Automating educational processes implementation by means of an ontological framework

Angels Rius[1], Jordi Conesa[1], Elena García-Barriocanal[2], Miguel Angel Sicilia[2]

[1] Estudis d'Informatica i Multimedia, Universitat Oberta de Catalunya,
Rambla del Poblenou, 156, E-08018, Barcelona, Spain
mriusg@uoc.edu, jconesac@uoc.edu
[2] Computer Science Department, University of Alcalá
Ctra. Barcelona km. 33,600.28871.Alcalá de Henares (Madrid), Spain
elena.garciab@uah.es, msicilia@uah.es

Abstract. In the area of eBusiness, automation from the process specifications is a reality that has provided relevant advantages and benefits. Even though the similarities between the areas of eBusiness and eLearning, the automation of processes based on their specifications is not yet possible in eLearning. The lack of standards and specifications to specify learning processes unambiguosly may be one possible explanation for that limitation. This paper proposes a method that generates automatically part of the implementation of learning processes from their specifications. In order to do so, the method uses the OKI-OSID specification. This specification proposes a framework for the creation of eLeaning sistems that promotes interoperability. Since the specification does not include a metamodel, an ontology that defines an OKI-OSID metamodel has been created. The ontology has been used to store the specifications of learning processes and to make necessary inferences in order to generate part of the processes' source code. Apart of the OKI-OSID ontology, the paper also presents some insights about some problems found in OKI-OSID and the underspecifications these lacks produces. An implementation of the method has been implemented in order to create the source code of learning processes in Java, but its extension to generate code in other languages is straighforward.

Keywords: interface, educational setting, pattern, specification, ontology

1 Introduction

Today, educational institutions can reuse a great deal of applications in order to create their own applications to improve learning processes. Applications aimed at e-learning environments can be created in two different ways: a) by using, and extending when necessary, the functions of a Learning Management System (LMS) or b) from scratch. In the first option, the particularities of the LMS platform reused must be taken into consideration, so possible extensions will depend mainly on the base LMS and its limitations. The second option requires the development of applications that can be adapted to the needs and management of each educational institution. Even though, extending an LMS can save implementation time and resources, we believe that implementing systems from scratch is preferable, given that it facilitates the adaptation of applications to each organisational environment irrespective of the LMS.

The Object Management Group proposed in the Model Driven Architecture methodology (MDA) a way to create platform-independent software systems from the models that represent their specifications. In MDA the software development is directed by models, meaning that the implementation is mostly done in a higher degree of abstraction: people implement by modeling instead of by programming [12][24]. These models can be seen as the ontologies the information systems need to know in order to perform their functions [5]. In other fields some authors have also created a framework using ontologies for reusing, extracting and extending large domain ontologies by means of a SOA (Service Oriented Architecture) approach [3]. Information systems that use ontologies to be improved are known as Ontology-Driven Information Systems (ODIS) [23]. The main problem in these systems is the necessity to have an ontology that is good enough to be of value to the information system. Creating this ontology is a difficult and very time consuming task that may, sometimes, exceed the time or efforts of developing the application from scratch. However, if we can create an ontology that is general enough to be reused by different information systems of the same domain, then the resources needed for its creation can be acceptable. In the context of eLearning, an ontology that describes the most used specifications or standards, such as the OKI OSID specification, can be used successfully in order to automate some processes, increase interoperability among systems and supporting the programmers in the creation of new applications.

The mid-term objective behind this work is to create an ODIS that supports the creation of learning applications using the MDA methodology. With this objective in mind, the objective of this work is two folded: 1) the creation of an ontology of the OKI-OSID specification that allow represent learning processes specifications, and 2) the creation of an ODIS system that uses the OKI-OSID ontology in order to support the implementation of learning processes.

In order to fulfill these objectives, the paper proposes a method for the specification of processes, irrespective of the learning system platform to be used, that is capable of establishing links between the processes that are carried out in different educational environments and their implementation. To prove the feasibility of the proposed model, a prototype was created that part-generate the implementation of educational scenarios by means of graphic representations in educational scenarios [7] and [6]. This framework is specially aimed at the higher education community and aims to offer support to the construction of learning systems and promote interoperability between different e-learning applications. In order to store and manage the specifications of learning processes the method uses the OKI-OSID ontology. This ontology has been created from scratch and defines the part of the OKI-OSID specification relevant for the proposed method. In addition, an in-depth study was also carried out on the O.K.I. OSID specification that has made it possible to create a metamodel and also to detect a number of its limitations.

This work contributes to the community in two main ways: the proposal of a new method to represent learning processes and their interactions with other eLearning applications, and an OKI metamodel that supports creating eLearning applications using a model driven approach (MDA).

This paper is composed of seven sections. After the introduction, the paper studies the relevant related work. Third section presents an overview of the proposed approach. Thereafter, section four describes the case study that will be used in order to exemplify the proposed approach. Fifth section describes the OKI metamodel that has been created. Later, a discussion about the created metamodel and the architecture developed to create partial implementation from the specification are done in section six. Finally, in the last section conclusions and further work are addressed.

2. Related Research

In the last few decades research in the area of e-learning has evolved considerably with respect to specifications and standards related to digital content, as is demonstrated by the number of specifications and standards that have been developed. For example, Common Cartridge is a standard under which the IMS Consortium has grouped numerous specifications and standards such as Content Packaging v1.2 [16], Question & Test Interoperability v1.2 [17], IMS Tools Interoperability Guidelines v1.0 [15], IEEE Learning Object Metadata v1.0 [13] and SCORM v1.2 [1].

Also, research into learning systems platforms has undergone significant advances, as is demonstrated by the proliferation of systems of this type, whether learning systems (such as Claroline [9], LAMS [18] or SharePointLMS [30]) or course management systems (such as Dokeos [10], ILIAS [14], Moodle [22] or SAKAI [28]).

However, there are no mechanisms to represent processes in educational environments for the purpose of obtaining a certain degree of automation, as in business with, for example, Business Process Modeling (BPM) ([26], [20]). Neither is there any ontology when representing processes that makes it possible to formalise processes in educational environments or to promote them being shared, although there are ontologies that formally describe educational environments. For example there are ontologies to describe: learning content [29], interaction between students and learning systems in collaborative environments [21], learning tasks [27], objects of learning and working groups [4] as well as other elements involved in collaborative environment scenarios. As regards frameworks for the construction of learning environments, ELF [18] and OSID (Open Service Interface Definition) by OKI [25] are worthy of mention. Both have a clear orientation towards services and attempt to define what the services of an e-learning system should be.

ELF, also known as e-Learning Framework, has been developed by different international organisations with the aim of offering support for the development and integration of systems in the e-learning environment, for research, and for educational administration, which has currently been subsumed by the e-framework initiative. This framework is made up of three levels, one for common services, another for services related to the e-learning domain and finally, a third for users and agents.

The O.K.I. OSID specification is another framework aimed specifically at higher education communities and aims to offer a set of web service interfaces for the construction of learning systems for such environments that promotes interoperability between applications. Version 2.0 of this framework onwards includes packages to facilitate the development of e-learning systems in languages such as Java C#, but there is still no metamodel to describe this specification on a conceptual level and, thereby, facilitate its use.

3. The Proposed Method

This section describes a method that supports the development of eLearning applications independently of the LMS used. The proposed method uses a domain ontology to support the automatic creation of part of the implementation of the elearning applications from their specifications. The method transforms graphic specifications of learning processes in the part of the code that implements them, independently of the LMS reused and the final programming language. We can see a graphical sketch of the proposed architecture in figure 1.

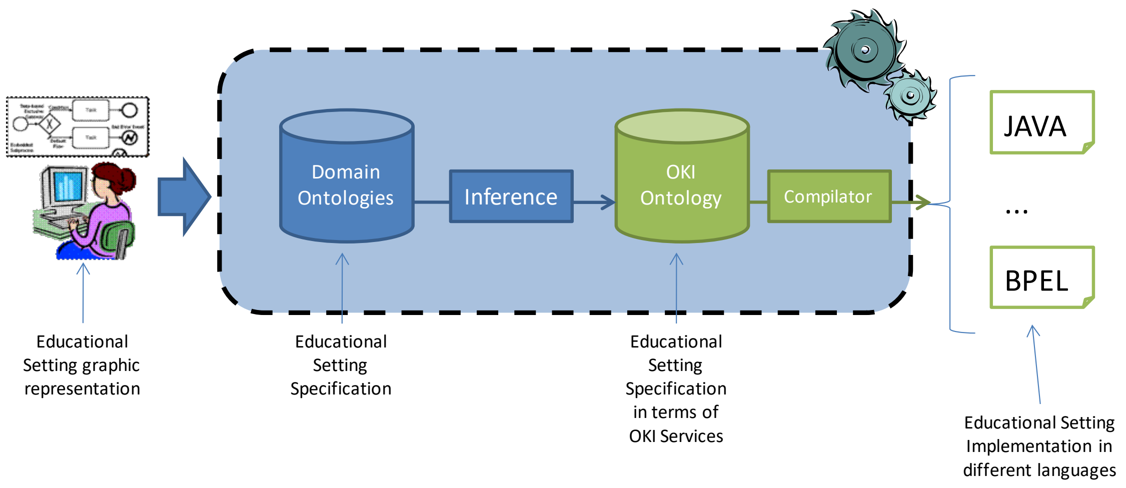
Domain Ontologies
Inference
OKI Ontology
Compilator
JAVA
...
BPEL
Educational Setting graphic representation
Educational Setting Specification
Educational Setting Specification in terms of OKI Services
Educational Setting Implementation in different languages

Figure 1: A method to create source code of eLearning processes from their specification

The proposed method can be seen as a black box that takes the specifications of the eLearning processes provided by the user and generates the source code that implement those processes in a given language (or specification). In order to provide that functionality the system is composed by a domain ontology and a compiler. In this paper we focus in the green elements of the figure, that is the elements that deal with the specification of learning processes in terms of OKI services and its final implementation (OKI ontology and the compiler). The graphical language and the interface that the designer uses in order to specify the learning processes are presented in [7] and [6].

The domain ontology allows specifying processes in the eLearning domain. For each process, the ontology allows representing its behavior, its actors (human or computer processes), its interactions, data sources used (queried and updated), etc. The representation of particular learning processes is done by the instantiation of such ontology. The instantiation may be performed taking into account a graphical specification of the processes entered by the user, such as in [7]. In order to provide a complete and mature interface that can represent implementation details of learning processes independently of the implementation language we have used the OKI-OSID specification. In particular, we created the ontology as a metamodel of the OKI-OSID specification.

For each learning process, stored in the domain ontology an inference engine processes a set of rules in order to translate the specification of the learning processes in terms of OKI services, creating several instances in the OKI ontology. Once the translation process is done, the OKI ontology will contain enough information to create a partial implementation of the specified processes. Thereafter, a compiler translates the knowledge stored in the ontology to source code that includes invocations to web services useful to implement the processes to automate in different programming languages. Even though the generated implementation is not complete, the system is able to generate most of the web services contracts and the calls necessary to implement the interactions between learning processes.

4 The Case Study

In order to exemplify our method and test it we used a real case, which is simple but representative enough to serve as a concept text. This case study consists of an educational scenario dealing with the assignation of teachers to a database subject of the Open University of Catalonia.

However, before describing the selected scenario as a case study, we need to define the concept of an educational scenario. In general, we understand an educational scenario to be a set of activities that occur in a learning environment with the aim of preparing that environment, offering support during the learning process and evaluating the results obtained and the competences acquired during the learning process. For example, educational scenarios could include the preparation of a course and the correction of activities or course evaluation.

Scenarios in educational environments tend to follow a common pattern, although they are adapted in one way or another depending on the institution, according to each particular organization and its rules. Therefore, the figures involved and the resources used to reach the established objective will vary according to the organisation context in question.

For example, before the start of the course any educational institution should assign teaching staff to the classrooms in which the course will take place. And a specific case of this generic scenario might be the scenario of Basic Consultancy Task (EBC in Catalan) Task Assignation for the Databases I course in the Autumn semester of 2010 at the Open University of Catalonia (UOC), which has been chosen as for the case study.

The Open University of Catalonia is an online university offering all types of courses including higher and postgraduate studies. Currently, more than 40,000 students are enrolled and among other qualifications it offers Technical Systems Engineering (ITIS) and Technical Management Engineering (ITIG).

One of the compulsory subjects in these courses is Databases I (BDI), which is taught in several classrooms given the high number of students enrolled. Due the practical nature of the subject it uses two types of classroom: theory classrooms and laboratory classrooms.

All subjects at the UOC are led by the lecturer responsible for the subject (PRA), who coordinates a team of teaching staff. The team members carry out the teaching and are known as consultants. The PRA, among other tasks, prepares the course and has to assign a consultant to each classroom. Assigning a classroom means that the consultant accepts responsibility for guiding the students assigned to the classroom during the learning process, correcting their activities and resolving any queries they may have. The UOC calls this a basic consultancy task (EBC).

The EBC Task Assignment scenario for the BDI course occurs when the system detects the need to assign consultants to classrooms. Therefore, for each course identified by subject and semester, consultants or expert PDCs in the subject are found who can teach the courses, whether on a theoretical or practical level, and this information is given, together with the number of classrooms of each type, to the PRA, who uses it to carry out an initial assignment proposal. This proposal is returned to the system, which, in turn, sends individual notifications of the assignment to each consultant for their acceptance. Once all the assignment acceptances have been collected, the EBC is established for the course and, therefore, can be registered in the UOC repository for payment purposes, at the same time as the system notifies the finalisation of the scenario.

From this description it is deduced that in this scenario the following teaching figures are involved: the PRA and the group of consultants, which, in this case, for the Catalan campus are three theory consultors and one for the laboratory. The required resources are the course data *(IdSubject_DBI, IdTerm_DBI*), the number of classrooms by type (*DefaultNumClassroomsByType_DBI*), the list of consultants of each type for BDI

(*ConsultantsByTypeList_DBI*), the list of consultant assignments to BDI classrooms (*AsignmentsConsultancyList_BDI*), the notifications of acceptance (*AssignmentToClassroom_DBI*), the confirmations of acceptance (*AssignamentAcceptance_DBI*), the list of EBC tasks for BDI (*TaskListEBC_DBI)* and the UOC repository where the available consultants for the course can be accessed and the assigned EBC tasks stored. The scenario can be represented as a single process known as *AssignEBCTasks_DBI_A10*

Graphically, the educational scenario described above can be represented as indicated in Figure 2.

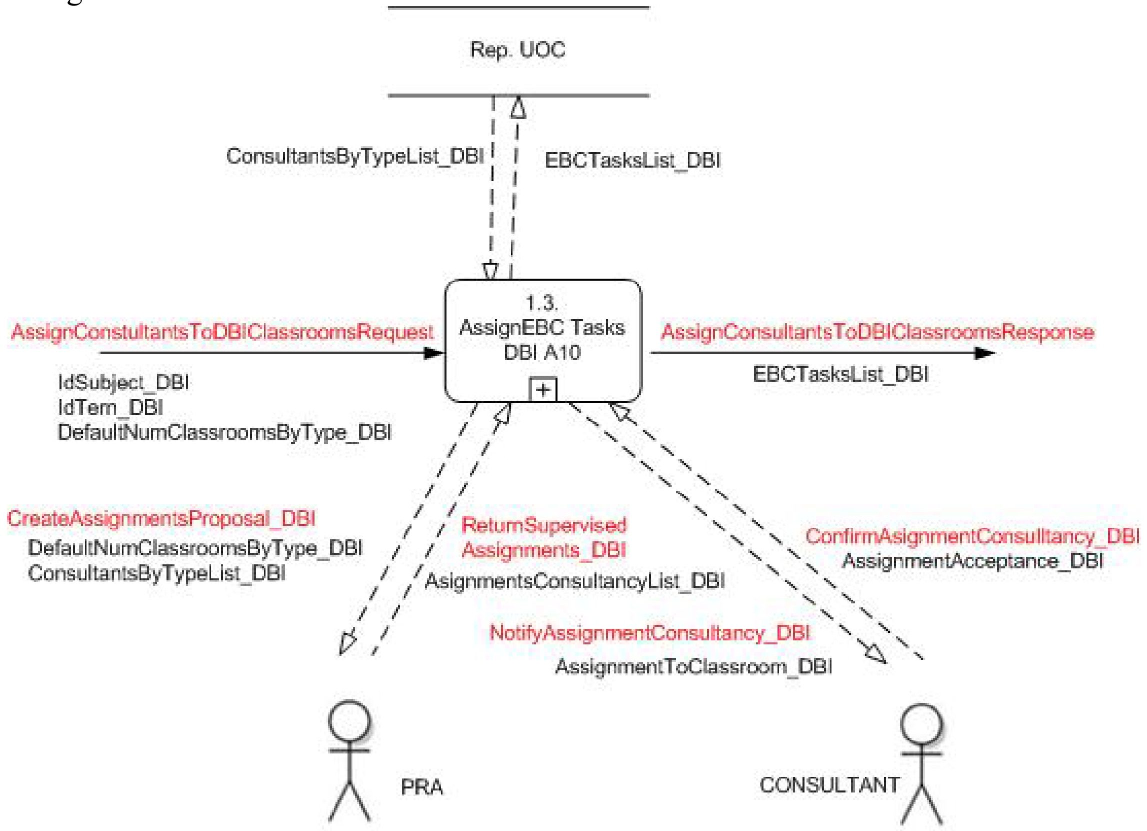
Rep. UOC
ConsultantsByTypeList_DBI
EBCTasksList_DBI
1.3. AssignEBC Tasks DBI A10
AssignConstultantsToDBIClassroomsRequest
IdSubject_DBI
IdTem_DBI
DefaultNumClassroomsByType_DBI
AssignConsultantsToDBIClassroomsResponse
EBCTasksList_DBI
CreateAssignmentsProposal_DBI
DefaultNumClassroomsByType_DBI
ConsultantsByTypeList_DBI
ReturnSupervised Assignments_DBI
AsignmentsConsultancyList_DBI
ConfirmAsignmentConsulltancy_DBI
AssignmentAcceptance_DBI
NotifyAssignmentConsultancy_DBI
AssignmentToClassroom_DBI
PRA
CONSULTANT

Figure 2 - EBC Task Assignment Scenario for the BDI course in the Autumn 2010 semester

In Figure 2 a representational language based on the standard BPMN [20], [26] is used as a notation, allowing the processes occurring in educational environments to be represented as sequences of processes that require the use of certain resources and the involvement of participants external to the system in order to perform the task. This language is described in detail in [7] and to facilitate its use a CASE tool has been developed to offer support to the specification of scenarios using this notation. More information about this tool can be found at [6].

5 The Domain Ontology: The Metamodel of the OKI-OSID Specification

The O.K.I. OSID specifications emerged in 2001 as an initiative of the Mellon Foundation to offer support to the development of LMS, which was later supported by MIT and other collaborating institutions. This specification defines an open architecture extendable for the construction of e-learning systems specially aimed at the higher education communities.

A principal contribution of this specification is the set of definitions of web service interfaces (Open Service Interface Definitions) proposed for communication between the components of a learning environment, whether part of the platform or between different platforms, designed to facilitate the adaptation towards new technology and integration with the rest of the technological infrastructure of learning systems.

There are different versions of this specification and, although there is a version 3, a metamodel of the second version (the last stable version of the specification) has been created, which offers a set of packages that facilitate the use of interfaces proposed in Java and C applications.

This section focuses on the metamodel of the O.K.I. OSID specification and presents, whose part required to test feasibility of the prototype for the case study. The full version of the metamodel can be obtained from [8].

Given the restrictions of space, only the OSIDs relevant for the case study are described. These OSIDs are:

a) **Agent** to define the roles of PRAs and consultants, as well as the different processes involved, such as Asign Tasks EBC BDI T10

b) **Scheduling** to define the consultancy tasks, specifically the EBC task and the assignment of said tasks to each consultant that will be teaching a course

c) **Course Management** to represent the assignment data, the semester in question, the course and the different course classrooms

d) **User Messaging** to describe the information that is exchanged in each interaction of the process with external elements, such as teaching staff

e) **Repository** to define the UOC repository and the information stored therein

Each of these OSIDs is defined through a set of properties and several associated methods per property and OSID. In order to facilitate its description, we present for each OSID:

1) An **UML class diagram** that graphically describes the classes of the OSID, the properties of each class, and the relationships between classes (both belonging to the OSID and external),
2) the classes contained in the OSID, and for each of them its properties and the necessary **rules** to **define derived properties** or **integrity constraints** using Semantic Rule Web Language (SWRL) and
3) the necessary **comments** regarding the **relationships** between classes and any **methods** that are not explicitly class constructors or getters and that need to be mentioned separately.

The derivation rules written in SWRL shown in this paper will be used by the reasoner in order to infer the instances of some classes and properties based in the information stored in other classes or properties.

5.1 OSID Agent

This OSID has been designed to create, manage and query agents, understanding an agent as a person or a process that interacts with the learning system. There are groups of agents that can contain other groups that are also managed and accessed through the interfaces of this package.

Figure 3 shows the class diagram for the O.K.I. OSID-v2 Agent package in which two classes can be distinguished: Agent and Group.

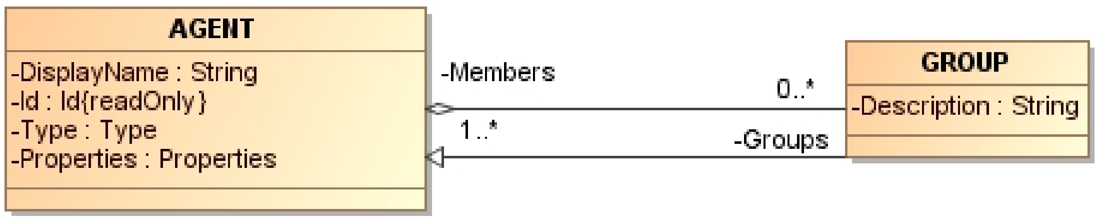

Figure 3 - ***OSID Agent*** class diagram

Agent class

An Agent is a body (individual or process) involved in specific services. Table 1 shows the properties of this class:

Property	Description
displayName	Denomination of an agent
id	Agent identifier, unique and non-transferable
properties	Properties of a course defined by implementation
type	Agent characterisation

Table 1: Properties of the Agent class

Even though the Agent class maintains relationships with many other classes, they are not mentioned in this section since navigability is in the opposite direction. They will be defined when the other classes are described.

One instance of the Agent class in the case study is the PRA in the DBI course (PRA_DBI) and each of the consultants (*TheoryConsultant_DBI_1, TeoryConsultant_DBI_2, TeoryConsultant_DBI_3, LaboratoryConsultant_BDI_1*), as well as the process *AssignEBCTaslks_DBI_A10.*

Group class

A Group is a type of specific agent representing the grouping of one or more agents under the same name or description.

Groups may contain other groups. Given that a group is a specialisation of an agent, it inherits all of its properties and also includes some new attributes, as shown in Table 2:

Property	Description
Description	Group description

Table 2: Properties of the Group class

An instance of a ***Group*** class is the BDI consultancy team (*ConsultancyTeam_DD*I) whose members are consultants for the BDI course.

Relationship Types

The O.K.I. OSID specification takes into account some relationship types between the classes Agent and Group: the agents that are members of a group (Members) and the groups where agents belong (Groups). Table 3 shows these relationship types:

Relationship types	Description	Domain	Range
Members	Members constituting a group	Group	Agent
Groups	Groups to which an agent belongs	Agent	Group

Table 3: Types of relationship of OSID Agent classes

Methods

This OSID offer interfaces for searching agents and groups using search criteria (getAgentBySearch, getGroupBySearch), by type (getAgentByType, getGroupByType), to check whether a group or group member is contained in a group (contains) and to determine which groups an agent belongs to (getGroupsContainingMembers).

5.2 CourseManagement OSID

The objective of the ***CourseManagement*** OSID is to support the creation and management of a course catalogue. Figure 4 shows the diagram of UML classes for this OSID.

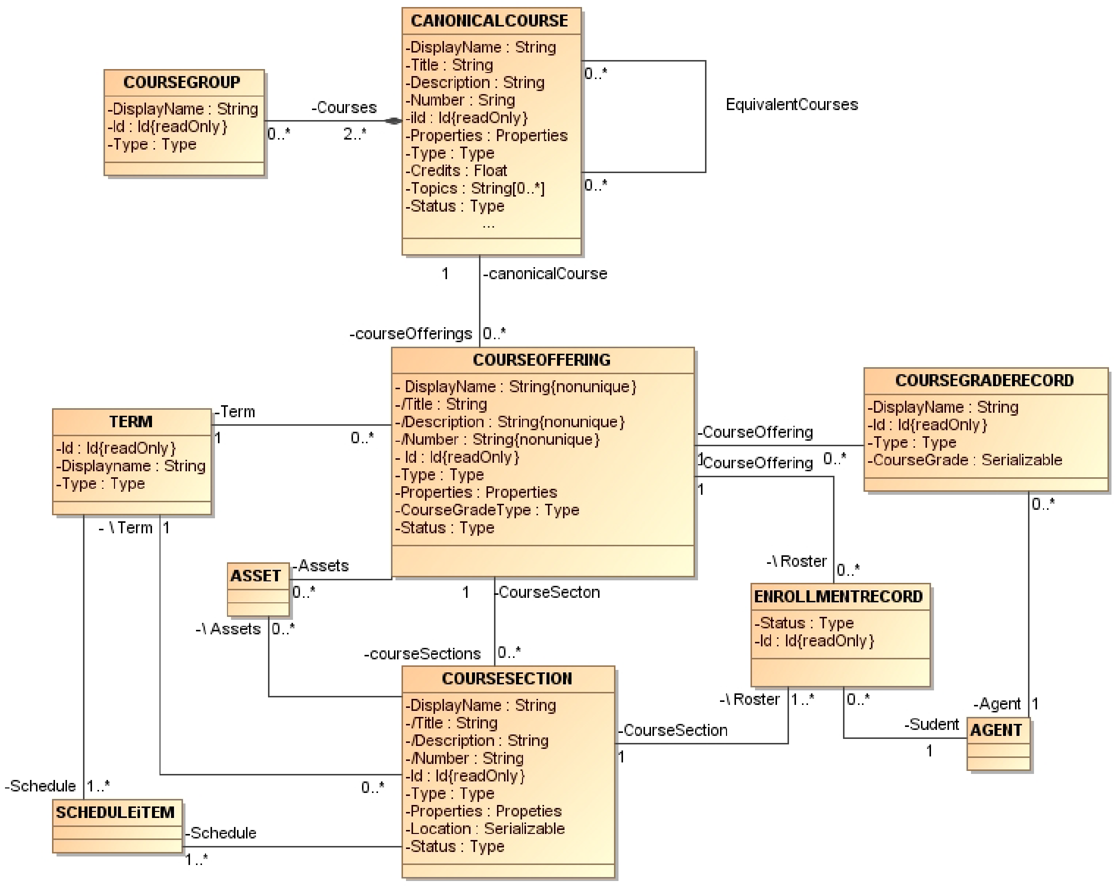

Figure 4: Diagram of classes of the CourseManagement OSID

The course catalogue is organised mainly around a three-level hierarchy of classes corresponding to the course concept (**CanonicalCourse**), a specific course in a certain semester (**CourseOffering)** and the classroom in which the specific course is offered (**CoursesSection**). However, it includes other classes such as **CourseGroup**, **EnrolmentRecord**, **CourseGradeRecord** and relates to other external classes to associate materials, evaluations, tasks or agents to each course.

Each of the classes that make up the UML class diagram in Figure 4, and which are relevant to the case study, are described below.

CanonicalCourse class

A *CanonicalCourse* defines the highest-level organisational structure in the CourseManagement package. A canonical course can be understood to represent the concept of a subject that is defined at study plan level, i.e. independently of specific academic courses and classrooms in which they are taught, and, within this context, in relation to other subjects in the same study plan.

All canonical courses are characterised by having a description, a title, a course number, a number of credits and a list of topics, in addition to other properties of the course such as the course type or status. Table 4 presents the properties of this class.

Property	Description
DisplayName	Denomination of a canonical course
Title	Course title
Description	Description of canonical course
Number	Course number
Id	Course identifier
Properties	Properties of a course defined by implementation
Type	Course type
Credits	Number of course credits
Topics	Course topics
Status	Course status

Table 4: Properties of the CanonicalCourse class

In the case study, an instance of a canonical course might be the *Databases I* course taught for two qualifications: *Computer Management Engineering* and *Computer Systems Engineering*. Therefore, an instance of this class is *Databases I* (*DBI*) for which the identifier, name, description, type and number of credits will be defined.

CourseOffering class

A *CourseOffering* is essentially the specification of a canonical course with a type of assessment for a definite academic period, in other words, a subject that is taught over a specific academic period. Table 5 shows the properties of this class.

Property	Description
DisplayName	Denomination of course offered
Title	Course title
Description	Description of course offered
Number	Course number
Id	Course identifier
Properties	Properties of a course defined by implementation

Type	Course type
CourseGradeType	Course evaluation type
Status	Course status

Table 5: Properties of the CourseOffering class

The properties *Title*, *Description*, and *Number* can be automatically derived from those corresponding to the *CanonicalCourse* or they may be redefined. Also, all courses offered have an evaluation type defined and have an associated course type and status indicator.

An example of an instance of this class is *Databases I* for the *Autumn semester of 2010 (DDI_A10)*.

Term class

A *Term* is an academic period that corresponds to a specific type. It should be understood as a period in which a set of tasks needed to teach a course is planned. Table 6 shows the properties of this class.

Property	Description
DisplayName	Denomination of academic course
Id	Academic course identifier
Type	Course type

Table 6: Properties of the Term class

One example of this class is the Autumn semester of 2010 (A10).

CourseSection class

A *CourseSection* or classroom is the specification of a course offered with a fixed location where the course is taught according to a given plan.

In the same way as for the course offered, the properties *Title*, *Description*, and *Number* can be derived from those corresponding to the *CanonicalCourse* class or they may be redefined on creating the classroom. Location of the course is fundamental in this class and is determined by the value of the *Location* property. It also has other properties, such as type or course status, associated with it. Table 7 shows the properties of this class.

Property	Description
DisplayName	Denomination of a classroom
Title	Classroom title

Description	Classroom description
Number	Classroom number
Id	Classroom identifier
Properties	Properties of the classroom to be defined by implementation
Type	Classroom type
Location	Classroom location
Status	Course status

Table 7: Properties of the CourseSection class

For the case study, there are as many instances of CourseSection as there are *Database* classrooms. This would create instances of theory type classrooms (Theo*DBI_1, TheoDBI_2, TheoDBI_3*) and instances of Laboratory type classrooms (*Lab_DBI_1*) – as many as needed for each type.

Relationship types

In the table 8 we present the relationship types among the *CourseManagement* OSID classes and to other external classes.

Relationship types	Description	Domain	Range
EquivalentCourses	Equivalent canonical courses	CanonicalCourse	CanonicalCourse
Courses	Canonical courses that form a group of courses	CourseGroup	CanonicalCourse
CourseOfferings	Courses offered associated with a canonical course	CanonicalCourse	CourseOffering
CourseSections	Classrooms associated to a course offered	CourseOffering	CourseSection
Term	Academic course associated to a course offering or classroom	CourseOffering, CourseSection	Term
CanonicalCourse	Canonical course corresponding to CourseOffering	CourseOffering	CanonicalCourse
CourseOffering	Course to which a classroom, course grade record or enrolment record is associated	CourseSection, CourseGradeRecord , EnrollmentRecord	CourseOffering
CourseSection	Classroom with which an enrolment record is associated	EnrollmentRecord	CourseSection
Assets	Materials associated to a course	CourseOffering,	Asset

	offering or a classroom	CourseSection	
Roster	Enrolments associated to a course or classroom	CourseOffering, CourseSection	EnrollmentRecord
Schedule	Planned tasks associated to a classroom	CourseSection	ScheduleItem
AgentId	Agent that defines a course grade record or that is enrolled on a classroom of a course	CourseGradeRecord , EnrollmentRecord	Agent

Table 8: Types of relation of CourseManagement OSID classes

Apart from the constraints that can be seen in figure 3 the *equivalentCourses* relationship is symmetric and transitive, and the relationships *term* and *assets* are symmetric.

Canonical courses may have equivalent canonical courses (EquivalentCourses) and each subject or canonical course can be offered on more than one course (CourseOfferings).

For each course offered there are relationships that means it is possible to determining the canonical course with which it is associated (CanonicalCourse), which academic course it belongs to (Term), which materials it uses (Assets), which students are enrolled on it (Roster) and in which classrooms it is taught (CourseSections).

The classrooms or CourseSections are related to the course they teach (CourseOffering), the academic period in question (Term), the course planning - understood as a set of tasks (Schedule) - the set of materials used (Assets) and all student enrolments (Roster).

The relationship between a canonical course or subject with respect to a group of courses is determined by the relation (Courses) in such as way that the courses that make up each group of courses can be known. The O.K.I. OSID specification does not provide interfaces for navigability in the opposite direction to find out which group of courses belong to a canonical course.

An assessment record belongs to a course offered (CourseOffering) and is defined by the evaluating agent (AgentId). O.K.I. OSID does not consider these relations in the opposite direction and, therefore, does not offer interfaces to find out which courses have been defined by an agent and what the assessment of a course was.

Neither is there a direct relationship between the students and the courses or classrooms. However, it is possible to know which students have been assigned to a classroom or course and those that are on a course or in a classroom by looking at the instances in the *EnrollmentRecord* class. The same applies course evaluators, a list of which can be obtained from the instances in the CourseGrade Record class.

Rules

The derivation rules that allow inferring the instances of the derived relationship types specified in the class diagram of Figure 3 are shown below:

[Rule-DescriptionOfCourseOffering] The description of a course offered may be derived from the description of the canonical course with which it is associated.

```
description(?ca, ?d)  ∧  courseOfferings(?ca, ?co) → description(?co, ?d)
```

[Rule-DescriptionOfCourseSection] The description of the classroom may be derived from the description of the canonical course with which the course taught in that classroom is associated.

```
description(?ca, ?d)  ∧  courseOfferings(?ca, ?co)  ∧  courseSections(?co, ?cs) →
description(?cs, ?d)
```

[Rule-TitleOfCourseOffering] The title of the course offered may be derived from the title of the canonical course with which it is associated.

```
title(?ca, ?t)  ∧  courseOfferings(?ca, ?co) → title(?co, ?t)
```

[Rule-TitleOfCourseSection] The title of a classroom may be derived from the title of the canonical course with which the course taught in that classroom is associated.

```
title(?ca, ?t)  ∧ courseOfferings(?ca, ?co)  ∧  courseSections(?co, ?cs) →
title(?cs, ?t)
```

[Rule-NumberOfCourseOffering] The number of the course offered may be derived from the number of the canonical course with which it is associated.

```
number(?a, ?n)  ∧  courseOfferings(?a, ?co) → number(?co, ?n)
```

[Rule-NumberOfCourseSection] The number of the classroom may be derived from the number of the canonical course with which the course taught in that classroom is associated.

```
number(?ca, ?n)  ∧  courseOfferings(?ca, ?co)  ∧  courseSections(?co, ?cs) →
number(?cs, ?n)
```

[Rule-TermOfCourseSection] The classrooms in which a course is taught have the same academic period associated with them as the courses that are taught in them.

```
term(?co, ?t)  ∧  courseSections(?co, ?cs) → term(?cs, ?t)
```

Methods

Interfaces are offered for consultation methods that allow group courses, canonical courses, course offerings, classrooms and enrolment records to be obtained by type (getCourseGroupByType, getCanonicalCourseByType, getCourseOfferingByType, getCourseSectionByType, getRosterByType). Academic periods can also be obtained from a specific date (getTermByDate).

There are also methods for the updating of descriptions (updateDescripton), titles (updateTitle) and numbers (updateNumber) of a course offered or a classroom.

5.3 Scheduling OSID

The objective of the *Scheduling* OSID is the association of agents to specific tasks that are assigned to them in a period of activity. For each task and agent involved a status is determined. The status indicates the level of commitment acquired by the agent in carrying out the task, as shown in Figure 5.

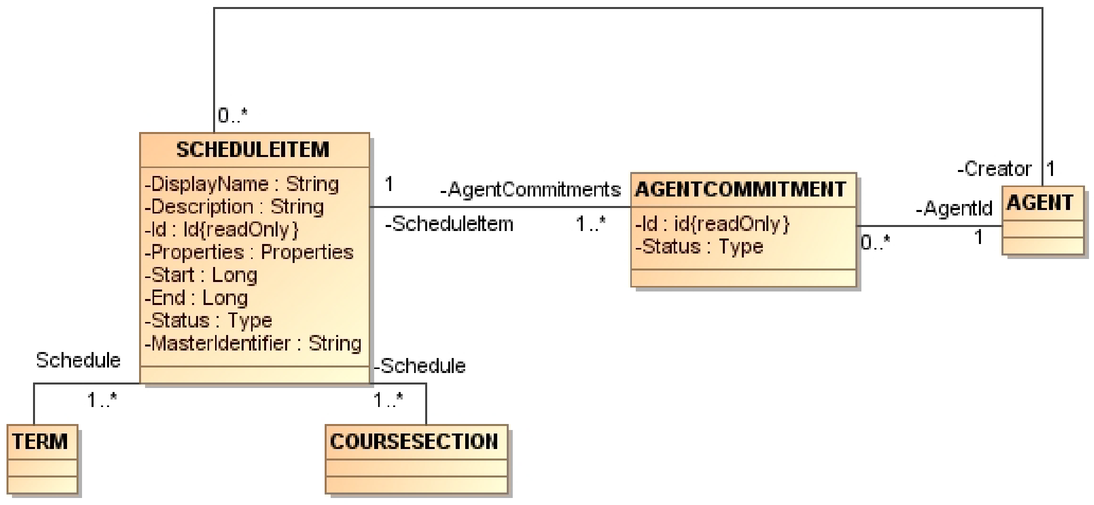

Figure 5: Diagram of classes corresponding to the Scheduling OSID

ScheduleItem class

The *ScheduleItem* class captures information relating to each of the tasks that must be carried out.

A scheduled item or task is characterised by the period in which the task must be carried out and the agents that are responsible for carrying it out. Table 9 shows the properties of this class.

Property	Description
DisplayName	Denomination of a scheduled item
Description	Description of a scheduled item
Id	Identifier of a scheduled item
Properties	Properties of a scheduled item defined by implementation
MasterIdentifier	Optional identifier provided by another application
Start	Start date
End	End date
Status	Scheduled item status

Table 9: Properties of the ScheduleItem class

One example of this type of class could be the EBC task at the UOC.

AgentCommitment class

AgentCommitment records the degree of commitment of each agent in the task in which they are participating. It is, therefore, a question of maintaining and managing the assignment of people to tasks and the completion status of the tasks, irrespective of the characteristics of the task and the period of time when they will be carried out, which are described in the *ScheduleItem* class.

The identifier properties of the assignment and the status of the assignment are the properties of this class, as shown in Table 10.

Property	Description
Id	Scheduled item identifier
Status	Status indicating the acquired level of commitment

Table 10: Properties of the AgentCommitment class

One example of this class could be the assignment of the EBC tasks for each consultant in the Database I course (*assignmentEBCTheoClass1_BDI, assignmentEBCTheoClass2_DBI, asignmentEBCTheoClass3_DBI, asignmentEBCLabClass1_DBI*).

Relatioship Types

The relationship types for the classes in this OSID are detailed in Table 11.

Relationship typesship	Description	Domain	Range
AgentId	Agent to whom a task is assigned with a certain level of commitment	AgentComitment	Agent
ScheduleItem	Scheduled task	AgentCommitment	ScheduleItem
Creator	Agent who schedules the assignment of the task	ScheduleItem	Agent
AgentCommitments	Responsibilities assigned in the scheduling of a task	ScheduleItem	AgentCommitment

Table 11: Types of relation for the Scheduling classes OSID

For each task scheduled and agents involved there is a commitment to carrying out the task on the part of the agent. Each responsibility, or agent commitment, assigned is associated with an agent (AgentId) and has a degree of commitment, which could be the same for carrying out different tasks. However, a scheduled task may have different agency commitments associated with it, as many as the number of agents involved.

What the O.K.I. OSID specification does not contemplate is the relationship of tasks in which a committed agent may be found and all the tasks with the same level of commitment.

Methods

The OSID provides interfaces for query methods that allow quering the scheduled items from their identifier (getScheduleItem). On the other hand, some methods that allow finding gaps in the schedules of several agents are also provided (getAvailableTimes). These last methods may be very useful before creating a scheduled item and for finding out the tasks assigned to a group of agents during a certain period (getScheduleItemsForAgents).

5.4 User Messaging OSID

The *User Messaging* OSID provides interfaces for managing the sending and receipt of messages as well as subscriptions for the receipt of messages associated with a theme or messages of a certain type.

Given that messaging takes place between agents, this OSID is directly related to the *Agent* OSID, whose *Agent* class must be expanded to make the sending/receipt of messages by subscription possible. Figure 6 presents the class diagram for this OSID.

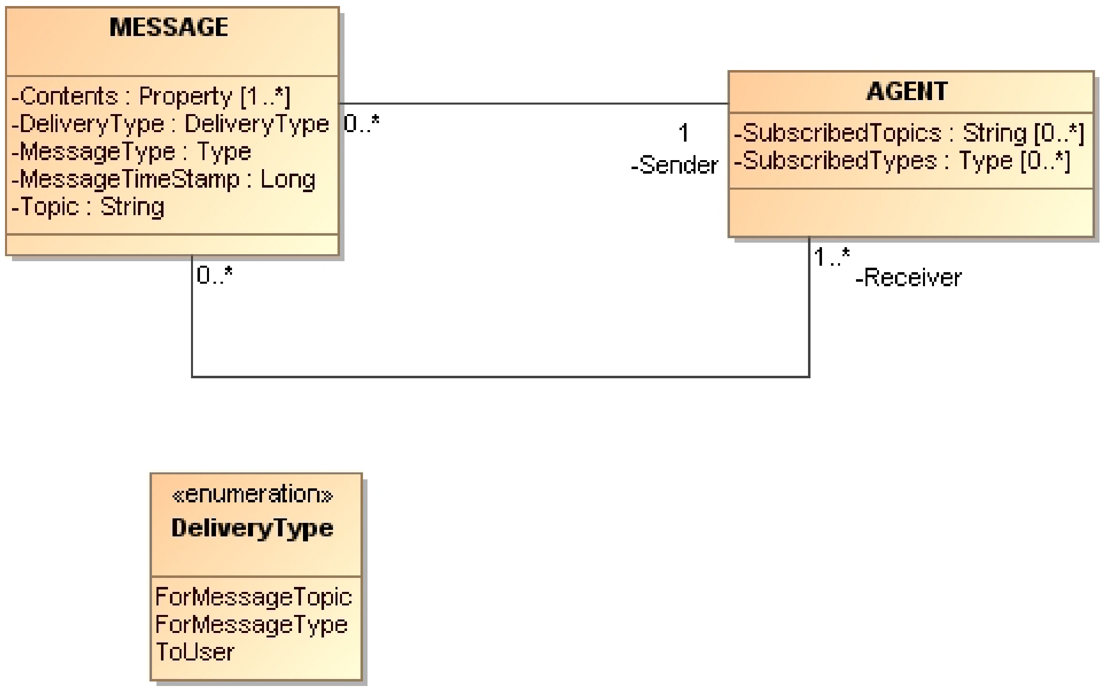

Figure 6: Class diagram for the UserMessaging OSID

It can be seen that a message is always sent by a single agent and can be received by several agents.

Message class

A *Message* is anything sent by an agent at any particular time. The properties of this class are presented in Table 12.

Property	Description
Content	Message content
DeliveryType	Delivery type
MessageTimeStamp	Time of sending
MessageType	Message type

Table 12: Message class properties

All messages have an associated topic, message type and delivery type. The delivery type may be by topic (sent to all subscribers to the topic), message type or by one or more users.

An example of this class could be notification of the assignment consultancy to each of the consultants on the Database I course (*NotifyAssignmentsConsultancy_DBI*) or the confirmation of such assignments (*ConfirmAssignmentConsultancy_DBI*), the content of which is a list of assignments to classrooms (*AssignmentsToClassrooms_DBI*) and the acceptance that is received from each of the consultants (*AssignmentAcceptance_DBI)*.

Relationship Types

The Message class is only related to the Agent class from the *Agent OSID*. All messages are sent by a single agent (Sender) and can be received by several agents (Receiver) as shown in Table 13.

Relationship types	Description	Domain	Range
Sender	Agent who sends the message	Agent	Agent
Receiver	Agent to whom the message is sent	Agent	Agent

Table 13: Types of relationship of the User Messaging OSID

It should be remembered that in order to make it possible to receive messages by topic or by message type there has to be a subscription. For this purpose, as commented above, the *Agent* class, which is external to this OSID, has been extended to offer two new properties that allow subscription by an agent to different topics (SubscribedTopics) and delivery types *(SubscribedTypes)*. It should be noted that subscription is optional and these properties, therefore, have no cardinality restrictions.

Methods

This OSID offers interfaces for the management of subscriptions (Subscribe/Unsubscribe) by an agent or their elimination (UnsubscribeAll).

Additionally, like consultation method interfaces, it offers the chance to find out which agents are subscribers (getSubscribers), which agents have subscribed to a certain topic (getSubscribersByTopic) and which messages have been received by topic (getReceiveByTopic), or by message type (getReceiveForMesageType).

It also provides interfaces to find out which messages have been sent by an agent (Send) or by a group of agents (SendToAll) or to purge specific messages (purgeMessage).

5.5 Repository OSID

The objective of the *Repository* OSID is to provide interfaces for storage and retrieval of all types of digital content and any information related to that content and/or its composition.

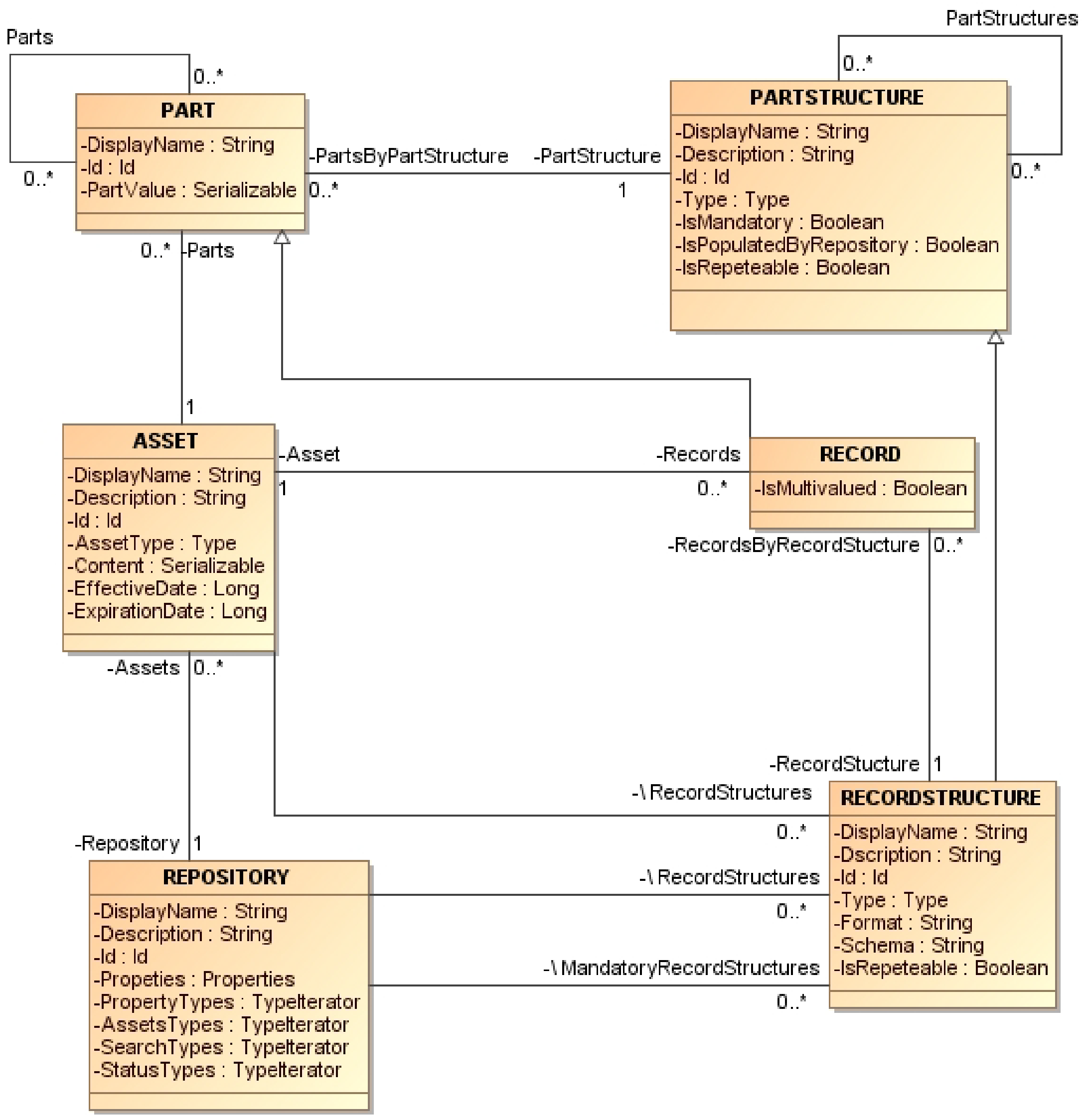

Figure 7: Diagram showing classes corresponding to the Repository OSID

The digital content is known as Asset and its composition may be very diverse. So to describe an Asset it may be necessary to use other classes such as: *Part*, *PartStructure*, *Record* and *RecordStructure*, which allow the structure and content of the contents stored in the repository to be determined. Figure 7 presents a diagram showing the classes of this OSID.

Asset class

An *Asset* is a digital resource stored in a repository that has a fixed period of validity.

The content of an asset may be made up of: 1) content only (e.g. a summary document for a subject), 2) content records and record structures describing the content (e.g. a semantically annotated PAC) and 3) only metadata described using record structures (e.g. metadata describing an examination format).

All assets have a type and content in addition to other properties which are presented in Table 14.

Property	Description
DisplayName	Denomination of digital resource
Description	Description of digital resource
Id	Identifier of digital resource
AssetType	Type of digital resource
EffectiveDate	Date of effect
ExpirationDate	Expiration date
Content	Content

Table 14: Properties of the Asset class

Instances of this class could be the list of consultants of each type for the Database I course (*ConsultantsListRetreived_DBI*) or the list of EBC task assignments to consultants for the same course (*EBCTaskSaved_DBI*).

Repository class

A *Repository* is a storage facility for digital resources, or parts of them that allows certain types of assets to be stored and managed.

The properties that define a repository are basically related to the type of digital content and the criteria for their localization. This is shown in Table 15.

Property	Description
DisplayName	Name of a register structure

Description	Description of a register structure
Id	Register identifier
Course type	Type of repository
Properties	Properties of repository defined by implementation
PropertyTypes	Property types
AssetTypes	Types of digital resources
SearchTypes	Search types
Status	Status for each search

Table 15: Repository class properties

Relationship types

The relationship types established between the classes in the Repository OSID and other external classes are presented in Table 16 and are described below.

Relationship types	Description	Domain	Range
Repository	Repository in which the digital resource is stored	Asset	Repository
Assets	Digital resources stored in a repository	Repository	Asset
Parts	Digital resources components of an asset or part of an asset	Asset	Part
Records	Records of an asset	Asset	Record
Asset	Asset corresponding to a record	Record	Asset
RecordStructure	Record structure that determines the organisation of a record	Repository	RecordStructure
RecordsByPartStucture	Records that have a determined record structure	RecordStructure	Record
PartStructure	Structure of part of an asset	Part	PartStructure
PartsByPartStructure	Parts that have a structure of part of a determined asset	PartStructure	Part
PartStructures	Structure of a part of an asset that contains other structures of parts of an asset	PartStructures	PartStructures
RecordStructure	Structure of a record that corresponds to a record	Record	RecordStructure
RecordStructures	Structures of records that recognise a repository or define an asset	Repository, Asset	RecordStructure

MandatoryRecordStructures	Obligatory record structures for the assets in a repository	Repository	RecordStructure

Table 16: Types of relationship in the Repository OSID

Apart of the cardinality constraints shown in figure 7, the relationship types *parts* and *partStructures* have also a constraint that defines their transitivity.

An asset is stored in a Repository and may be made up of Parts which, in turn, may be made up of other Parts. An asset is not made up of content only, and will have more than one Record and each of these records will have structure (RecordStructures) describing how the records making up the asset are organised.

Parts of digital resources or parts of assets have an associated record structure, i.e. information about the organisation of the asset's parts (PartStructure). These record structures for asset parts may in turn be made up of other record structures (PartStructures).

The records correspond to an Asset and have a record structure (RecordStructure). Given that the records are parts of an asset that have some part structure associated with them, which, in turn, may be made up of other part structures, it is possible to derive all the record structures corresponding to an asset through their transitivity.

The repositories are of a certain type and recognise certain record structures, some of which are obligatory (MandatoryRecordStructures).

Rules

The set of rules defined for this OSID are not required for the case study and, therefore, are not included in this section because of space restrictions. However, they can be found in [8] together with the extended O.K.I. OSID metamodel.

In our case we used rules to express model constraints or to derive new data according to the OKI metamodel. Thus such rules can be also considered as part of the parser mechanism that translates educational settings descriptions into OKI descriptions. Each SWRL rule may be divided in two parts: the antecedent and the consequent. When the antecedent is hold true then the consequent should also be true. When that happens, if the consequent is not in the information base then the information base is extended with the facts stated in the consequent. The created rules derives information about educational settings to be added to the OKI construct. In that way, the antecedent holds when there is enough information in order to generate an OKI construction from the educational settings definition, and then the OKI construction is created using such information.

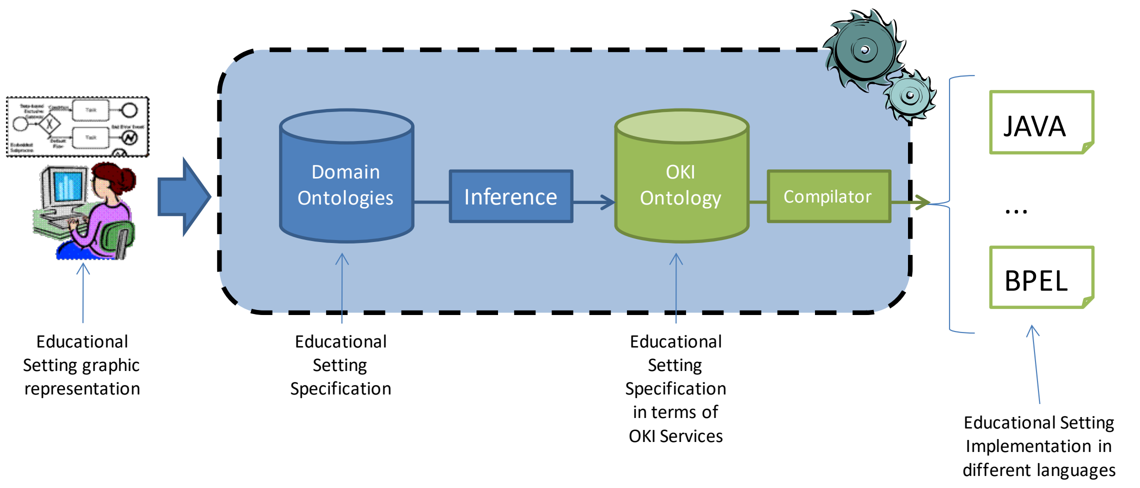
Domain Ontologies
Inference
OKI Ontology
Compilator
JAVA
...
BPEL
Educational Setting graphic representation
Educational Setting Specification
Educational Setting Specification in terms of OKI Services
Educational Setting Implementation in different languages

Figure 8: Schema of the prototype created to generate source code of eLearning processes from their specification

6 Implementation

The prototype created to test the feasibility of the proposed model allows for the partial implementation of educational scenarios described in accordance with the O.K.I. OSID specification.

This section describes the technology used to construct the proposed environment (figure 8); the results obtained using the prototype developed for the case study described above, and its validation.

6.1 The OKI Ontology

The prototype developed is based in an ontology that made it possible to formalise the semantics of the metamodel, in OWL + SWRL, of the O.K.I. OSID specification. This formalisation aims to facilitate the transcriptions of the specifications to code and, specifically, in terms of web service invocations defined by O.K.I. OSID for use in C# or Java programs.

The O.K.I. ontology was created in OWL and extended with SWRL and can be downloaded from http://personal.uoc.edu/personalonto/ontologies. This metamodel formalisation was chosen because it facilitated the representation of knowledge through formal languages and their sharing, since both OWL and SWRL permit interoperability of specifications. OWL is a first-order logic language based on XML, while SWRL is a combination of OWL-Lite (the most basic part of OWL that guarantees computability) and RuleML, which enables rules to be expressed using Horn clauses in high-level language, which is especially useful for the objective pursued here.

The Protégé ontology editor (an open code, multiplatform editor and one of the most frequently used nowadays) was used to facilitate the creation of the ontology. It also offers facilities for the installation of *plug-ins*, which allow graphic visualisation of the ontology, and the editing of SWRL rules, which are of special interest for building the prototype.

6.2 Technical details

Once created and instantiated based on the case study, knowledge can be extracted from the ontology by obtaining a list of web service invocations so that the prototype acts as a compiler of OWL+SWRL for C#.

C# was chosen to take advantage of the packages of web service definitions offered by the O.K.I. OSID specification and to continue with the .NET that was used in the construction of the CASE tool, which facilitates the graphic specification of scenarios so that a prototype integrating them can be constructed in the future.

6.3 Generation of partial implementations

The generation of partial implementations of educational scenarios using the prototype consists in the incremental instantiation of the O.K.I. ontology and the automatic extraction of knowledge.

This starts with the graphic representation of the educational scenario in accordance with the notation described in [7]. To do this, each participant and process is represented as an instance of the Agent class and each arrow on the diagram represents an exchange of information between the process and the participants. or the process and the resources, provoking the instantiation of one of the other classes of the ontology, according to type and correspondence with the O.K.I. OSID metamodel described in the previous section. The O.K.I. ontology for each case study may be downloaded from http://personal.uoc.edu/personalonto/ontologies.

In general, instantiation allows us to:

1) Define the types of parameter of the different invocations or the values of the methods that are invoked.

   For example, in the case study there are an instance of the class *Repository* called *RepUOC*, an instance of the class *Asset* called *ConsultantsListRetrieved_DBI*, and a relationship between them. Such asset has a content that can be used in a message. In our case example, the *ConsultantsListRetrieved_DBI* asset contains *ConsultantsListByType_DBI*, which is used in *CreateAssignmentsProposal_DBI message*. If our interest is in obtaining a list of consultants so that the PRA can make a proposal for classroom assignation to BDI consultants, it will suffice to create an instance of the Asset class to represent the getAsset method.

2) Define the methods to be used.

   For example, the *CreateAssignmentsProposal_DBI* message, which goes from the process (*AsignEBCTasks_DBI_A10*) to the PRA, would be represented in the ontology as an instance of the Agent class with the value Send, corresponding to the method needed to deliver the message. On the other hand, we need to have defined the message to be delivered, which would contain a list of consultants by type (*ConsultantsByTypeList_DBI*), and the number of classrooms of each type

(*DefaultNumClassroomsByType_DBI*), so that the implementation of an invocation for the message Send can be deduced, the content of which would be that of the *CreateAssignmentsProposal_DBI* message.

Every time the instantiation follows an interaction or the use of a resource there should be an extraction of knowledge from the ontology. To do this, the use of an OWL+SWRL to C# compiler is proposed, which uses the information stored in the ontology and the packages provided by the O.K.I. OSID specification, to allow the generation of the list of web service invocations implemented by this information exchange or resource use.

```
// Code fragment generated automatically. Since code is partial the programmer should finish some parts of the code.
// Elements between @element@ are gaps that the programmer should fill in with the necessary information.

public @EBCTasksList_DBI_Type@ Assign_EBC_Tasks_DBI_A10( @IdSubject_DBI_Type@ idSubject_DBI,
              @IdTerm_DBI_Type@ idTerm_DBI, @DefaultNumClassroomsByType_DBI_Type@ DefaultNumClassroomsByType_DBI )

{
       // return variable is created
       @EBCTasksList_DBI_Type@ response = new @EBCTasksList_DBI_Type@();

   // Repositories necessary for the educational setting are obtained
   Repository Rep_UOC = new Repository( @parameters for the REP_UOC repository@ );

   // message bound to PRA is prepared and sent
   @ConsultantsByType_DBI_Type@ ConsultantsByType_DBI = Rep_UOC.getAssetBySearch( @SearchCriteria@, @TypeSearch@ ).getContent();
   @DefaultNumClassroomByType_DBI@ DefaultNumClassroomByType_DBI = @get DefaultNumClassroomByType_DBI@;
   PRA pra = @get PRA@;
   UserMessagingManager.send(
                             pra,
                             ConsutancyByType_DBI.toString() + DefaultNumClassroomByType_DBI.toString(),
                             “CreateAssignmentsProposal”,
                             “Topic”
                             );

   // message from PRA is received and processed
       @AssignmentsConsultancyList_DBI_Type@ AssignmentsConsultancyList_DBI = UserMessagingManager.Receive();

   // message bound to CONSULTANT is prepared and sent
   CONSULTANT consultant= @get CONSULTANT@;
   @AssignmentToClassroom_DBI_Type@ AssignmentToClassroom_DBI = @get AssignmentToClassroom_DBI@;
   UserMessagingManager.send (consultant, AssignmentToClassroom_DBI,  “NotifyAssignmentConsutlancy_BDI”, “Topic”)

   // message from CONSULTANT is received and processed
       @AssignmensAceeptance_DBI_Type@ AssignmensAceeptance_DBI = UserMessagingManager.Receive();

   // response of the educational setting is calculated
   @EBCTaskList_DBI_Type@ EBCTaskList_DBI = @get EBCTaskList_DBI@;
   response = DigitalRepositoryManager.CreateAsset(EBCTaskList_DBI, @description@, “assetType”);


   // Repositories are updated
   RepUOC.addAsset( EBCTaskList_DBI_Asset.getId() );


       return response;
}
```

Figure 9: The fragment of code automatically generated from the educational setting Assign_EBC_Tasks_DBI_A10. Since it is a partial implementation there are some parts indicated between @@ that should be filled by the programmers.

Therefore, the compiler should identify the instances of the ontology that refers to the parameters for each O.K.I. method. Later, the values of the parameters (taken from the ontology) will be used as parameters in the C# interface that describes the OKI method. It may be that not all formal invocation parameters can be substituted, in which case the invocation should be completed manually. We can see in Figure 9 an example of a fragment of generated code.

Consequently, once the extraction of knowledge from the ontology has been carried out incrementally, respecting the logical sequence described by the scenario, the web service invocation required to implement each scenario can be obtained automatically and, since they are defined in accordance with the O.K.I. specification, are valid for any learning system.

6.4 Prototype Validation

The validity of the O.K.I. ontology has been tested on several levels. The Pellet reasoner allowed for automatic verification that it is well written on a formal level and that it contains no contradictions. The instantiation through the case study enabled us to test its satisfactibility; however its completion has not been tested as it is an ontology for open environments [2]. As regards the extraction of knowledge by the compiler it allows partial implementations of educational environments to be generated and therefore their usefulness.

7 Lessons learnt

The creation of the O.K.I. ontology required an in-depth study of the specification, which made it possible to detect a number of limitations. In the following we comment the limitations found in the specification and how they affect to the possible use of the OKI-OSID:

- The ***Agent*** OSID does not contemplate the possibility that an agent might have more than one role in a scenario, which is necessary in order to describe complex processes in which the same participant can play two or more roles. For example, in the case study the PRA could play two roles, one as the lecturer responsible for a subject and the other as a consultant of the subject. In such a case, the OKI-OSID specification would need to create two agents for the same participant in order to let the same person act as a PRA and a consultant in the same scene.

- The ***Assessment*** OSID does not allow sharing published activities among the different classrooms of the same course. In the OSID specification each activity is only associated to a classroom, making not possible to associate an activity to either a course or all the classrooms of the given course.

- The ***Course Management*** OSID does not allow assigning more than one assessor to a course. Thus the OKI specification does not allow the evaluation of the each course activity by a different evaluator independently. In addition there's a lack of interfaces related to the management of courses. For instance, the OKI specification does not provide interfaces to know which courses or classrooms belong to a given academic period or which course a student has enrolled for, although the latter can be deduced. The same occurs with the qualifications of a course, the evaluations carried out by a specific evaluator, the delivered activities belonging to a student, the evaluations carried out by an evaluator or the courses where a student is enrolled. However, this information can be retrieved by means of a search, for example looking for, among all the delivered activities, the activities delivered by a student.

- The **Grade OSID** does not allow defining evaluation objects at the course level or for a group of activities of a course. So is due to the fact that the evaluation object is associated to each activity of each course. Then this metamodel don't let to share evaluation criteria or rules among different classrooms of the same course, or similar activities of a given course independently.
- The ***Scheduling*** OSID does not support scheduling of tasks made up of different stages unless it is done together with the Workflow OSID. The preparation of question tests independently to be joined at the end is not allowed, for instance.
- The ***Repository*** OSID does not consider certain forms of assets used in other standards such as hierarchical or sequential composition.
- The ***User Messaging OSID*** should extend the Agent class in order to allow sending and receiving messages by either topic or type message.
- The **Workflow OSID** is not useful to describe flows of processes with the aim of achieving a further automation. Some of the noticed lacks are the following:
    - The initial conditions and the output states are expressed in a textual way. This is not useful to align preconditions, postconditions and the automatic processing.
    - Reusability of processes, tasks and steps are not allowed.
    - It does not seem suitable to achieve modularity. There isn't possible to establish any kind of relation between pairs of processes and therefore, it does not allow process composition.
    - Each step is associated to only one role. Thus, it does not permit representing complex processes in which participants can assume more than one role.
    - It requires considering the OSID Authorization. Such OSID describe a service of lower level than those considered in the OKI metamodel.

It is important to mention that the presented limitations only focus in the studied OSIDs, which are the OSID necessary to describe high level educational settings. Other limitations of OKI specification could be found when studying the OSIDs from other perspectives or studying other OSID packages.

8 Conclusions and further work

The main contribution of this article is to propose an environment capable of automatically generating web service invocations as part of the implementation of educational scenarios from their specification. However, other contributions derives from this work, such as 1) the creation of an OKI-OSID ontology, which can be downloaded from http://personal.uoc.edu/personalonto/ontologies, and 2) the diction of a set of limitations to the O.K.I. OSID specification.

The proposed environment has as its front-end an OWL+SWRL ontology that describes the semantics of a metamodel of the O.K.I. OSID specification, which has been defined from the study of that specification. As support for the instantiation of the ontology a CASE tool was available which facilitated the graphic representation of the educational scenarios that would populate the ontology. And for the extraction of knowledge an

OWL+SWRL to C# compiler was used to generate web service invocations defined in accordance with the O.K.I. OSID specification from the information stored in the ontology and the web service specifications contained in the C# packages offered by O.K.I. OSID.
Apart from adapting the ontology to eliminate detected limitations, we propose the integration of other standards and specifications to extend the number of OKI interfaces following other authors who had extended the SCORM Content Aggregation Model in order to describe learning objects [11]. For example, it would be interesting to take into account the IEEE-LOM standard [13] for the definition of resources or the IMS-QTI [17] for the definition of assessable learning activities. Another potential further work is the creation of a generic framework that uses different ontologies in the specification of educational scenarios focused to their implementation. That would allow to generate the source code of the educational scenarios using any specification and written in any language.

References

[1] Advanced Distributed Learning, Sharable Content Object Reference Model Version 1.3.1, http://www.adlnet.org. (2004)

[2] A. Gómez-Pérez, M. Fernández-López & O.Corcho, Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web, Springer Verlag (2004)

[3] A. Flahive, D.Taniar, W. Rahayu, B.O. Apduhan, Ontology tailoring in the semantic grid, Computer, Standards & Interfaces, Special Issue: Specification, Standards and Information Management for Distributed Systems, 31(5), pp.870-885 (2009)

[4] A. Inaba, T. Tamura, R. Ohkubo, M. Ikeda, R. Mizoguchi & J. Toyoda, Design and Analysis of Learners' Interaction based on Collaborative Learning Ontology, Proceedings of the 1st European conference on computer-supported collaborative learning (Euro-CSCL'2001) p. 308–315, Maastricht, The Netherlands (2001)

[5] A. Olivé, Conceptual modeling of information systems. Springer-Verlag New York Inc. (2007)

[6] A. Rius, J. Conesa, D. Gañán, A DSL Tool to assist specifications of educational settings. The 2010th International Conference on Semantic Web and Web Services (2010)

[7] A. Rius, Un marco formal para la representación de escenas educativas reutilizables, Internet Interdisciplinary Institute (UOC) (2010)

[8] A. Rius, J.Conesa, E. Garcia-Barriocanal, Metamodelo de la especificación OSID versión 2 de OKI to be published in IN3 Working Paper Series, UOC (2011)

[9] Claroline, http://www.claroline.net/

[10] Dokeos, http://www.dokeos.com/es

[11] E. Jui-Lin L, C. Hsie, A relation metadata extension for SCORM Content Aggregation Model, Computer, Standards & Interfaces, Special Issue: Specification, Standards and Information Management for Distributed Systems, 31(5), pp.1028-1035 (2009)

[12] D.S. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing: Applying MDA to Enterprise Computing, Indianapolis, Indiana: Willey Publishing Inc. (2003)

[13] IEEE Learning Technology Standards Committee [LTSC WG12], Draft for Learning object metadata, http://ltsc.ieee.org/wg12/files/LOM_1484_12_1_v1_Final_Draft.pdf (2002)

[14] ILIAS, http://www.ilias.de/docu/

[15] IMS Global Learning Consortium, IMS Tools Interoperability Guidelines v1.0 Final, http://www.imsglobal.org/ti/index.html (2006)

[16] IMS Global Learning Consortium, Public Draft 2 Content Packaging Specification v.1.2, http://www.imsglobal.org/packaging/ (2007)

[17] IMS Global Learning Consortium, Public Draft 2 IMS Question & Interoperability v2.1, http://www.imsglobal.org/question (2008)

[18] JISC e-Learning Frameworks and Tools, http://www.jisc.ac.uk/whatwedo/programmes/elearningframework.aspx (2004)

[19] LAMS, http://www.lamsinternational.com/

[20] M. Chinosi, A. Trombetta, BPMN: An introduction to the standard, Computer, Standards & Interfaces, In Press, Accepted Manuscript, Available online DOI: 10.1016/j.csi.2011.06.002.

[21] M. Ikeda, U. Hoppe & R. Mizoguchi, Ontological Issues of CSCL Systems Design, Proceedings of the 7th World Conference on Artificial Intelligence in Education pp. 16-19 (1995)

[22] Moodle, http://moodle.org/

[23] N. Guarino, Formal Ontology and Information Systems (pp. 3-15), IOS Press (1998)

[24] O. Pastor, J.C. Molina, Model-driven architecture in practice: a software production environment based on conceptual modeling. Berlin, Heidelberg: Springer Verlag (2007)

[25] OKI, Open Service Interface Definitions. v 2.0.0, http://heanet.dl.sourceforge.net/project/okiproject/Doc%20%28previous%20OSID%20versions%29/Full%20Documentation%20Set/OSID_Documentation_rc6.1.pdf (2004).

[26] OMG & BPMI.org, Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification, dtc/06-02-01 (2006)

[27] Mizoguchi, R., Sinitsa, K., & Ikeda, M.,Task Ontology Design for Intelligent Educational/Training Systems, Positon Paper for ITS 96 Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs ,Vol. 96, pp. 1-21 (1996).

[28] SAKAI, http://www.sakaiproject.org /

[29] S.C. Kabel, B.J. Wielinga, R. de.Hoog, Ontologies for Indexing Technical Manuals for Instruction, AI-ED (Artificial Intelligence in Education) conference, Proceedings workshop on ontologies for intelligent educational systems pp. 44-53 (1999)

[30] SharePointLMS, http://www.sharepointlms.com/