variables found in most programming languages. They are not identical, however: they also have value constraints and a scope, and automatic persistence properties can be manipulated directly by user input, or alternatively via the consequences of conditions. LD conditions will be familiar for anyone with a programming background. However, these conditions are not imperative, as in most programming languages; the runtime environment must determine when the conditions should be evaluated. Through these conditions many different LD structures can be shown or hidden, which is the one of the principles behind personalization in LD.

Finally, Level C contributes only a notification mechanism to the specification. Notifications inform users about events occurring during runtime.

A learning design can be bundled together with its resources to form a unit of learning (UOL). A UOL can be compressed into a single file suitable for further processing. From LD, we derive a set of requirements that any LD runtime must meet in one way or another: validation, publishing, provisioning, population, personalization and integration. We define an LD engine as a software component capable of processing the LD specification. An engine is indifferent to the user interface used to present the engine’s results to the user. The software that renders the engine’s output is called the LD player; one engine can have many players. An engine is designed as part of an enclosing framework, such as a learning management system.

The aforementioned requirements form the starting point for the design of an LD engine that we discuss in chapter 3. This design is from the perspective of a finite state machine (FSM). We use LD properties to capture the state of an FSM, and extend these properties by the concept of implicit properties which, unlike their explicit counterparts, are not defined by the UOL authors. Rather, they are generated by the engine when the UOL is published; they typically capture completion and visibility states. We discuss how a UOL can be populated by real users using the concept of a run. Multiple runs can be created for a single UOL, each with their own user population. The users are assigned to one or more of the roles defined in the corresponding UOL. We explain the relationship between runs, roles and the scope of the properties. Each FSM is represented by all implicit and explicit properties belonging to a user performing a role in a UOL run. We argue that there is no single FSM for a UOL, but rather a collection of them. A single FSM is unambiguously addressed by a run, a user and a role.

Having defined how state is represented by the FSM collection, we focus next on state transitions. These are triggered by events which can be generated through direct user intervention or via external incentives such as the passing of time. They can cause state changes, which in turn can trigger conditions defined in the UOL. This way, a single event can cause a ripple effect not constrained to a single FSM, but able to affect many FSMs throughout the engine. We elaborate on the concepts of a dispatcher and event handlers responsible for this event processing. The event handlers act upon the design in the UOL, processing the consequences of conditions defined in the UOL but also dealing with business rules defined by LD such as the completion of roles-parts, acts and plays. During the UOL's publication, all LD business rules are expressed as conditions using an extended version of the LD condition language.

Because properties can be shared between several FSMs, FSMs can simultaneously change state as a result of altering the value of a single property. This automatically deals with any synchronization issues during the learning flow orchestration. Because the event handlers ensure that each FSM is in the correct state at any given moment in time, personalization becomes a 'fill in the blanks' exercise where references to properties in the UOL are simply replaced by their actual values in the appropriate FSM, regardless of whether these properties are explicit or implicit.

Our engine design was put into practice through an implementation called CopperCore, which has been released as open source using the GPL license. The CopperCore engine is intended to be reused as a service via its APIs. In chapter 4, we take the perspective of a software agent doing just this. We refer to this agent as the client. The CopperCore API is split into a CourseManager API and an LDEngine API. The CourseManager API provides access to the engine's management functionality, including the publication of a UOL, the creation of user accounts, runs, and the assignment of users to roles. These methods are necessary to prepare a UOL for its execution.

The execution itself is achieved through the LDEngine API, which returns personalized XML snippets resembling parts of the original UOL. We discuss the three main calls of the LDEngine API in more detail, and show how the returned XML snippets are based on the original UOL. We also describe in detail how a client can call the CopperCore engine, and elaborate on the expected output. The relationships between the consecutive API calls are further clarified via a sequence diagram representing a typical client scenario.

Finally, we discuss why we implemented the CopperCore engine as a J2EE application. We review some of the possibilities and issues involved in the different deployment strategies for the client and the engine.

Chapter 5 takes the CopperCore engine as starting point. Given this reference implementation for LD, how can other specifications and learning services be integrated in a generic fashion? We present an architecture wedged between the client and the CopperCore engine which allows new services to be added while requiring minimal code changes in any existing clients that may want to use this new architecture. This is important because the CopperCore engine had been released for some time when work on this service architecture started.

We define service adapters, which position themselves between the original service and a client, and replicate the original API of a service. One adapter informs a dispatcher about calls to the connected service that could be relevant for other services, while other service adapters monitor these events and react to them if necessary. The dispatcher functions as a service bus, relaying events between services. The adapters are defined for various service types such as LD, IMS QTI, forums, search, etc. For each type, multiple adapter implementations may exist, each of which must register with the dispatcher and thereby inform the dispatcher that it should be used for the associated service. This allows the flexible configuration of services. We have implemented this service integration architecture and released it as CopperCore Service Integration (CCSI); just like CopperCore, it can be downloaded from SourceForge and is available as open source under a GPL license. CCSI can be installed as addition to CopperCore by simply being deployed on the same application server.

We elaborate on the CCSI architecture by using the integration of LD and IMS QTI as an example. We achieved this integration by synchronizing IMS QTI outcome variables with LD properties on the basis of lexical similarity, an approach which was also recommended by IMS. Finally, we discuss some alternative approaches and argue why we chose in favour of the CCSI implementation.

Chapter 1 reflects on the impact of CopperCore and CCSI on the LD community by reviewing the use of both products in other research and developments. Both developments of CopperCore and CCSI' were iterative processes carried out in the context of several externally funded projects. These projects contributed the necessary resources, but also to the practical validation of the design and implementation of CopperCore and CCSI.

ALFANET was the founding project and resulted in the first release of CopperCore on SourceForge. CopperCore itself was integrated as a separate service in the ALFANET framework. This was followed by a series of SLeD projects carried out with the British Open University which delivered a complete new player, and CopperCore was enhanced to support level C and thereby the full specification. It was also extended with SOAP-compliant APIs, added to open up the engine for non-Java environments. Furthermore, SLeD facilitated the development of CCSI. The SLeD products were installed by Liverpool Hope University to run pilots with its own students as part of a JISC evaluation project. These pilots revealed performance issues with CopperCore which were then successfully addressed in the final SLeD projects.

The UNFOLD project provided a platform for the LD community to meet and exchange ideas and experiences. We briefly describe some of the research and developments presented in the context of the UNFOLD project that reused CopperCore and CCSI. Finally, we take a closer look at the reusing of CopperCore in the TELCERT and ELeGI projects.

Based on the impact of CopperCore and CCSI, we conclude that CopperCore has established itself as the de facto reference runtime environment for LD. We also conclude that many learning design authors have used CopperCore as a reference to help them better understand the specification. At the same time, they also tested the engine in real world practice by deploying and testing designs for all specification levels. The engine has been used many times in various ways, thereby demonstrating its reusability. We also show that a number of new services have been successfully developed for CCSI. We therefore conclude that we have successfully addressed the two research and development questions of this thesis.

CopperCore and CCSI dealt with the biggest obstacles to the uptake of the LD specification. However, the uptake has still been disappointing. This has been ascribed to the toolset's lack of maturity, felt most significantly in the authoring environments. The current LD authoring tools are inadequate for supporting non-expert users, and enforce the top-down model of authoring which seems so natural to LD. In chapter 7, we thus propose a complementary authoring approach that closely integrates CopperCore and CCSI in a Personal Competence Manager (PCM). This approach combines both worlds: on the one hand, the informal approach with easy-to-use editing tools favoring bottom-up authoring; and on the other, the more formal, top-down approach currently favored by the LD toolset.

The PCM allows users to develop their personal competences by selecting competence profiles. Each competence may have one or more associated competence developments plans (CDP). These CDPs contain a number of activities that support acquisition of the competence. The PCM provides easy editing of these constructs, which can be considered a form of creating simple units of learning. These simple units of learning, however, are not LD compliant. Although the lower threshold of this kind of editing is beneficial to most authors, we provide several arguments (e.g. accountability, reproducibility, extensibility, quality control) as to why a formal UOL can be beneficial. The concepts of the PCM, such as competences, competence profiles and competence development plans, can be mapped onto LD, making it possible to export any CDP as a UOL. Such exported UOLs may be enhanced by using the regular LD authoring tool set, like Reload. The common LD authoring tools are still required to modify such a UOL; we therefore consider our approach complementary to these tools.

The authoring cycle is completed by feeding the produced UOL back into the PCM as an alternative to the original CDP. This cycle makes it possible to use the most appropriate authoring tools for the situation at hand. Authors can benefit from the ease of use of the PCM's simple authoring environment in situations where having a formal specification is not valuable or sensible. They may also decide to export their learning design to LD when, for example, it has matured into a stable state and requires further refinement not offered by the PCM.

This authoring approach requires close integration of CopperCore and the PCM. This integration must be seamless; to this end, several issues require further research. These include rolling-on and rolling-off users, inclusion or exclusion of services, and role assignments. The latter can be especially complicated because the PCM does not distinguish any formal roles: ad hoc roles may emerge and be formalized in the resulting LD design.

In chapter 8 we review our results. We reflect on our research and development questions by discussing how we met the set of requirements for an LD engine formulated in chapter 2. We conclude that we have successfully answered both questions, but also identify several topics that require future research and development. We argue that the XML schema formalism is lacking some expressiveness. Therefore, LD cannot solely be described via an XML schema; additional descriptions in natural language are still necessary. This is not ideal, as it could lead to different interpretations of the specification. We propose to use first-order logic to formalize LD's expected runtime behavior, and to use this formalism to automatically generate the CopperCore engine's implicit conditions.

We also touch upon some criticism of our choice of J2EE. We argue that modern persistence frameworks could help simplify the engine, and elaborate on the reported performance problems: although these were addressed, we identified additional measures needed to make CopperCore suitable for use at an enterprise level. We propose performance improvements by introducing a more efficient approach to property fetching, but also discuss some of the drawbacks.

We then reflect on a more harmonized service architecture based on the CopperCore and CCSI concepts. This integrated approach splits up the current engine into separate services, each with a separate API. This architecture is elegant, extensible and flexible, but we also anticipate performance issues with this approach.

We propose a generic approach (i.e. not only limited to LD and CopperCore) to solving the provisioning issues of chapter 7. We present an initial architecture that can transform a UOL's abstract handler into a URL pointing to a fully deployed UOL instance.

Finally, we propose to examine more closely the success of Web 2.0. By identifying some of its typical characteristics and comparing them with the designs and implementations presented in this thesis, we identify some areas of potential future research.

Samenvatting

Samenvatting[1]

IMS Learning Design (LD) is een formele taal voor het vastleggen van *learning designs*. LD maakt hiervoor gebruik van de eXtensible Mark-up Language (XML) als metataal. Een metataal is een taal die wordt gebruikt om een andere taal te beschrijven. Zo is bijvoorbeeld de Nederlandse grammatica een metataal voor de Nederlandse taal. Toen de LD-specificatie werd gepubliceerd ontstond de behoefte aan een referentie-implementatie voor deze specificatie. Zo'n referentie-implementatie maakt de specificatie inzichtelijker en eenvoudiger te begrijpen voor gebruikers, doordat gemaakte ontwerpen 'afgespeeld' kunnen worden. Zo kunnen softwareontwikkelaars experimenteren met de integratie van LD in hun eigen omgevingen. Ontwikkelaars hebben ook baat bij een werkende implementatie als referentie voor hun eigen implementaties van de specificatie. Echter, het ontwerp en de bouw van zo'n referentie-implementatie is geen vanzelfsprekendheid. De specificatie combineert namelijk eigenschappen van verschillende talen. Zo heeft LD kenmerken van een imperatieve programmeertaal waarbij de volgorde van de *statements* bepaald hoe ze later dienen te worden uitgevoerd. Echter, LD is ook declaratief. Dit geldt met name voor de condities van LD, die een voortdurende evaluatie behoeven, waarbij de volgorde niet van tevoren is vastgelegd. In dat opzicht lijken condities op productieregels in een productiesysteem. LD is ook declaratief in een andere betekenis van het woord. LD veronderstelt, dat de implementatie in diverse ondersteunende diensten voorziet, die door middel van een simpele declaratie kunnen worden gespecificeerd. Dit geldt bijvoorbeeld voor de *services* waaraan de LD-specificatie refereert. LD is ook een persistente taal, wat inhoudt dat alle persistentie automatisch door een implementatie afgehandeld dient te worden. Tot slot lijkt LD ook op een *workflow*-taal die de interacties tussen personen, rollen en activiteiten in het *learning design* orkestreert. Zo'n LD-implementatie vergt aanzienlijke middelen en inzet, zelfs indien hiervoor een ontwerp beschikbaar is dat zich al in de praktijk heeft bewezen. Het is derhalve verstandig om zo'n referentie-implementatie herbruikbaar te maken voor diverse situaties en omgevingen.

De eerste onderzoeksvraag van dit proefschrift is daarom als volgt geformuleerd:

i) Hoe kan een herbruikbare geheel compatibele referentie-implementatie voor de IMS Learning Design specificatie worden gebouwd?

LD leunt op andere specificaties en *e-learning* diensten. Hoewel de specificatie een vrij gedetailleerde beschrijving bevat hoe deze andere specificaties dienen te worden geïntegreerd op lexicaal niveau, is er zeer weinig gespecificeerd over de consequenties hiervan tijdens de uitvoering van zo'n ontwerp in *runtime*. Deze situatie heeft geleid tot het tweede onderzoeksvraag van dit proefschrift:

[1] Bij de vertaling hebben we zoveel mogelijk termen vertaald. Waar dit niet goed mogelijk was zijn de originele Engelse termen cursief gedrukt.

ii) Hoe, gegeven de referentie-implementatie voor de LD-specificatie (i), kunnen implementaties voor andere *e-learning* specificaties en diensten op een generieke manier worden geïntegreerd tijdens de *runtime*?

In hoofdstuk 2 zien we dat LD drie varianten kent, waarbij elke variant een uitbreiding is op de vorige. LD-niveau A bevat de belangrijkste constructies van de specificatie en vormt daarmee het meest basale niveau van de specificatie. Het biedt constructies voor het formuleren van leerdoelen, voorwaardelijkheden, rollen en activiteiten. De *method* combineert rollen en activiteiten in *role-parts*. *Role-parts* kunnen worden geordend via *acts* die op hun beurt worden geordend via *plays*. De *method* is geïnspireerd op de metafoor van het theater. Acteurs spelen hier ook een rol en voeren handelingen (*activities*) uit op het toneel. Iedere acteur kent de rol die hij geacht wordt te spelen. Acteurs weten wanneer ze op het toneel moeten verschijnen en hoe ze met andere acteurs samen moeten spelen. De *play,* het draaiboek in LD, bepaalt dus welke activiteiten, wanneer en door wie moeten worden uitgevoerd. LD-niveau B voegt hieraan de mogelijkheid tot personalisatie toe. Niveau B van de specificatie maakt het mogelijk om *properties* te definiëren die essentieel zijn voor de personalisatie. Deze *properties* zijn vergelijkbaar met variabelen van reguliere programmeertalen. De specificatie kent echter speciale eigenschappen toe aan deze variabelen. LD-properties kenmerken zich doordat ze regels kennen die de toegestane waarden bepalen. *Properties* hebben ook nog een instantiebereik en zijn automatisch persistent. De waarden voor deze *properties* worden oftewel rechtstreeks door een gebruiker ingevoerd, of zijn het gevolg van het evalueren van een conditie. Deze LD condities zullen een ieder die enige programmeerervaring heeft, bekend voorkomen. Maar de condities van LD zijn niet imperatief zoals bij de meeste programmeertalen. De implementatie is verantwoordelijk voor de evaluatie, in de juiste volgorde, van de relevante condities. Via deze condities is het mogelijk om verschillende LD-elementen te tonen of te verbergen. Dit verbergen en tonen is de basis van de personalisatie van een ontwerp. Tot slot voegt niveau C van LD slechts een notificatiemechanisme toe aan de specificatie. Dit notificatiemechanisme informeert gebruikers, via berichten, over gebeurtenissen die tijdens de uitvoering van een ontwerp zijn opgetreden.

Een *learning design*, inclusief alle benodigde bronnen, kan worden gebundeld, om zo een leereenheid te vormen (UOL). Een UOL kan vervolgens worden gecomprimeerd tot een enkel bestand dat geschikt is voor verdere verwerking door een *runtime* omgeving. We definiëren een *LD-engine*, of *engine*, als een softwarecomponent die in staat is om de regels van de LD-specificatie te interpreteren en toe te passen op een UOL. Een *engine* heeft zelf geen gebruikersinterface. Een zogenaamde *LD-player*, of *player*, gebruikt de resultaten van de *engine* om deze in een geschikt formaat aan de gebruiker te presenteren. Dezelfde *engine* kan door verschillende *players* worden gebruikt. De *engine* is ontworpen om in een bredere context te worden ingezet. Bijvoorbeeld, als onderdeel van een bestaande e-learning omgeving. Aan de hand van de LD-specificatie beschrijven we een aantal categorieën van eisen waaraan elke LD-implementatie moet voldoen. Deze categorieën van eisen zijn als volgt: validering, publicatie, facilitering, bemensing, personalisatie en integratie.

De bovengenoemde categorieën van eisen zijn uitgangspunt geweest voor een ontwerp van een LD-*engine* zoals we die in hoofdstuk 3 hebben besproken. De idee achter het ontwerp is het concept van een *finite state machine* (FSM). De LD *properties* representeren de toestand van zo'n FSM en we introduceren impliciete *properties* om alle eigenschappen van LD te kunnen vastleggen. Impliciete *properties* zijn, in tegenstelling tot hun tegenhangers, expliciete *properties*, niet gedefinieerd door de auteurs van de UOL. Zij worden gegenereerd door de *engine* op het moment dat de UOL wordt gepubliceerd. Deze impliciete *properties* beschrijven de toestand van bepaalde eigenschappen van objecten, zoals bijvoorbeeld afronding en zichtbaarheid. We lichten toe hoe een UOL kan worden bemenst met gebruikers door het concept van een *run* te introduceren. Meerdere van dergelijk *runs* kunnen worden gecreëerd voor één enkele UOL waarbij iedere *run* bemenst is door zijn eigen groep gebruikers. Deze gebruikers worden vervolgens toegewezen aan een of meerdere rollen zoals die in de UOL zijn gedefinieerd. Vervolgens komt de relatie tussen de *run*, de rol en het bereik van de *properties* aan de orde. Elke FSM is opgebouwd uit de verzameling van alle impliciete en expliciete *properties*. Deze verzameling *properties* wordt geadresseerd door de gebruiker, de *run* en de rol die de gebruiker vervult in die *run*. We concluderen dan ook dat er voor een enkele UOL een hele verzameling van FSM's bestaan. Iedere individuele FSM wordt geïdentificeerd door de *run*, de gebruiker en zijn rol. Nu we hebben beschreven hoe de *engine* kan worden gezien als verzameling van FSM's, richten we ons op de toestandsveranderingen. Toestandsveranderingen zijn het gevolg van gebeurtenissen. Deze gebeurtenissen, kunnen het gevolg zijn van directe interactie van de gebruiker met het systeem, maar ze kunnen ook veroorzaakt worden door externe prikkels, zoals het verstrijken van tijd. Deze toestandsveranderingen kunnen op hun beurt weer leiden tot nieuwe gebeurtenissen enzovoort. Op deze manier kan een rimpeleffect van toestandsveranderingen ontstaan dat zich niet beperkt tot één enkele FSM. Dit rimpeleffect kan zich uitspreiden over vele FSM's. Vervolgens werken we de *dispatcher* en de *event handlers* verder uit die verantwoordelijk zijn voor de verwerking van deze gebeurtenissen. De *event handlers* worden gedefinieerd door het ontwerp, zoals dat is vastgelegd in een UOL. *Event handlers* verwerken de consequenties van de expliciete condities zoals die zijn vastgelegd in een UOL. Echter, ze verwerken ook de consequenties van de impliciete regels die zijn bepaald door de LD-specificatie zelf, zoals bijvoorbeeld de afronding van *rol-parts*, *acts* en *plays*. Tijdens de publicatie van een UOL, zullen alle impliciete LD-regels worden uitgedrukt als expliciete condities. Hiervoor hebben we een licht aangepaste versie van de LD-conditietaal gebruikt.

Omdat sommige *properties* gedeeld worden door meerdere FSM's, kunnen meerdere FSM's ook gelijktijdig van toestand veranderen als gevolg van de verandering van een waarde van slechts één enkele *property*. De synchronisatie van de *learning-flow* tussen verschillende FSM's wordt op deze wijze automatisch geregeld. Omdat iedere FSM te allen tijden in de juiste toestand verkeert, is personalisatie van een UOL vereenvoudigd tot een soort invuloefening waarbij alle verwijzingen naar *properties* in een UOL worden vervangen door de werkelijke waarden van de FSM. Het maakt hierbij niet uit of het hierbij gaat om expliciete of impliciete *properties*.

We hebben ons engineontwerp gerealiseerd via de CopperCore implementatie. CopperCore is uitgebracht onder de GPL *open source* licentie. De CopperCore *engine* is bedoeld om te worden hergebruikt en beschikt daarom over een aantal API’s.

Hoofdstuk 4 benaderen we vanuit het perspectief van een software agent die de CopperCore *engine* gaat gebruiken via deze API’s. We noemen een dergelijke software agent ook wel *client*. De CopperCore API is opgesplitst in een CourseManager API en een LDEngine API. De CourseManager API geeft toegang tot de administratieve functionaliteit van de engine. Dit omvat de publicatie van UOLs, het aanmaken van gebruikersaccounts, *runs* en de toewijzing van gebruikers aan de rollen. Deze methoden zijn erop gericht om een UOL te gereed te maken zodat hij kan worden ‘afgespeeld’. Voor het feitelijke uitvoeren van een UOL wordt de LDEngine API gebruikt. De LDEngine API genereert gepersonaliseerde XML fragmenten die een grote gelijkenis vertonen met onderdelen van de oorspronkelijke UOL. We bespreken de drie voornaamste LDEngine API methoden in detail en laten zien hoe de resulterende XML fragmenten zijn afgeleid van de originele UOL. We beschrijven in detail hoe een *client* de CopperCore *engine* kan aanroepen en welke resultaten dit oplevert. Een sequentiediagram licht de relaties tussen de opeenvolgende API calls toe waarbij we een typisch scenario als uitgangspunt gebruiken.

Tot slot bespreken we de reden waarom wij gekozen hebben om CopperCore als een J2EE-toepassing te implementeren. Hierbij laten we een aantal van de configuratiestrategieën en mogelijkheden voor zowel de *client* als de *engine* de revue passeren.

Hoofdstuk 5 neemt de CopperCore *engine* als uitgangspunt. Centraal staat de vraag hoe educatieve specificaties en diensten op een generieke manier geïntegreerd kunnen worden via de CopperCore referentie-implementatie voor LD. We presenteren een architectuur die als wig tussen de *client* en de CopperCore *engine* is geplaatst. Deze architectuur maakt het mogelijk om nieuwe diensten toe te voegen, terwijl tegelijkertijd slechts minimale wijzigingen nodig zijn in de broncode van reeds bestaande *clients* om gebruik te maken van deze nieuwe architectuur. Dit is belangrijk omdat de CopperCore *engine* reeds enige tijd beschikbaar was toen deze diensten-integratie-architectuur werd ontworpen. Om de inbreuk op de bestaande code te beperken zijn *service adapters* gedefinieerd die zich nestelen tussen de originele dienst en een aanroepende cliënt. Deze *service adapters* repliceren de originele API van de dienst die ze integreren. Een *service adapter* informeert een *dispatcher* over gebeurtenissen die van belang kunnen zijn voor andere diensten. Andere *service adapters* houden deze gebeurtenissen in de gaten en reageren hierop indien dit van belang is. De *dispatcher* werkt dus als een soort *service bus* en is een doorgeefluik voor berichten binnen het systeem. We hebben verschillende *service adapters* gedefinieerd, elke voor verschillende soorten specificaties en diensten, zoals bijvoorbeeld LD, IMS QTI, fora, zoekdiensten enz. Voor elke *service adapter* kunnen in principe meerdere implementaties bestaan. Zo'n implementatie voor een *service adapter* meldt zichzelf aan bij de *dispatcher*. Dit maakt een dynamische configuratie van de diensten mogelijk. We hebben deze diensten integratie architectuur geïmplementeerd en uitgebracht onder de naam CopperCore Service Integration (CCSI). Net als CopperCore is CCSI via SourceForge te downloaden. CCSI is beschikbaar als *open source* door middel van een GPL licentie. CCSI kan worden geïnstalleerd als aanvulling op CopperCore door deze op dezelfde applicatieserver te installeren.

Vervolgens werken we de CCSI-architectuur verder uit door de integratie van LD en IMS QTI als voorbeeld te nemen. Wij hebben deze integratie bereikt door de IMS QTI uitkomstvariabelen en de LD-*properties* met elkaar te synchroniseren op basis van hun lexicale gelijkenis. Deze aanpak wordt ook aanbevolen door IMS. Tot slot bespreken we een aantal alternatieve benaderingen en beargumenteren we waarom we uiteindelijk hebben gekozen voor de CCSI oplossing.

Hoofdstuk 6 begint met de vaststelling, dat beide onderzoeks- en ontwikkelvraagstukken van dit proefschrift in principe zijn beantwoord. We reflecteren op de invloed van zowel CopperCore en CCSI op de LD-gemeenschap door het gebruik van beide producten in andere onderzoeks- en ontwikkelprojecten te bekijken. De ontwikkeling van CopperCore en CCSI was een iteratief proces en deze iteraties hebben plaatsgevonden in de context van extern gefinancierde projecten. Deze projecten hebben niet alleen bijgedragen aan de ontwikkeling van CopperCore- en CCSI-implementaties, maar ze hebben ook bijgedragen aan de validering ervan.

Het ALFANET-project was het startsein voor deze ontwikkelingen en de eerste versie van CopperCore op SourceForge was een resultaat van dit project. CopperCore fungeerde als een aparte dienst in een overkoepelende ALFANET-architectuur. ALFANET werd opgevolgd door een reeks SLeD-projecten die samen met de Britse Open Universiteit zijn uitgevoerd. Als onderdeel hiervan is CopperCore verbeterd zodat ook niveau C van de LD-specificatie wordt ondersteund. Bovendien zijn SOAP compatibele API's aan CopperCore toegevoegd, zodat ook niet-Java-omgevingen gebruik kunnen maken van CopperCore. SLeD stond ook aan de basis van de ontwikkeling van CCSI. Voorts is er een geheel nieuwe *player* in het kader van SLeD ontwikkeld. Een aangepaste versie van de SLeD-omgeving is door Liverpool Hope University gebruikt om praktische ervaringen op te doen met hun studenten als onderdeel van een JISC-evaluatieproject. Deze experimenten brachten problemen met de prestaties van CopperCore aan het licht bij gebruik met grotere aantallen gebruikers. In het laatste SLeD-project zijn deze problemen geadresseerd en grotendeels verholpen.

Het UNFOLD-project verschafte de LD-gemeenschap een platform om elkaar te ontmoeten en van gedachten te wisselen over nieuwe ideeën en ervaringen.

We beschrijven kort enkele van de onderzoeks- en ontwikkelactiviteiten die in het kader van UNFOLD zijn gepresenteerd en gaan daarbij met name in op het hergebruik van CopperCore en CCSI. Tot slot bespreken we het hergebruik van CopperCore in de TELCERT- en EleGI-projecten.

Gebaseerd op de invloed van CopperCore en CCSI op de LD-gemeenschap kunnen we concluderen dat CopperCore is uitgegroeid tot de *de facto* referentieimplementatie voor LD. We concluderen ook dat de LD-gemeenschap CopperCore heeft gebruikt om een beter en dieper inzicht te krijgen in LD-specificatie. Tegelijkertijd heeft diezelfde gemeenschap CopperCore in de praktijk gevalideerd, door het testen van hun ontwerpen met behulp van CopperCore voor alle niveaus van de specificatie. Bovendien hebben we gezien dat CopperCore meermaals is hergebruikt en dat dit op verschillende manieren is gebeurd. We hebben ook vastgesteld dat er met succes diverse nieuwe diensten aan CCSI zijn toegevoegd. Daarmee rechtvaardigen we de conclusie dat we beide onderzoeks- en ontwikkelvragen van dit proefschrift succesvol hebben beantwoord.

CopperCore en CCSI hebben de grootste barrière voor een succesvolle start van LD weggenomen. Echter, we moeten ook vaststellen dat het gebruik van LD nog altijd achter blijft bij de verwachtingen. Het gebrek aan volwassenheid van de huidige generatie LD-software is hiervoor de hoofdoorzaak. Met name de beschikbare auteurs omgevingen worden als problematisch ervaren. Deze omgevingen ondersteunen de niet-expertgebruiker onvoldoende en ze leggen de auteurs een top-down benadering op, die erg eigen is aan LD maar vaak niet wenselijk wordt gevonden. Daarom presenteren we in het volgende hoofdstuk een complementaire aanpak voor deze top-down benadering door gebruik te maken van een hechte integratie van CopperCore en CCSI in de Personal Competence Manager (PCM).

In hoofdstuk 7 beschrijven we een aanpak voor de LD-auteursproblematiek die twee werelden combineert. Aan de ene kant is er de informele aanpak met eenvoudig te gebruiken software die een bottom-up aanpak toestaat. Aan de andere kant is er de meer formele top-down benadering, zoals die momenteel wordt ondersteund door de LD-auteursomgevingen. De PCM ondersteunt gebruikers bij het bereiken van hun competenties via het kiezen van competentieprofielen. Iedere competentie in zo'n profiel kent één of meerdere competentie-ontwikkelplannen (CDP). Deze CDP's bevatten de leeractiviteiten die nodig zijn om een competentie te bereiken. In feite biedt de PCM eenvoudige voorziening die gemakkelijk zijn te gebruiken om leereenheden te maken. Deze leereenheden zijn echter niet LD-compatibel. Deze manier van leereenheden maken is weliswaar erg gemakkelijk voor auteurs, maar er zijn ook omstandigheden waarbij een meer formele beschrijving van zo'n leereenheid door middel van LD te prefereren is. De factoren die bij deze overweging een rol spelen zijn o.a.: aansprakelijkheid, reproduceerbaarheid, uitbreidbaarheid en kwaliteitscontrole. Doordat de concepten van de PCM vertaald kunnen worden naar constructen van LD is het mogelijk om een leereenheid vanuit de PCM te exporteren naar een UOL. Zo'n UOL kan vervolgens met de beschikbare auteursomgevingen voor LD aangepast en/of verbeterd worden. Omdat we de bestaande LD-auteursomgeving nog altijd nodig hebben, beschouwen we onze aanpak dan ook als complementair. De ontwikkelcyclus kan vervolgens worden gesloten door een al dan niet aangepaste UOL weer te importeren via de PCM. Een dergelijke UOL vormt dan een alternatief voor de originele leereenheid. Op deze manier is het mogelijk om de meest geschikte auteursomgeving te gebruiken al naargelang de behoefte. Auteurs kunnen profiteren van het gemak en de eenvoud van de PCM voor situaties waarbij een formele representatie van hun leereenheid niet belangrijk is of geen toegevoegde waarde heeft. Diezelfde auteurs kunnen gebruik maken van de kracht van de formele LD-specificatie door hun initiële ontwerp te exporteren naar een UOL omdat er bijvoorbeeld behoefte is aan verfijningen van het ontwerp waarin de PCM niet kan voorzien.

De geschetste aanpak vergt een nauwe integratie van CopperCore met de PCM. Deze integratie zou naadloos moeten zijn en er zijn nog verschillende problemen die opgelost moeten worden voordat dit een feit is. Deze problemen betreffen het toewijzen van gebruikers aan *runs,* de integratie van diensten en het toewijzen van rollen. Zekere het laatste kan problematisch zijn omdat de PCM geen formeel onderscheid maakt tussen de rollen van gebruikers. In de PCM kunnen ad hoc rollen ontstaan die vervolgens in de resulterende UOL worden geformaliseerd. Hoe we deze toewijzing automatisch kunnen regelen, blijft een vraag en mogelijk zal hiervoor in sommige gevallen menselijk interventie nodig zijn.

In hoofdstuk 8 bekijken we de resultaten. Hierbij gebruiken we de categorieën van eisen, zoals die in hoofdstuk 2 zijn besproken, als leidraad. We komen tot de conclusie dat we de onderzoeks- en ontwikkelvraagstukken van dit proefschrift met succes hebben beantwoord. Echter we identificeren ook enkele aandachtsgebieden die verder onderzoek en ontwikkeling vergen. We stellen dat het XML-schemaformalisme onvoldoende expressief is. De LD-specificatie kan niet in zijn geheel worden beschreven met een XML schema. Daardoor zijn er additionele regels nodig die momenteel in natuurlijke taal zijn beschreven. Dit geldt des te meer voor de beschrijving van het *runtime* gedrag omdat hier helemaal geen formalisme is gebruikt. Dit is niet ideaal, omdat dit tot verschillende interpretaties kan leiden. Daarom stellen we voor om eerste orde logica te gebruiken voor het vastleggen van het gewenste *runtime* gedrag. Dit formalisme kan worden gebruikt om de impliciete condities te genereren die nodig zijn voor de correcte verwerking van LD.

We reageren op kritiek voor onze keuze voor het complexe J2EE. We stellen dat moderne persistentie-oplossingen kunnen helpen met het vereenvoudigen van het ontwerp. Voorts gaan we dieper in op de prestatie-problemen van CopperCore waardoor CopperCore nog niet geschikt is om op institutioneel niveau te worden ingezet. Om deze prestaties te verbeteren stellen we voor om het ophalen van de *properties* efficiënter te maken door de *properties* te bundelen.

Vervolgens bespreken we een meer modulaire, geharmoniseerde diensten-architectuur die de concepten van CopperCore en CCSI combineert. De huidige

CopperCore *engine* wordt hierbij in verschillende kleinere diensten opgesplitst, waarbij ieder dienst zijn eigen API heeft. Deze aanpak is elegant, uitbreidbaar en flexibel, maar we verwachten ook wel problemen met de prestatie van dit systeem.

We stellen ook een generieke aanpak voor om de problemen van hoofdstuk 7 omtrent de beschikbaarheid van *runs* op te lossen. We presenteren een eerste versie van een architectuur die in staat is om abstracte *handles* te vertalen naar URL’s voor de corresponderende run. Onze oplossing is niet beperkt tot LD en CopperCore, maar zou ook in andere omgevingen kunnen worden toegepast.

Tot slot kijken we naar het succes van de huidige Web 2.0 toepassingen. We identificeren enkele eigenschappen van Web 2.0 en vergelijken deze met de ontwerpen zoals ze in het proefschrift zijn gepresenteerd. Op basis van deze vergelijking suggereren we enkele gebieden die verder aandacht verdienen.

Acknowledgement

Acknowledgements

This thesis is a milestone in a journey I embarked on almost 10 years ago. It all started in 1998 with the exciting prospect of developing a new e-learning environment for the Open University of the Netherlands. A new educational language, later to become known as EML, was to play an important role in this new environment, and ultimately resulted in the research and developments described in this thesis. I have met many fellow travellers on this journey, some whose company I enjoyed for only parts of it, others who still accompany me until this day. To all of them I owe my gratitude for their stimulating company and the opportunity to do such arresting work together. I wish to express my appreciation to some in particular, while acknowledging at the same time that I will no doubt forget to mention many others. To them I apologize upfront.

First of all I wish to thank my supervisor and co-supervisor. Rob Koper stimulated me to write this thesis and provided me with the opportunity to do so. I felt honoured, because I am not part of OTEC's academic staff and this opportunity is not something to take for granted. Jan van Bruggen was willing to take on the role of co-supervisor at a late stage in the process, and his invaluable comments helped me set my beacons and stay on course. Colin Tattersall was my initial co-supervisor, and thanks go to him for having faith in me and believing in a good outcome right from the start.

Also special thanks to Harrie Martens, who was one of my fellow travellers from the first hour. He co-developed and co-authored most of the work discussed in this thesis. He was also my sounding board in sometimes heated discussions. We developed a professional respect for each other that could easily withstand occasional disagreements: our cooperation hit the bulls eye. Thanks also to Peter van Rosmalen, who paved the way for me by taking this PhD route first. His help in various roles such as project leader, fellow PhD student and occasionally ad hoc bellboy lightened my work more than once, while his famous one-liners always brightened my day. And to Francis Brouns, who was also a fellow traveller and companion from the start. She often had to deal with the consequences of my taking this journey, both at work and in private. To Mieke Haemers, too, who was a true coach throughout the writing of this thesis. She did a wonderful job helping with all the formalities as well as the final proofreading.

Many thanks also to all the people in the ALFANET project team: Cristina Arana, Carlos Fuentes, Jesús González Boticario, Carmen Barrera, Olga Santos, Jürgen A. Schmidt, Ingeborg Hoke, Elsa Escala, Adalberto Moutinho, Francisco Barros, Roberto Canada, Peter van Rosmalen, Harrie Martens, René van Es, Patricia Poelmans, Frans Mofers, Harrie Passier, Slavi Stoyanov, John van der Baaren, Leo Wagemans. A project which had a rough start, but which I will remember foremost as the foundation for the CopperCore engine.

Many thanks to all those in the various SLeD projects: Patrick McAndrew, Martin Weller, Will Woods, Juliette White, Alex Little, Simon Hutchinson, Mark

Barrett-Baxendale, Harrie Martens and Rob Nadolski. I especially enjoyed the fruitful and uncomplicated cooperation of our various projects.

I would also like to thank Phil Beauvoir, Paul Sharples and Scott Wilson for their many contributions and inspiring discussions in the UNFOLD and TENCompetence projects. Many thanks to Ruud Lemmers, too, for his efforts to keep the TENCompetence developments on track. I learned a lot from our joint development efforts.

Finally, I wish to express my gratitude to everyone, past and current, from the OTEC development programme, without whom this thesis would not have been possible. And last but not least, to Marc Verhooren for his contributions to the Edubox system and LD specification, which in many respects paved the way for the work presented in this thesis.

Curriculum Vitae

Curriculum Vitae

Hubert Vogten, born on the 16th of February 1967 in Sittard the Netherlands, began his career in 1990 at the Open University of the Netherlands (OUNL). Since then, he has been active in educational technology in various roles. He started developing hypertext and hypermedia systems in projects both for the university itself and in industry, mostly using either Smalltalk or C++, until in 1994 he founded his own company which successfully developed a system capable of representing OUNL curricula and the complex associated regulations. This system handled the remapping of some 10,000 student records onto OUNL's redefined curricula.

In 1994 Vogten was hired by the European Association of Distance Teaching Universities for the JANUS project (EU 3rd framework). In the following year, he joined the European Open University Network, where he was responsible for the network's technology development. He participated in the EOUN project (4th framework) and wrote a successful proposal for the WIRE project (EU Ten-Telecom programme). During this period he also worked with new and emerging technologies such as satellite technology through VSATs, point-to-point and multi-point video conferencing, interactive television, email, chat, computer conferencing, and new web technologies to establish a 'European Virtual University'. This meant he was involved in setting up one of the earliest websites in the Netherlands.

In 1998 Vogten was asked to rejoin the Open University to work on a new virtual learning environment. As a result he was closely involved in the development of EML, and helped develop its first series of prototypes. This led to a range of Edubox systems going into production at the OUNL. At the same time, he was also involved in standardizing EML through IMS, work which eventually resulted in the release of the LD specification in 2003.

Vogten has participated in various R&D projects such as ALFANET (5th framework); a series of SLeD projects (JISC); and TENCompetence (6th framework), for which he helped research and develop an open source runtime environment for LD. His current work focuses on researching and developing services for lifelong competence development.

SIKS Dissertatiereeks

SIKS Dissertatiereeks

1998

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup, (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17	Stefan Manegold (UVA) Understanding, Modeling, and Improving Main-Memory Database Performance

2003

2003-01	Heiner Stuckenschmidt (VU) Ontology-Based Information Sharing in Weakly Structured Environments
2003-02	Jan Broersen (VU) Modal Action Logics for Reasoning About Reactive Systems
2003-03	Martijn Schuemie (TUD) Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04	Milan Petkovic (UT) Content-Based Video Retrieval Supported by Database Technology
2003-05	Jos Lehmann (UVA) Causation in Artificial Intelligence and Law - A modelling approach
2003-06	Boris van Schooten (UT) Development and specification of virtual environments
2003-07	Machiel Jansen (UvA) Formal Explorations of Knowledge Intensive Tasks
2003-08	Yongping Ran (UM) Repair Based Scheduling
2003-09	Rens Kortmann (UM) The resolution of visually guided behaviour
2003-10	Andreas Lincke (UvT) Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11	Simon Keizer (UT) Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12	Roeland Ordelman (UT) Dutch speech recognition in multimedia information retrieval
2003-13	Jeroen Donkers (UM) Nosce Hostem - Searching with Opponent Models
2003-14	Stijn Hoppenbrouwers (KUN) Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15	Mathijs de Weerdt (TUD) Plan Merging in Multi-Agent Systems
2003-16	Menzo Windhouwer (CWI) Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17	David Jansen (UT) Extensions of Statecharts with Probability, Time, and Stochastic Timing
2003-18	Levente Kocsis (UM) Learning Search Decisions

2004

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

2005

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching -- balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

2007

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance

2007-06	Gilad Mishne (UVA) Applied Text Analytics for Blogs
2007-07	Natasa Jovanovic' (UT) To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08	Mark Hoogendoorn (VU) Modeling of Change in Multi-Agent Organizations
2007-09	David Mobach (VU) Agent-Based Mediated Service Negotiation
2007-10	Huib Aldewereld (UU) Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
2007-11	Natalia Stash (TUE) Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
2007-12	Marcel van Gerven (RUN) Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
2007-13	Rutger Rienks (UT) Meetings in Smart Environments; Implications of Progressing Technology
2007-14	Niek Bergboer (UM) Context-Based Image Analysis
2007-15	Joyca Lacroix (UM) NIM: a Situated Computational Memory Model
2007-16	Davide Grossi (UU) Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
2007-17	Theodore Charitos (UU) Reasoning with Dynamic Networks in Practice
2007-18	Bart Orriens (UvT) On the development an management of adaptive business collaborations
2007-19	David Levy (UM) Intimate relationships with artificial partners
2007-20	Slinger Jansen (UU) Customer Configuration Updating in a Software Supply Network
2007-21	Karianne Vermaas (UU) Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
2007-22	Zlatko Zlatev (UT) Goal-oriented design of value and process models from patterns
2007-23	Peter Barna (TUE) Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramírez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

2008

2008-01 Katalin Boer-Sorbán (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wouter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains

2008-16 Henriëtte van Vugt (VU)
Embodied agents from a user's perspective

2008-17	Martin Op 't Land (TUD) Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18	Guido de Croon (UM) Adaptive Active Vision
2008-19	Henning Rode (UT) From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
2008-20	Rex Arendsen (UVA) Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.
2008-21	Krisztian Balog (UVA) People Search in the Enterprise
2008-22	Henk Koning (UU) Communication of IT-Architecture
2008-23	Stefan Visscher (UU) Bayesian network models for the management of ventilator-associated pneumonia
2008-24	Zharko Aleksovski (VU) Using background knowledge in ontology matching
2008-25	Geert Jonker (UU) Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
2008-26	Marijn Huijbregts (UT) Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled