On the use of an IMS LD ontology for creating and executing Units of Learning: Application to the Astronomy case study

Eduardo Sánchez[1], *Manuel Lama*[1], *Ricardo R.Amorim*[2], *Juan Carlos Vidal*[1], *and Adrián Novegil*[1]

[1] Intelligent System Group.
Dept. Electronics and Computer Science
School of Physics
Universidade de Santiago de Compostela
www.gsi.dec.usc.es

[2] UNEB/FACAPE
Universidade do Estado da Bahia
www.uneb.br

Abstract: This paper describes how the Astronomy case study is modelled, created and executed, using a IMS LD ontology. The ontology was aimed at overcoming the expressiveness limitations of the IMS LD XML Schema by means of a taxonomy of concepts and a set of formally defined axioms. With regard to the authoring (creation) stage, the ontology has been used to enable the automatic validation of IMS LD documents in WebLD, a web authoring tool to create IMS LD Units of Learning. With regard to the execution stage, a service-oriented architecture was developed to allow the execution of IMS LD Units of Learning, whose learning processes have been modelled through Petri Nets.

Keywords: IMS Learning design, ontologies, authoring tools, execution engines, workflows, Petri nets.

1 Introduction

The Astronomy case study is focused on the acquisition of knowledge in the field of astronomy. More precisely, the learners are aimed at classifying the planets with respect to their distance from the Sun (from the nearest one to the most distant). The teacher also wants the learners to work together, applying both collaborative and negotiation methods with their peers. The method used by the teacher to reach these objectives consists on proposing a game strategy in which learners are grouped into two teams (Team A and Team B). Resources and services will be available to help the learners in acquiring new knowledge, exchanging information with teammates, and negotiating with the other team. The overall strategy is shown in the activity diagram of figure 1.

For clarity, we have initially described the case study by considering a single *Play* with the following run-script of *Acts* and *Activities*:

- Act I: Cooperative part
    - Activity Structure I.1: Solving planet names and planet order.
        - Learning Activity: evaluate interview.
        - Learning Activity: discuss in forum.
        - Learning Activity: members negotiation.
    - Activity Structure I.2: Verify process
        - Support Activity: add new clue.
        - Learning Activity: discuss in forum.
        - Support Activity: monitoring.
- Act II: Individual part
    - Learning Activity II.1: questionnaire.

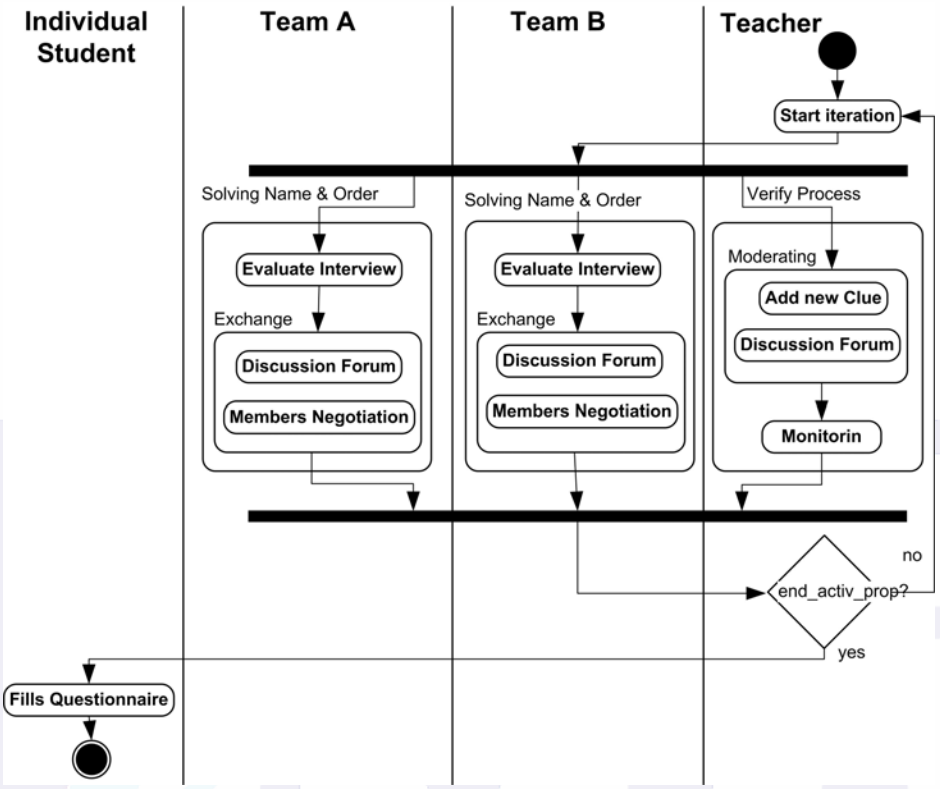

Figure 1. Activity diagram of the case study

The two Acts are performed serially, while the Activity Structures of Act I are performed concurrently. The duration of Act I, and therefore the beginning of Act II, depends on the Teacher´s decision. Team A and Team B are Roles in charge of performing the Activity Structure I.1, Teacher is the Role associated to Activity Structure I.2, and Student is the role responsible to cover the Learning Activity II.1. Notice that for brevity we have not considered an initial Act where the objectives are explained and clues are distributed among the teams.

The paper presents how the Astronomy case study can be modelled, created and executed using an IMS LD ontology. Section 2 overviews the ontology, briefly describing the concept taxonomy and some relevant knowledge axioms. Section 3 explains how the ontology is used in the authoring (creation) stage. Section 4 outlines a service-oriented architecture to execute IMS LD documents. Finally, section 5 provides a discussion on the benefits of using ontologies and the future challenges that need to be addressed.

2 IMS LD Ontology: An ontology to model Units of Learning

The IMS Learning Design specification (IMS LD), IMS (2003), is an Educational Modelling Language (EML) that provides a model of semantic notation to describe both the content and processes of units of study (Koper, 2001). This specification, drawn up by the IMS/LDWG work group, is an integration of the EML developed by the OUNL (Open University of Netherlands), with other existing IMS specifications for the exchange and interoperability of e-learning material. The OUNL EML is a meta-vocabulary that is defined based on the diversity of concepts existing in a wide range of pedagogic techniques. The IMS LD incorporates the OUNL EML, and describes the structure and educational processes based on a pedagogic metamodel. IMS LD represents Units of Learning by means of a method that is made up of a number of activities carried out by both learner and staff in order to achieve some learning objectives. It allows the combination of various techniques (traditional, collaborative, etc.), and facilitates the description of new ones. From the proposed specifications, the

IMS LD has emerged as the de facto standard for the representation of any learning design that can be based on a wide range of pedagogical techniques.

The IMS LD information model specifies three levels of modelling: (1) level A, containing the core concepts needed to model any pedagogical situation; (2) level B, extending the level A with Properties and Conditions, aimed at supporting more sophisticated behaviours enabling personalization and learner interactions, and (3) level C, adding Notifications to the previous levels. Furthermore, IMS LD is intended to support the execution of IMS LD documents, IMS (2003). To accomplish this, the architecture of an IMS LD system should support the modelling and creation of IMS LD documents (Authoring), validation, publication, and population of IMS LD documents (Production), and personalization and role population issues (Delivery).

2.1 Use of Ontologies

Most of the tasks involving the authoring, production and delivery of IMS LD documents could be automated by using software agents working on behalf of authors, publishers, and learners. In order to fully use the capabilities of those agents, all the knowledge regarding with the learning design process is required to be described in a formal and explicit way. Ontologies (Gómez-Pérez, Fernández-López and Corcho, 2004) come handy to describe formally and explicitly the structure and meaning of the metadata elements; that is, an ontology would semantically describe the metadata concepts. In the educational domain, several ontologies have been proposed: (1) to describe the learning contents of technical documents (Kabel, Wielinga and de How, 1999); (2) to model the elements required for the design, analysis, and evaluation of the interaction between learners in computer supported cooperative learning (Inaba, Tamura, Ohkubo, Ikeda, Mizoguchi and Toyoda, 2001); (3) to specify the knowledge needed to define new collaborative learning scenarios (Barros, Verdejo, Read and Mizoguchi, 2002); or (4) to formalize the semantics of learning objects that are based on metadata standards (like LOM) (Brase and Nejdl, 2004). The focus of that research is either on the development of a taxonomy of concepts on the basis of an established theory or specification (1 to 3), or on the formal definition of the metadata using an ontology language (4). However, none of them deal with the formal description of the meaning of the concepts, and they do not address the ontological modelling of any specification for learning design.

Ontologies are then required (1) to formally describe the semantics implicit in the information model, and (2) to enable software agents to perform complex tasks. Some examples could be:

1. Authoring Stage. A set of axioms could be used by authoring agents to guide the creation of IMS LD documents, as well as to validate their logical consistency.
2. Production Stage. The ontology could be used by production agents to automate different tasks such as: resource allocation based on availability, creation of new IMS LD runs based on the number of registered students, role assignments based on student profiles, and so on.
3. Delivery Stage. The ontology could also be used to solve the problem of personalization or adaptation of the learning design to the individual users. It involves different tasks such as the selection of suitable learning activities, contents, etc.

2.2 IMS LD Ontology: taxonomy and axioms

To develop the Learning Design ontology we have created a concept taxonomy (figure 2), which describes the elements of the IMS LD conceptual model and the IMS LD information model, and a set of axioms (Table 1), which formally constraints the semantics of the concept taxonomy on the basis of the explanations formulated in natural language in both information and behavioural models (Amorim, Lama, Sánchez, Riera and Vila, 2006).

The concepts were obtained from elements and relations explicitly represented in the IMS LD information model document. A set of axioms was additionally included to guarantee the consistency of the definitions and to represent implicit knowledge. The axioms were obtained from restrictions

identified after an extensive analysis of the explanations, written in natural language, provided by the IMS LD Information Model and the Best Practice and Implementation Guide documents. In Table 1 we present some relevant examples.

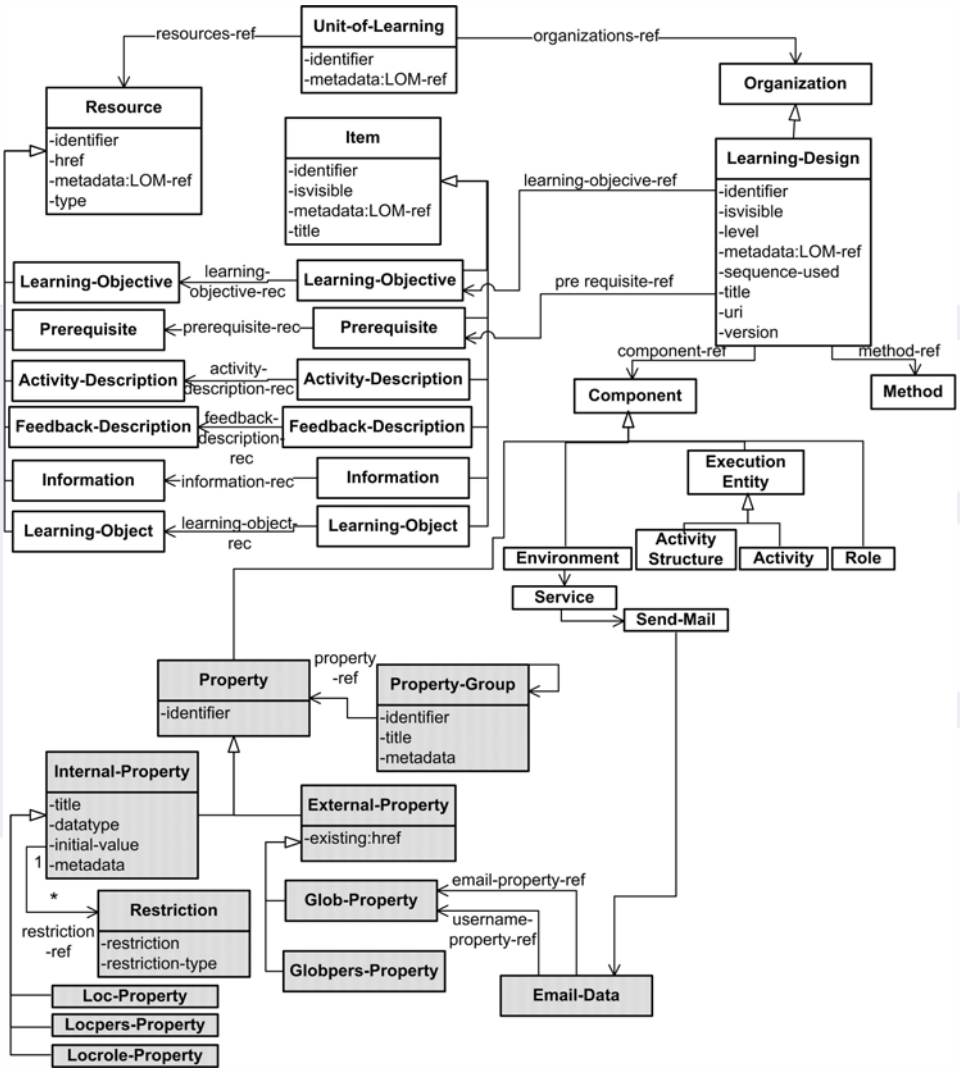

Figure 2: IMS LD concept taxonomy with level A (white) and level B (grey) concepts

In figure 2, the Unit of Learning concept (UoL) integrates the description of the Resources and the Learning Design (LD). The Resource concept allows representing various entities, like physical resources, and concepts whose attribute description is domain-dependent. The Learning Design concept is related to the Learning Objective (defining the intended outcomes) and the Prerequisite (describing required previous knowledge) concepts. These concepts are subclasses of Item and they can be mapped onto the equivalent concepts in the Resource hierarchy. The Components (Role, Execution Entity and Environment) of the Learning Design represent pieces of the educational process. Role instances participate in an Execution Entity in the context of an Environment.

The IMS LD level B adds the Property concept to the level A to extend the description of the elements of level A. Properties represent data that can be stored enabling to maintain information about Role, Environment and the state of Execution Entities such as Play, Act, Activities. The properties constitute an essential part of monitoring, personalization, assessment, and user-interaction processes. Level B also adds the Conditions concept (not shown), which is aggregated into the Method concept and it is used along with properties to facilitate the learning design refinement. For example, in a Method,

1	IMS LD Specification	*Inf-Mod, Page 83 (item 17)*: "Environments are connected to activities, activity-structures or roles (in a role-part). When an activity-description is visible, always the connected environment (including the content structure of the environment) must be made visible. It must be possible to access and see the activity-description and the content of one of the objects or services within the environment at the same time."
	Explanation	When the value of the `isvisible` attribute of an `Activity Description` is "true", the value of that attribute for the `Environments` connected to the `Activity` associated to the `Activity Description` must be also true.
	Formal Description	∀ a, ad, e, lo, s | a ∈ Activity ∧ ad ∈ Activity-Description ∧ activity-description-ref(ad, a) ∧ e ∈ Environment ∧ lo ∈ Learning-Object ∧ s ∈ Service ∧ learning-object-ref(lo, e) ∧ service-ref(s, e) environment-ref(e, a) ∧ isvisible(ad) = "true" → isvisible(lo) = "true" ∧ isvisible(s) = "true"
2	IMS LD Specification	*Inf-Mod, Page 40 (item 0.5.1)*: "This element states that a play is completed when the last act is completed."
	Explanation	The `Act` referred as the value of the attribute `when-last-act-completed` of a `Play` must be one of the `Acts` associated to the `Play`.
	Formal Description	∀ p, cp, a | p ∈ Play ∧ cp ∈ Complete-Play ∧ complete-play-ref(cp, p) ∧ a ∈ Act ∧ when-last-act-completed(a, cp) → act-ref(a, p)
3	IMS LD Specification	*Inf-Mod, Page 51*: "Property values may be calculated from the values of other properties. It is also possible to take over the property value of another property (with property-ref)."
	Explanation	In a Change-Property-Value, if the related property value was obtained from the value of other property, both, the datatype of the property to be changed and the datatype of the other property must be the same.
	Formal Description	∀ cpv, p, p1, pv | cpv ∈ Change-Property-Value ∧ p, p1 ∈ Property ∧ pv ∈ Property-Value ∧ property-ref(p, cpv) ∧ property-value(pv, cpv) ∧ property-ref(p1, pv) → datatype(p) = datatype(p1)

Table 1: Example of axioms: Level A (1 and 2), and Level B axioms (3)

Conditions can be introduced either to show or hide entities such as Play, Activity and Environment in a personalized way. The Conditions concept has a typical programming language structure: IF [expression] THEN/ELSE [show/hide something or change property-value].

2.3 Modelling the Astronomy case study

Figure 3 shows how the instances of the Astronomy scenario are represented using our IMS LD ontology, which models both IMS LD Level A and Level B concepts. A Unit of Learning instance, called *Astronomy*, is introduced to associate *interview-rec* and *questionnaire-rec*, which are Resource instances of type Learning Objects, with the *Learning Astronomy Game* Method, which describes the learning run-script. For each Act, different Role Parts have been defined in order to associate Roles with Activity Structures and Learning Activities. For each Learning Activity, an Environment is introduced. Services like e*xch-inf-forum* and *exch-inf-chat* are naturally associated to those environments. Furthermore, Level B Conditions and Properties are required to model the end of the *Cooperative part* Act. When the Teacher decides to finalize it, s/he simply has to stop the *verify-process* activity by setting the value *monitoring-oc*, of type On Completion, to true. In order to hide the visible activities of Act I, this action also changes the value of the Property *End-Activ-Prop* to true. When it happens, the Expression *End-Activ-Expr* is evaluated, the Condition *End-Activ-Cond* is then satisfied, and the ThenElseModel *End-Activ-TEM* is carried out. Two Hide actions are then fired: *Mod-hide*, which hides the *verify-process* Activity Structure, and *solv-plan-hide*, which hides the *solving-planet-name-order* Activity Structure.

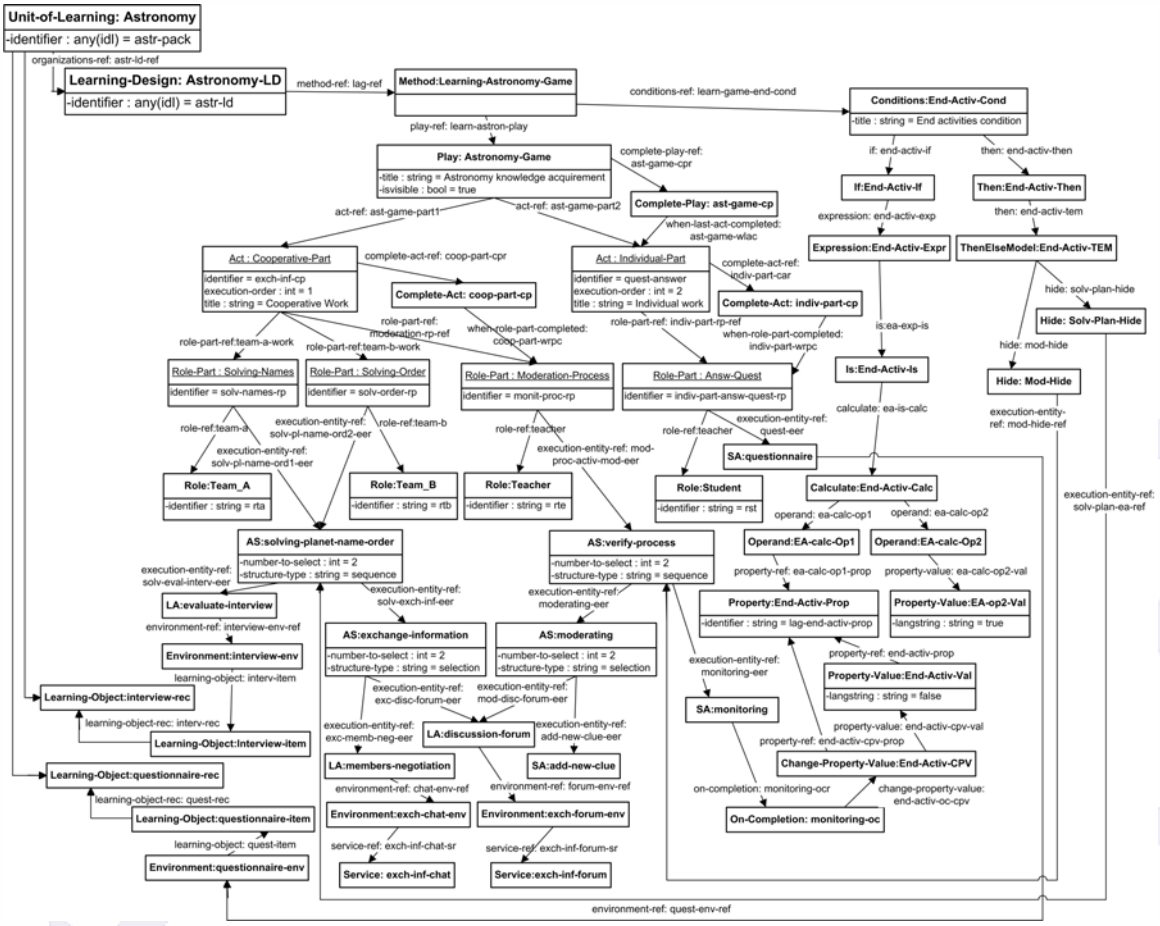

Figure 3: Instances of the Astronomy Case Study using the IMS LD ontology

3 WebLD: Ontology-based authoring tool to create Units of Learning

There exist two main authoring tools to create IMS LD Units of Learning (UoLs): Reload LD Editor and CooperAuthor. Reload LD Editor was developed at the University of Bolton (Reload, 2007) and offers a graphical environment used for editing and composing packages in conformity with the IMS LD specification. For each UoL, the core IMS LD concepts can be created and edited. The authoring process consists on filling in all the form fields, according to the IMS LD specification. On the other hand, CopperAuthor was developed by the OUNL (CooperAuthor, 2007). It provides a graphical interface that gives the user the possibility of developing and validating units of learning, visualizing the resulting XML code and completing units of learning. CopperAuthor also allows to assign roles with runs in CopperCore and a previsualisation with CopperCore for executing the created course.

Some limitations can be found on these tools. Because of their nature as desktop applications, they are not accesible “anywhere”, and they require software instalation on every workplace. Moreover, there is no rigurous semantic validation, and it is then possible to create semantically inconsistent UoLs.

To overcome these limitations, we have developed WebLD, a web authoring tool aimed at creating learning designs and validating them with the IMS LD ontology. In the following sections we discuss the application requirements, the design, the implementation details, and an authoring example to show the graphical interface.

3.1 WebLD Requirements

In what follows we introduce the main requirements of the application:

- Functional requirements
  - FR1. Creation, list and edition of UoLs.
  - FR2. Creation, list and edition of Activities.
  - FR3. Creation, list and edition of Roles.
  - FR4. Creation, list and edition of Environments.
  - FR5. Creation, list and edition of Learning Objects and Services.
  - FR6. Creation, list and edition of Plays.
  - FR7. Creation, list and edition of Acts.
  - FR8. Creation, list and edition of Role-Parts.
  - FR9. Export and import of UoLs in IMS LD format.
  - FR10. Semantic validation of UoLs.
- Level-of-service requirements
  - LoS1. Anytime/anywhere accessibility.
  - LoS2. Easy-to-use navigation in order to introduce and edit the UOLs components.
  - LoS3. Interoperable with other IMS LD level A tools.
  - LoS4. Extensible to new functionalities supporting IMS LD level B and C.

3.2 Design

The design mainly comprises a software architecture, describing the static components of the application, and sequence diagrams, which show the behaviour of the architecture in response to user requests.

3.2.1 Software architecture

Figure 4 shows the software architecture of the application, which consists of components, connectors, as well as a topology determining the arrangement of components and connectors. Some design patterns have been used: Model-View-Controller (MVC), InterceptingFilter, FrontController, TransferObject, and Data Access Object. MVC is used as a common pattern to implement the presentation layer, the filters are used to validate and pre-processed every client request, FrontController determines a unique access point for every user action, and the other two patterns manage both data transfer and data access. The main types of component are listed below:

- *Views (V)*: Generate the user interface and present data to users.
- *Filters*: Validate and preprocess all client requests before reaching the Controller.
- *Controller*: Manages the client requests and decides which action will follow on each case. It also handles how the Forms are passed through the different Actions and Views components.
- *Forms (F)*: Encapsulate both actions carried out by users on the browser as well as data introduced on each request. These components are like value objects that package data that will be used for Actions as well as Views components.

- *Actions (A)*: Perform the business logic. They are triggered by the controller, which delegates the control to them in order to retrieve and process data from the database. These components are connected with Data Access Objects to access data from the database.
- *Data Access Objects (DAOs)*: Manage the fine grain access to the data stored in the database.
- *Value Objects (VOs)*: Encapsulate data that can be transferred througout the application.
- *Database manager*: Manage the connection with the database.
- *Utilities*: Provide additional functionality to Views, Actions and DAOs.
- *Libraries*: Third-party libraries to support specific actions.

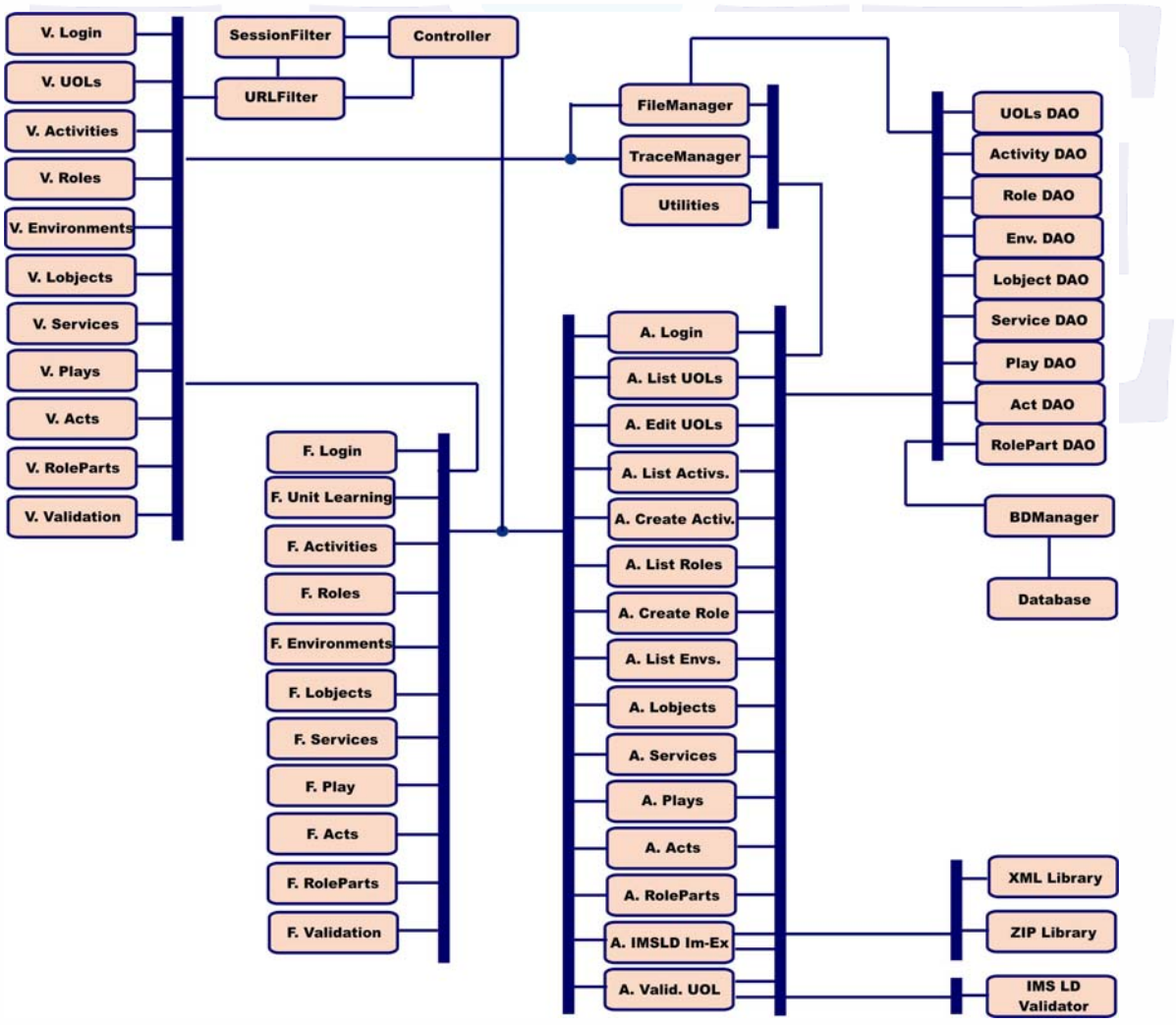

Figure 4: Software architecture of WebLD

3.2.2 Sequence diagrams

The sequence diagram of figure 5 shows the steps carried out to create new activities and new roles. The flow is initiated from the view component V.UoL that requests the presentation of the list of activities for that unit. After the list is presented in the view component V. List Activities, a user can request the creation of a new activity. The request is filtered by the SessionFilter component and then forwarded to the Controller. It dispatches the control to the action component A. Create Activity, which then invokes the ActivityDAO to access the database in order to create the activity. A similar procedure, starting from the view component V. List Roles, is followed to create new roles.

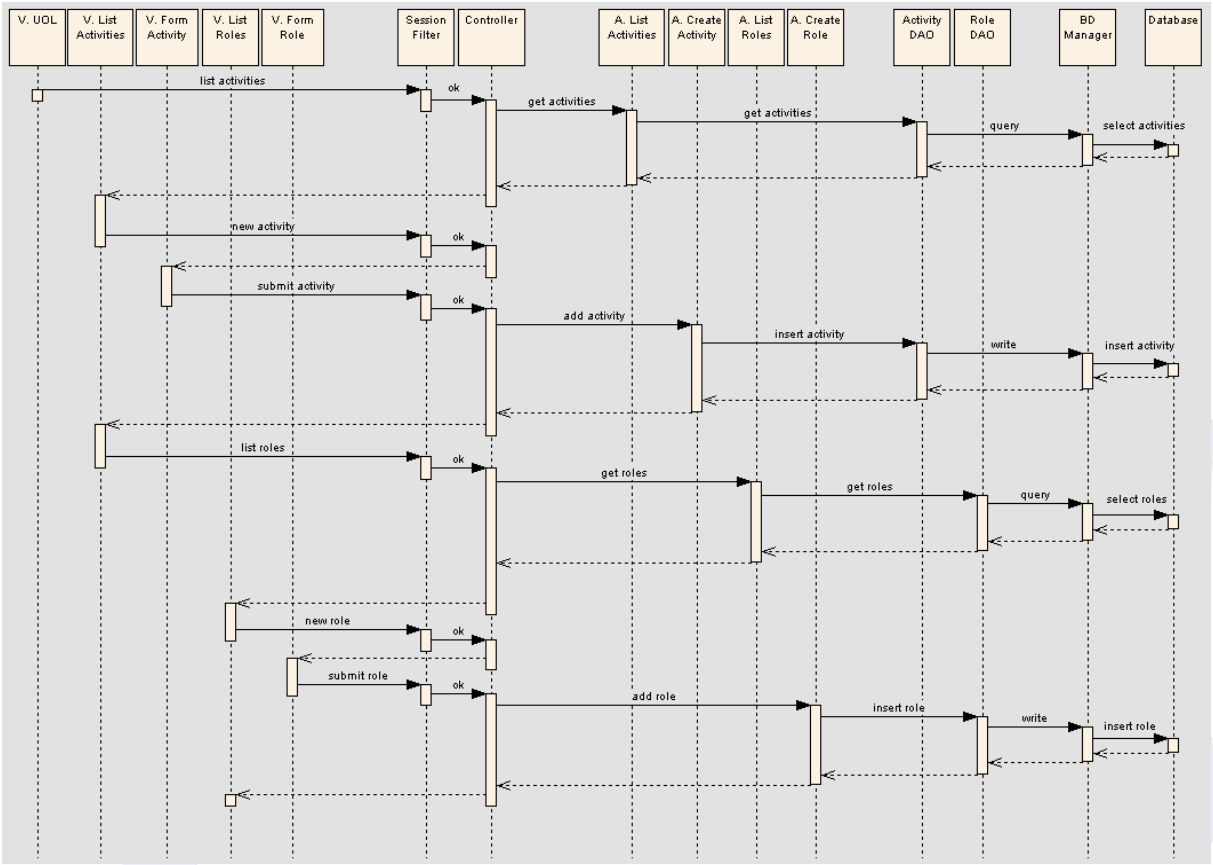

Figure 5: Sequence diagram to create new activities and roles in WebLD

3.3 Implementation

The following technologies have been used to implement the application:

- The J2EE platform has been used as the main development technology.
- Struts was chosen as the framework to implement the web/presentation tier, which means the views, forms, actions and the controller component.
- Xerces2 Java Parser 2.9.0, an XML Schema processor, to implement the IMS LD export/import component.
- MySQL Server 5.0 as the database manager.
- Jboss 4.0.5 was chosen as the application server. In this way, we allow the future implementation of the business logic with EJB 3.0 components.
- Java and F-Logic to implement the validation component. The key element in this component is the IMS LD ontology that provides axioms and rules to guarantee the semantic coherence of each UoL document (Amorim et al., 2006).

3.4 Creating the Astronomy Unit of Learning

A typical WebLD session starts with the creation of a new Unit of Learning. For each UoL, an IMS LD Method is automatically generated. After that, WebLD provides the user with three different strategies to build the design:

- Bottom-up. Under this strategy the UoL components (activities, roles and environments) are created in the first stage, whereas the UoL method (Plays, Acts and Role-parts) is composed of the previously defined components
- Top-down. Under this approach, the UoL method is initially defined and the UoL components are created afterwards.
- Hybrid. It combines the bottom-up and top-down approaches, facilitating the user to switch between both component creation and method definition.

As for the Astronomy UoL we have chosen the first strategy. Figure 6 shows the edition of the Evaluate-interview learning activity. Moreover, it is possible to associate and edit an environment. In this case, the Interview-env environment was chosen.

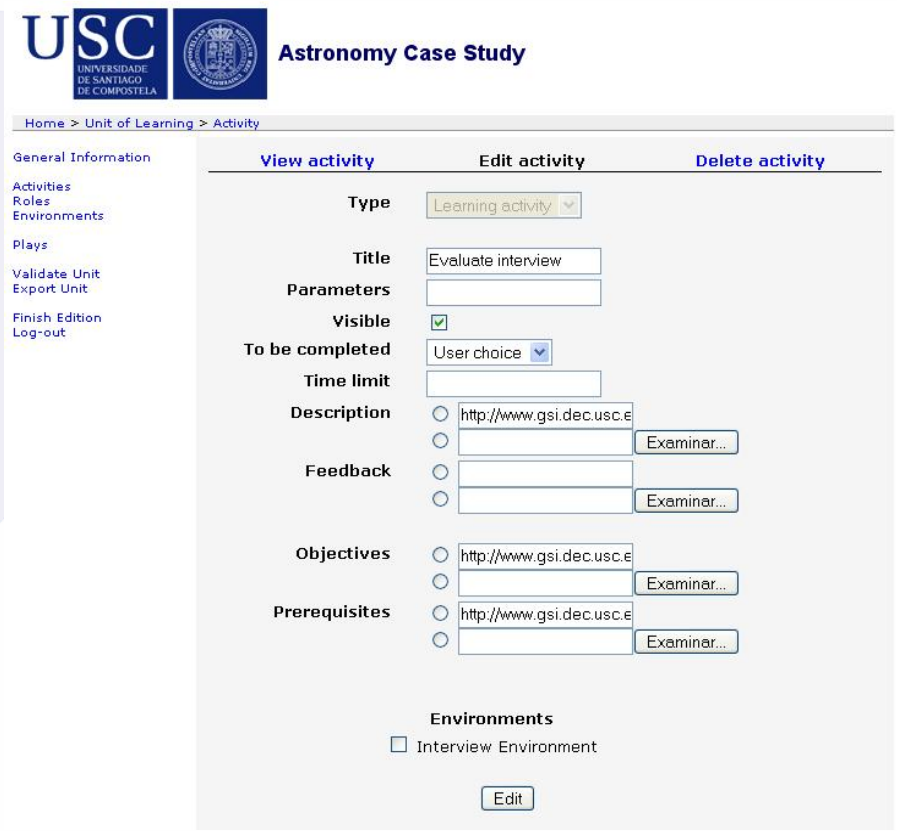

Figure 6: WebLD user interface: form page to create/edit a new activity

The *Interview-env* environment can be characterized by selecting related learning objects and services. For this specific environment, only the *Interview-rec* learning object was added (figure 7). For other environments, like the *exch-chat-env* Environment, associated to the *exchange-information* Activity Structure, the chat/conference service should be created.

Once the activities, environments and roles are created, we can move on and create the method components: Plays, Acts and Role-parts. Figure 8 shows that the *Solving-names* Role-part is defined by associating the *Solving-planet-names* Activity Structure and the *Team A* Role.

However, the most relevant feature of WebLD is the semantic validation of UoLs. The IMS LD elements of the UoL are automatically validated against the axioms of the IMS LD ontology (section 2.2). Figure 9 shows a typical result of the validation process.

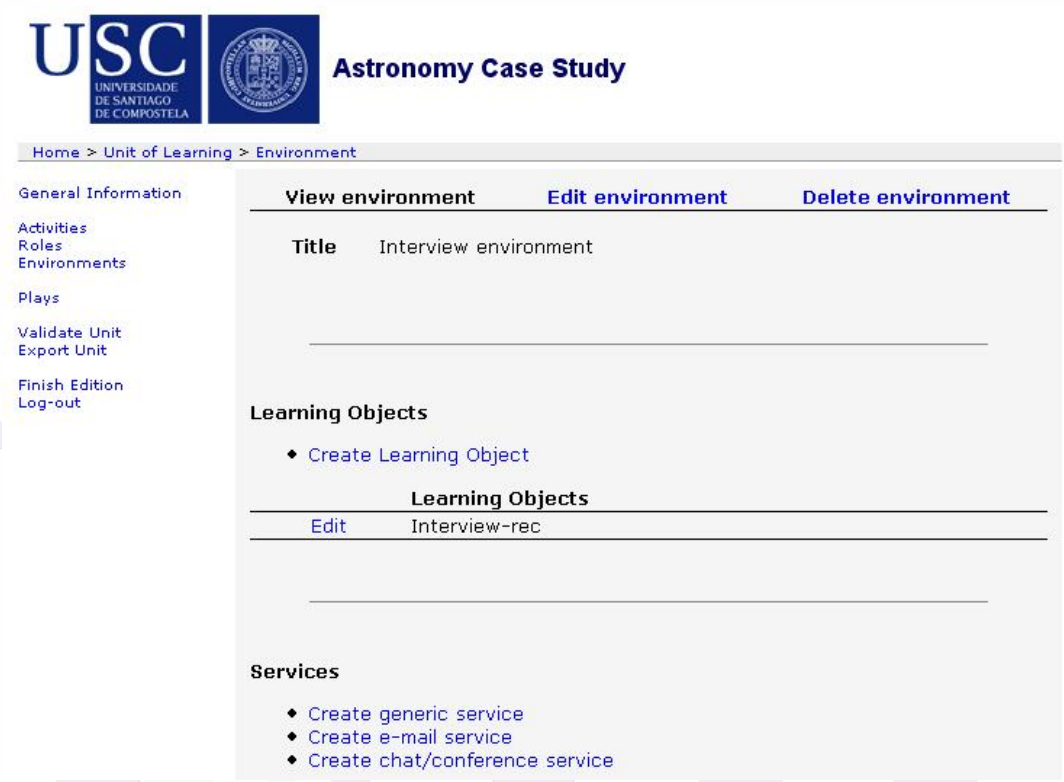

Figure 7: WebLD user interface: environment page showing related learning objects and services

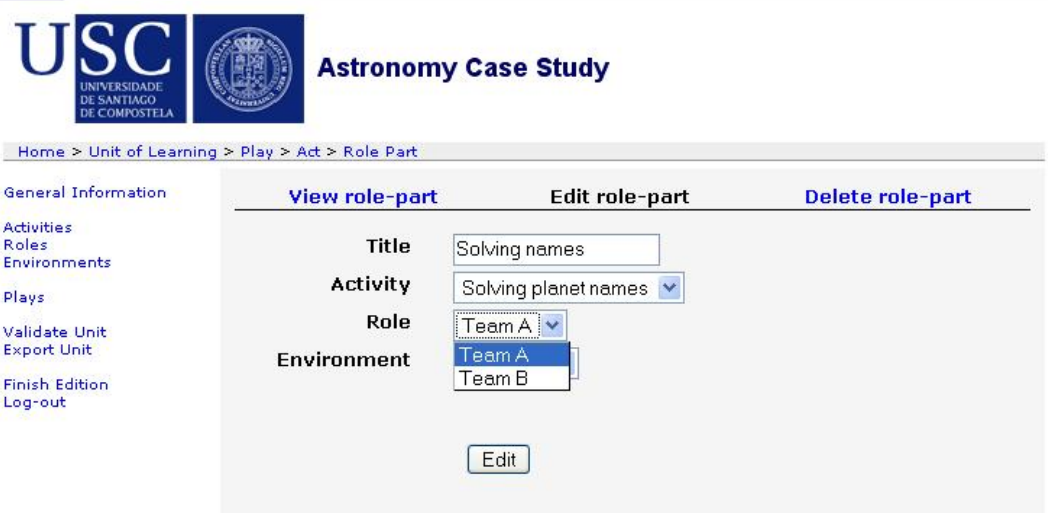

Figure 8: WebLD user interface: form page to create a new role-part

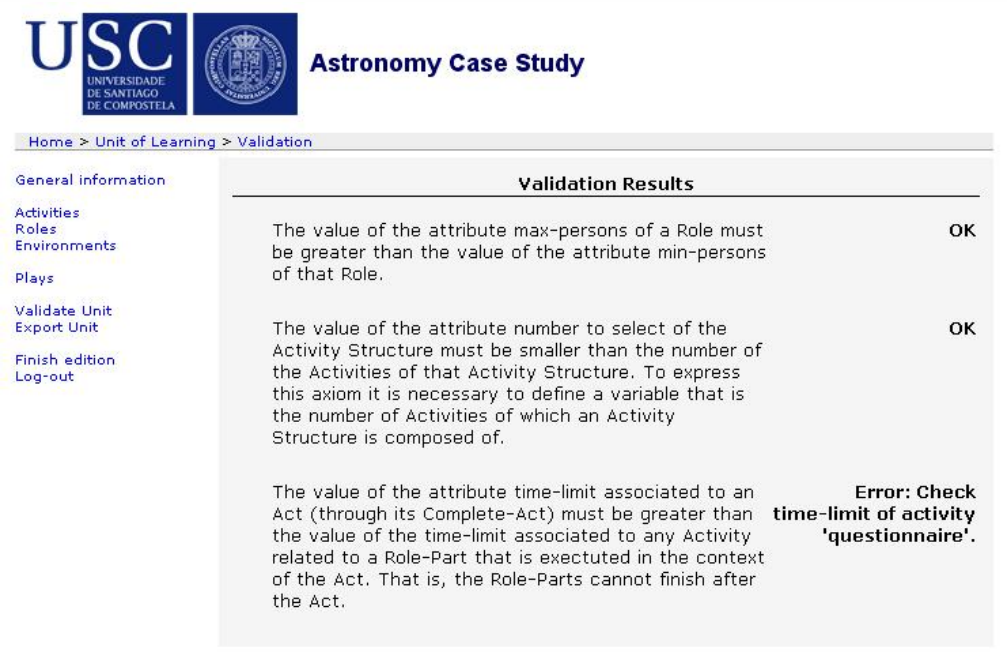

Figure 9: Validation of IMS LD Units of Learning

4 Service-oriented architecture to execute UoLs

Nowadays, the CopperCore engine (Vogten and Martens, 2005) is the unique platform available for executing IMS LD learning designs. This engine has two main drawbacks concerning with its reuse and maintenance: (1) the design does not facilitate its integration with other e-learning applications; and, (2) modifications and/or evolutions of the IMS LD would force the current engine, as well as the e-learning application in which it could be integrated, to be re-designed and re-implemented.

To solve these problems we have developed an execution engine based on the IMS LD ontology, workflows/Petri Nets, and web service technologies that allow developers to design and execute units of learning following the IMS LD specification. In what follows we describe how the Units of Learning are modelled and then executed on a service-oriented architecture.

4.1 Petri Nets to model learning processes of Units of Learning

The execution of a Unit of Learning can be seen as the coordination of a set of activities with a set of participants. By considering these activities as processes, we can cast the execution of UoLs as the execution of workflows. Therefore, the execution of an unit of learning is approached as a workflow modeling problem in which the activities represent the tasks to be performed, whereas the methods, plays and acts, impose constraints on the workflow structure.

Workflow modeling techniques are various and heterogeneous: process algebra's (Milner, 1999), Petri nets (Van der Aalst, 1998), and vendor specific diagrams are the most representative solutions. In the absence of a standard language, a solid theoretical foundation with a graphical semantics would pave the way to facilitate the definition of these processes. On this regard, we have resorted to high-level Petri nets (ISO/IEC 15909-1, 2002) to design the workflows that are implicitly defined on every Unit of Learning. Petri nets provide: (i) a formalism with a graphical representation, (ii) an explicit representation of states and events of processes, and (iii) a set of algorithms and tools that facilitate their analysis and verification.

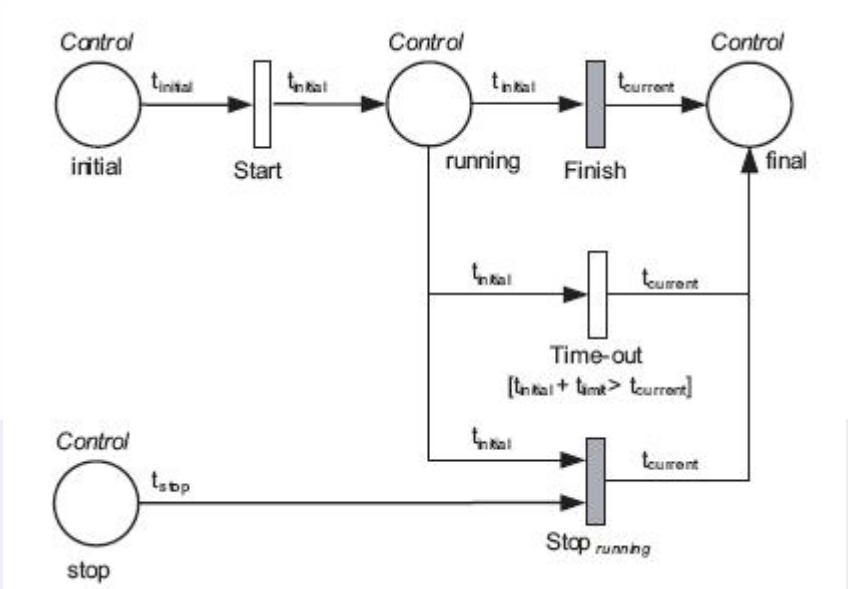

Figure 10: Petri net structure that models the execution of methods, plays, acts, role-parts, activity-structures and activities

The Petri net depicted in figure 10 is used to model the execution of any method, play, act, role-part, activity-structure, or activity. This common structure unifies the way in which any of these elements is executed and stopped. Therefore all these elements will move through the same states. This net has two input interfaces, the initial and stop places, one output interface, the final place, and has two distinguishable parts:

- The upper part performs the execution of a method, play, etc. in two steps: firstly with the firing of the Start transition which sets the time stamp token in which the task has been started; and secondly with the firing of the Finish transition which will create a token in the final place indicating that the method, play, etc. has ended.
- The lower part models the two ways in which a method, play, etc. can be stopped. The first one stops the execution if the period of activity of the task has exceeded the timeout. In this case, the Time-out transition is fired and produces a new token in the final place. The second one, stops the execution when a token is located in the Stoprunning place. This situation usually happens when the user decides to stop the execution.

The complete life-cycle of Units of Learning is modelled with more complex Petri Nets. A detailed description of this work can be found in Vidal, Lama, Bugarín and Sánchez (2008).

4.2 Web services to execute Units of Learning

An execution engine to execute Units of Learning following the design principles of service-oriented architectures was developed. The execution involves the following operations over UoLs: (1) introduction of UoLs either as IMS LD XML Schema documents or instances of the Learning Design Ontology, (2) translation into the Petri nets ontology, and (3) management and execution on the Service-Oriented Architecture. All these operations are carried out in three architectural layers (figure 11):

- The first layer contains an LMS that both students and teachers may use to access to courses and therefore to their content, design and edition. WebLD can be used in that layer to create UoLs, and then invoke the execution engine deployed in the second layer. In addition, Moodle (Rice, 2008), which is a free distribution course management system (CMS), can also be used

as LMS. Moodle provides a PHP-based web services invocation layer, which would be in charge of the dialogue with the execution engine.

- The second layer contains the core of the execution engine: OpenESB (OpenESB, 2008). OpenESB, or Open Enterprise Service Bus, is an implementation of an Enterprise Service Bus-based JBI (Ten-Hove and Walker, 2005) developed by Sun Microsystems. This bus facilitates an easy integration of Web services and enabling the creation of loosely coupled applications.
- The third layer represents the set of Web services (Hansen, 2007) that implement the business logic of our application. In this case, the implementation of services is based on the Web Services Stack developed by Sun Microsystems and named Metro (Metro, 2008). This layer contains the reasoner FLORA-2 (Flora-2, 2008) which is an advanced object-oriented knowledge-based language and application development environment. The language of FLORA-2 is a dialect of F-logic with numerous extensions, including meta-programming in the style of HiLog and logical updates in the style of Transaction Logic. This reasoner provides the means for reasoning with ontologies and supports the validation and reasoning on Units of Learning represented as Petri nets.

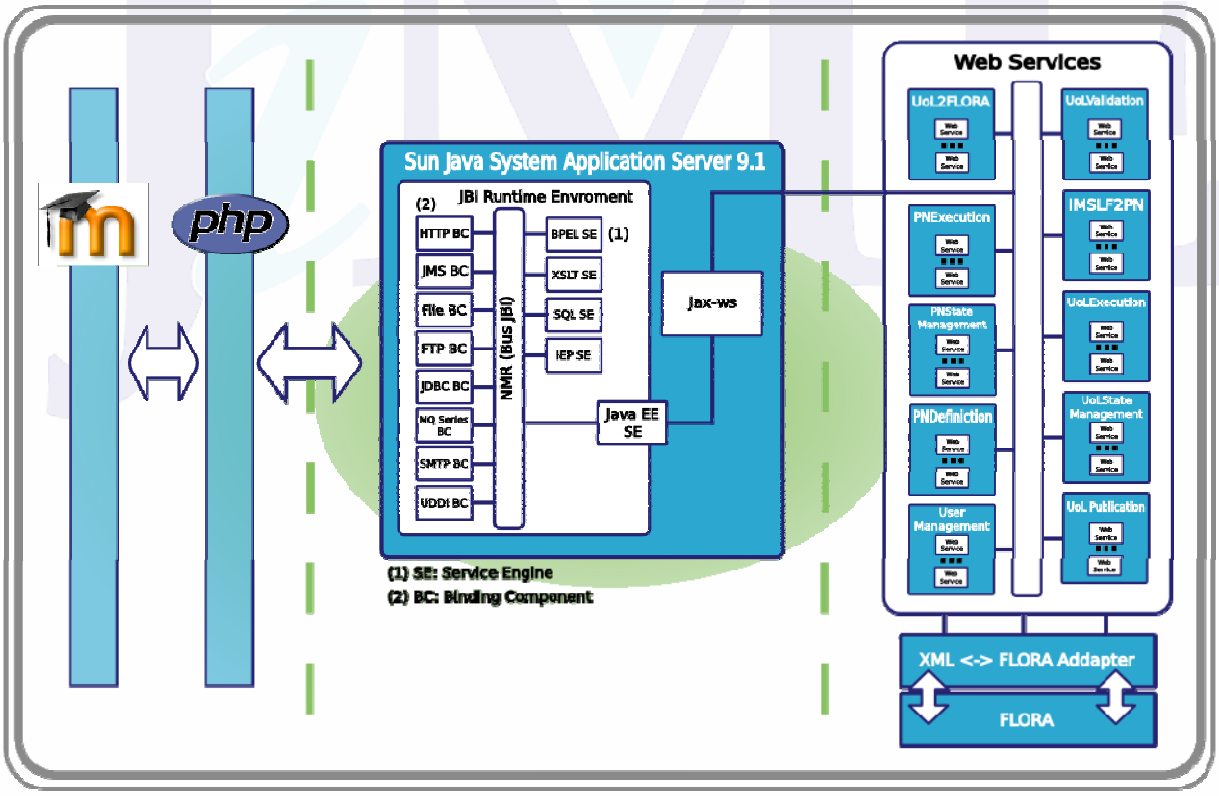

Figure 11: Multi-layer service-oriented architecture to execute UoLs. On the left side, an LMS is drawn to play the role of a client. In the middle, an application server powered with an OpenESB Runtime environment, is used to coordinate the execution of the learning process. On the right side, a set of web services that are used to translate UoLs into the IMS LD ontology, represent the learning processes as Petri Nets, and finally execute them as a workflow

4.3 Executing the Astronomy Unit of Learning

The execution of the Astronomy Unit of Learning starts after the creation of the UoL. By using WebLD, the UoL is represented in terms of the IMS LD ontology, therefore we can move on to translate the learning processes into Petri nets. Figure 12 illustrates the Petri net representation of the *exchange-information* Activity Structure (AS). It is composed of two Learning Activitities (LA), *members-negotiation* and *discussion-forum*, which are modelled on the basis of the common Petri Net

structure of figure 10. The temporal sequence of the Learning Activities is governed by the AND-JOIN transition, which takes care that both ones are finished before stopping the Activity Structure. The translation of learning processes into Petri nets as well as the execution of those Petri Nets, can be both carried out by invoking the corresponding web services deployed in the third layer of the Service-Oriented Architecture.

The execution of the complete Astronomy UoL is finally managed by the OpenESB deployed in the second layer of the Multi-layer service-oriented architecture (figure 11). Within this architecture, each UoL is considered as a workflow and executed by the OpenESB, and each activity is represented as a Petri Net and implemented as a web service.

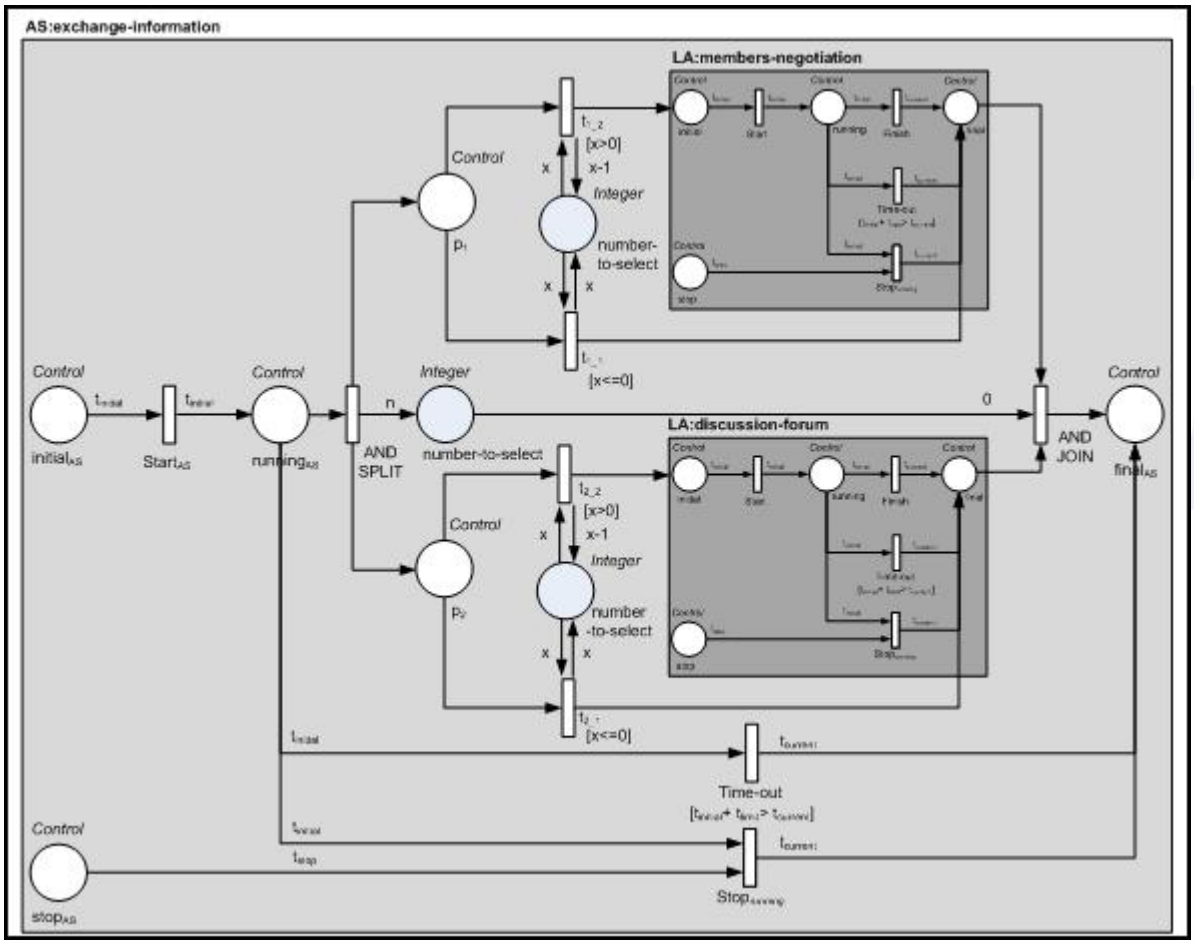

Figure 12: Petri net structure that models the execution of the *exchange-information* Activity Structure

5 Discussion

This work is part of a research line that is focused on the authoring and execution of Units of Learning based on our IMS LD ontology. Regarding with the proposed authoring tool, a further step is needed to improve the usability of the graphical interface. The IMS Learning Design poses a paradigm shift that focuses on design rather than contents (lectures and lab exercises). Therefore, a learning design metaphor is needed to facilitate the transition between the current and the new design practices. Graphical approaches, like MOT+LD (Paquette, Leonard, Lundgren-Cayrol, Mihaila and Gareau, 2006), or simplified versions of the learning design concept, like the LAMS system (Dalziel, 2006), which focuses on sequences of different types of activities, should pave the way to solve this problem and facilitate the design task to learning designers. As for the service-oriented execution engine, the

future work will be focused on the study of the formal properties of the proposed Petri nets models as well as the support for IMS Learning Design level B.

Acknowledgements: Authors would like to thank the financial support in carrying out this work from the Xunta de Galicia and the Ministerio de Industria, Turismo y Comercio under the projects PGIDT06SIN20601PR and TSI-020301-2008-9, respectively.

6 References

Amorim R., Lama M., Sánchez E., Riera A. and Vila X. (2006) A learning design ontology based on the IMS specification. *Journal of Educational Technology and Society*, 38-57.

Barros, B., Verdejo, F., Read, T., & Mizoguchi, R. (2002). Applications of a Collaborative Learning Ontology. *Proceedings of the Second Mexican International Conference on Artificial Intelligence (MICAI 2002)*, Yucatan, Mexico, 301-310.

Brase, J., & Nejdl, W. (2004). *Ontologies and Metadata for eLearning*. In Staab, S. & Studer, R. (Eds.) Handbook on Ontologies, Berlin: Springer-Verlag, 555-574.

CopperAuthor. CopperAuthor Project. (2007). Accessed online on May, 2007 at: http://www.copperauthor.org

Dalziel J.R. (2006) Lessons from LAMS for IMS Learning Design. *Sixth International Conference on Advanced Learning Technologies*, Kerkrade (Netherlands), IEEE Press, 1101 – 1102.

Flora-2 website (2008). Accessed online on April, 2008 at: http://flora.sourceforge.net.

Gómez-Pérez, A., Fernández-López, M., & Corcho, O. (2004). *Ontological Engineering*, Springer Verlag: Berlin.

Hansen M. D. (2007). *SOA Using Java(TM) Web Services*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

IMS Global Learning Consortium (2003). IMS Learning Design Information Model. Version 1.0 Final Specification. Accessed online on 10 April, 2006 at: http://www.imsglobal.org/learningdesign/ldv1p0/imsld_infov1p0.html

Inaba, A., Tamura, T., Ohkubo, R., Ikeda, M., Mizoguchi, R., & Toyoda, J. (2001). Design and Analysis of Learners Interaction based on Collaborative Learning Ontology. *In Dillenbourg, P., Eurelings, A., & Hakkarainen, K. (Eds.)* Proceedings of the Second European Conference on Computer-Supported Collaborative Learning (Euro-CSCL'2001), Maastricht, 308-315.

ISO/IEC 15909-1. (2002) High-Level Petri Nets - Concepts, Definitions and Graphical Notation.

Kabel, S., Wielinga., B., & de How, R. (1999). Ontologies for indexing Technical Manuals for Instruction. *Proceedings of the AIED-Workshop on Ontologies for Intelligent Educational Systems*, LeMans, France, 44-53.

Koper, R. (2001). Modelling units of study from a pedagogical perspective the pedagogical meta-model behind EML. Accesed online on May 26, 2006 at: http://dspace.learningnetworks.org/retrieve/33/ped-metamodel.pdf

Metro website. (2008). Accessed online on April, 2008 at: https://metro.dev.java.net.

Milner, R. (1999) *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, Cambridge, UK.

OpenESB website (2008). Accessed online on April, 2008 at: https://open-esb.dev.java.net.

Paquette G., Leonard M., Lundgren-Cayrol K., Mihaila S., and Gareau D. (2006). Learning design based on graphical knowledge-modeling. *Journal of Educational Technology and Society*, 97 – 112.

Reload. RELOAD Project. (2007). Accessed online on May, 2007 at: http://www.reload.ac.uk

Rice, W. (2008). *Moodle 1.9 E-Learning Course Development*. PACKT publishing.

Ten-Hove R. and Walker P. (2005). Java Business Integration (JBI) 1.0. Sun Microsystems, Inc. Accessed online on April, 2008 at: http://jcp.org/en/jsr/detail?id=208

Van der Aalst, W.M.P. (1998) The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 21{66).

Vidal J.C, Lama M., Bugarín A. and Sánchez, E. (2008). The Application of Petri Nets to the Execution of IMS Learning Design. To be published in *Proceedings EC-TEL 2008*.

Vogten H., and Martens H. (2005). CopperCore 2.2.2. Accessed online on April, 2008 at: http://www.coppercore.org