# Implementing the CROP Reference Architecture: The CROP Learning Object Editor

Maria Tsiakmaki
PHD Student
Department of Computer Science & Telecommunications
TEI Larissa, Greece
2410626338, 0030
tsiakmaki@teilar.gr

Chrysafis Hartonas
Professor
Department of Computer Science & Telecommunications
TEI Larissa, Greece
2410684575, 0030
hartonas@teilar.gr

## ABSTRACT

Concept, Resource, Order, Product (CROP) is a reference architecture for adaptive Learning Objects owned by Semantic Learning Services developed by the second author. According to CROP, composite Objects are essentially recursive, and adaptively is an emergent property of Learning Service communication and collaboration. CROP is formally represented as an OWL ontology consisting of the framework's concepts and its definitions. In this paper we present a free, open-source, java-based graphical editor that populates CROP ontology with instances of Learning Objects at runtime through a (guided) graph-like interface. While developing this tool we proceeded to some adjustments on CROP ontology and further clarifications on the architecture. Our ultimate vision is to design Semantic Learning Domains where repositories of such ontologies exist and Services collaborate for delivering adaptive Objects to custom Learners' needs.

## Categories and Subject Descriptors

J.1 [Computer Applications]: Administrative Data Processing – Education

## General Terms

Design, Experimentation, Standardization.

## Keywords

CROP learning objects, learning services, semantic web.

## 1. INTRODUCTION

The domain of our discourse is Learning Services in the Semantic Web that offer adaptive Learning Objects to Learners. Some major issues on this field are (a) the architecture for Learning Objects, (b) the architecture of the Learning Service, e.g. the Participants, the discrete Roles that Participants can play, and the interactions that take places among the Participants, (c) the architecture of the Learning Domains, i.e. the environment that these Services live and collaborate, (d) the strategies that Services use in order to adapt their Objects according to Learner's needs throughout their collaboration with other Learning Services.

Concentrating on the first issue, we need an architecture on Learning Objects that enables us to: (a) describe atomic (unstructured, free-standing) or compound (involving other Learning Objects) Learning Objects, (b) describe static (structured at design time) or dynamic (restructured at run time, e.g. when learning difficulties are detected), (c) reuse them, (e) make them discoverable in a Learning Domain (search-able), and (d) tailor them to Learners' needs (adaptation).

We further argue that adaptively is not a requirement to be met by an individual Learning Object. On the contrary, we consider Learning Objects as part of a Learning Domain offered by Learning Services. We also adopt the Role Modeling approach [8], for modeling the reciprocal actions that take place within a Learning Domain.

In this report we present the implementation of an editor of CROP Learning Objects based on our on-going research on adaptive Learning Services. This paper is structured as follows: in section 2, we present the CROP Reference Architecture. Section 3 presents the editor of CROP Objects. Finally, we present the current open issues and future work.

## 2. INTRODUCING THE CROP REFERENCE ARCHITECTURE

### 2.1 The main idea

The CROP reference architecture [3] [1] is designed to model the structure of adaptive Learning Objects owned by Semantic Learning Services in a Learning Domain. The architecture supports the composition of Objects that suit to Learner's requirements, and the dynamic modifications of Objects during the learning process, through the collaboration of Learning Services [13]. An adaptive response is the result of reasoning on information available from the Learner Profile, the Learning Objects, and the learning process.

In CROP four notions take part: Concept, Resource, Order, and Product. A Learning Object is atomic (Resource), or composite (Product) that recursively contains other Learning Objects. Each Learning Object has one target Concept, i.e. a term that describes the educational objective of the Learning Object, and some prerequisite Concepts, i.e. Concepts that are required in order to use this resource. The whole learning process is monitored by Order, a process that is enabled for every Learner – Learning Object pair. Its main purpose is to help the Learner execute a learning process with success by providing the necessary information to the Learning Service that owns the Learning Object.

CROP Learning Objects are described by a set of metadata. The IEEE LOM standard [5] is adopted for that purpose.

Moreover, each CROP Object is associated to one or more Learning Style Designators [2]. The Designators indicate cognitive and learning style characteristics of Learners for whom this object is (more) appropriate.

## 2.2 Ontological representation of CROP architecture

According to CROP, all Concepts are part of the Content Ontology, an ontology that disambiguates all the educational objectives related to the Learning Object. Learning Objects, as we envision them, are contained in the Learning Object Repository of the Domain, and their Content Ontologies are also sub-ontologies of a Global Domain Ontology. Classes in the Content Ontology also populate the instances of Educational Objective class in CROP ontology (Figure 1).

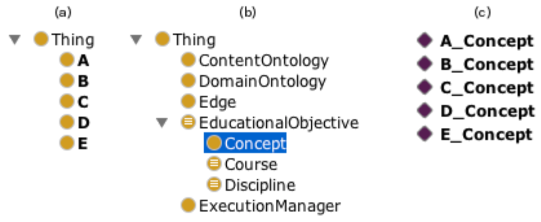

**Figure 1: (a) Content Ontology example (b) CROP Ontology sample, (c) Content Ontology' s Classes as instances of Concept class**

CROP objects have one Concept Graph, a graph that is formed by connecting the instances of educational objectives with the is-prerequisite-of relation.

CROP also builds one KRC (Knowledge Requirements Chart) Graph on top of the Concept Graph, where on each KRC Node a set of Learning Objects is associated. Each Learning Object in the set must have target Concept equivalent with the target concept associated to the KRC Node, and prerequisite Concepts among the prerequisite Concepts of the Learning Object and the prerequisites of the current Node. In case of a Learning Resource, KRC contains one KRC Node that is associated to the Physical Location of the actual learning material (e.g. a document, an image, a quiz...). Concept Graph Nodes that are not contained in the KRC Graph represent the prerequisite Concepts of the Learning Object. Later, the successful execution of all the Learning Objects contained in a KRC Node results in the acquisition of the associated Concept by the Learner.

To achieve an executable object, CROP specifies pairs of Execution Model (XModel) and Execution Graph (XGraph). The XGraph is built on top of KRC, keeping the set of Learning Objects that are associated on a KRC Node as set of Learning Act Nodes. In case of a Learning Resource, it contains one Learning Act Node that points to the Physical Location. The Physical Location is the escape condition of the execution loops that are necessarily contained in composite Learning Objects (Products). The XGraph also enables Authors to add edges between Learning Acts, so as to impose their execution sequence, and two types of Nodes: Dialogue and Control Nodes. Dialogues are static conversations with the Learner that enable the later to decide the next available Learning Act. Controls are associated with a threshold, in order to prevent or allow a Learner from the next Learning Act (e.g. a minimum average on the Assessment Resources that Learner should achieve in order to obtain a Concept). Figure 2 depicts the layers of CROP.

The XModel determines the sequence of the available Learning Acts wherever it is unspecified, respecting the prerequisite-of relations and XGraph Edges. It may be designed to implement specific instructional strategies. Thus each XModel provides a different learning experience to the Learners. Each XGraph may be associated to one or more XModels, but not vice versa.

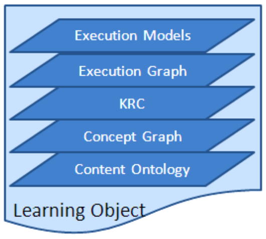

**Figure 2: CROP Object layers**

Three processes monitor the execution of a Learning Object namely Execution Node Managers, Execution Manager and Order. An Execution Node Manager (XNodeManager) is responsible for applying the sequencing rules of the XModel on an XGraph Group of Nodes that is associated to a KRC Node. Likewise an Execution Manager (XManager) is responsible for applying the sequence rules of an XModel on an XGraph.

Order process monitors the interaction between the Learner and the Learning Service and issues relevant reports to the Service that owns the Learning Object. Order notifies the Service for fail or successful execution of Learning Objects. Execution failures trigger adaptation procedures on the Learning Service, where the later provide alternative Learning Objects during the execution. These resources might be owned by the Learning Services or acquired after the communication and collaboration with other Services of the Learning Domain.

Figure 3 presents the main structure of CROP architecture.

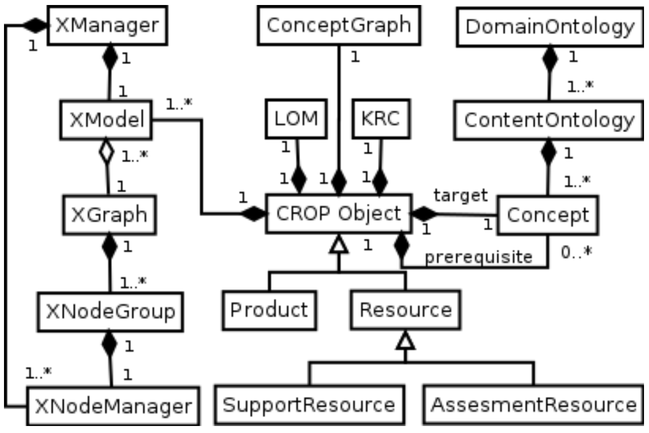

**Figure 3: CROP Architecture Class Diagram**

## 2.3 Glimpses of adaptation

We regard adaptation as an emergent property for Learning Service composition and collaboration rather than an inherent property for stand-alone Learning Objects. Under this premise, apart from the unaffected and simple customization through Dialogues and Controls, and the different learning experience that various XModels imply, adaptation is triggered by diagnosed Learner needs during the *search* and the *execution* of Learning Objects.

In Learning Domains, *search* is enabled for Learners and Learning Services. Learners submit a request for a Learning Object that honors a specific target Concept. Services submit a request for a Learning Object either when they are trying to respond to a request that initially could not satisfy by adapting an Object that they own, or during an execution after a notification from the Order. For instance, suppose the Learner's target is "Complex Number System". One Service owns an Object that requires the previews knowledge of "Real Number System". According to Learners' Model, Learner does not know about "Real Number System". Service tries to mutate the current Object by including Objects that teaches the prerequisites that are unknown by the Learner in the corresponding KRC Nodes and consequently to the XGraph as Teaching Acts. In case the Service does not own such Objects, it will start a new search to the Domain in order to find this type of Objects by other Services (service collaboration).

Also adaptation can happen during the *execution* of a Learning Object. For instance, when a Learner fails to achieve the required threshold of an Assessment Resource the Order process notifies sends a failure notification to the Learning Service. The later tries to find alternative Objects to enhance Learner’s experience on the subject. Consequently, the KRC Node is updated with more Objects, and XGraph with more Learning Acts.

## 3. IMPLEMENTING CROP

The outcome of CROP architecture is an ontology that describes all its concepts and the relations among them. During this research we have created a proof of concept editor for such Objects. The editor guides Authors across the steps of creating CROP Objects. Its final output is the CROP ontology populated with instances that depicts the designed Object.

### 3.1 Design principles

The CROP Editor is designed to simplify the creation of CROP Learning Objects, respecting the Leaners' goals. The following checklist identifies some core principles and practices to assist Authors to produce Objects that are appropriate in particular set of circumstances and prompt to changes.

- The use of *one* educational objective. CROP Objects have a specific and well-defined purpose, i.e. their target Concept. Having one target Concept encourages the construction of relatively small and self-contained Objects rather than large and cumbersome Objects. Cohesive Objects can easily be changed without affecting the rest composite Object, can easily be sequenced and reused across a Domain.
- The use of *some* prerequisite-of Concepts. In Concept Graph of composite Objects, each Concept has some prerequisite-of dependencies with other Concepts. Consequently, each Object is a set of cooperating Objects, each of which implements functionality independent of the others. Redundant associations can be avoided when Authors consider keeping low the number of Objects that will be affected when one Object change its content.
- The Content ontology should match both domain and learning task. Content ontology should describe courses' entities and their relation given the purpose of teaching and of interoperability across a Learning Domain.
- The KRC Node is a self-contained teaching action targeting a single Concept. Learners that successfully execute all the contained Objects eventually acquire the target Concept. Objects contained in the set should avoid repeat the same content. A Concept in the Concept Ontology can be defined as equivalent of a disjunction of a set of Concepts, e.g. ComplexNumber *Operation* can be a disjunction of *Conjuration* $\vee$ *Division* $\vee$ *Addition* $\vee$ *Multiplication* $\vee$ *Substruction*. Thus a KRC Node that is associated to *Operation* can contain Objects with target Concept *Operation* or *Conjuration*, or *Division* or *Addition* or *Multiplication* or *Substruction*.
- The use of *some* Execution Models. Each Object is associated to one or more Execution Model that ultimate on a different learning experience. It provides one type of adaptation based on Dialogues, Controls and the ordering of Learning Acts. The XModel can implement specific instructional strategies that later can be combined with Learner Model.

Generally, there is no one correct way to model a Learning Object. There are always viable alternatives. The most appropriate solution depends on the type of Learner that Authors have in mind and the extensions that they anticipate. Also, Learning Object development is an iterative process.

### 3.2 Discrete steps for creating CROP Objects using the editor

The Crop Editor builds Learning Objects according to CROP reference architecture. Initially, Authors create a new project. A project contains Learning Objects (Resources and Products), information about the Domain Ontology, the CROP Ontology that will be kept synchronized during the Authors manipulation and graph images.

The editor (Figure 4) contains 6 main panels (docks). On the left there is the Learning Objects explorer, where all the project’s Learning Objects are listed, grouped by their target Concept. Below, the Content Ontology panel shows the ontology classes and tools for simple ontology editing. The main panel contains the active’s Object graphs. Transfer handlers enable Authors to drag end drop concepts to Concept Graph panel. Below, the Problems and Console panels contain error and warning messages to the Authors during their working with a project. On the right, Palette contains the extra transferable nodes to Graphs. Below, the Properties panel is associated with the active graph and concept Author’s selections, and contains the information and actions related with the selected state.

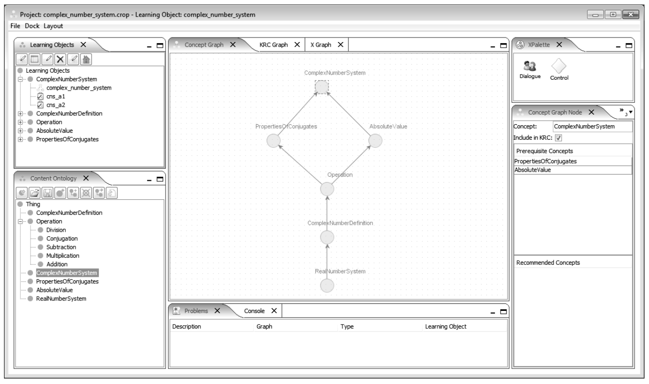

**Figure 4: The CROP Editor**

While describing the discrete steps for creating CROP Objects, some parts of example screenshots will be given, hopefully, for a better understanding. The example is about a Learning Object that teaches the “Complex Number System” to beginners.

Step 1 – Specify the Content Ontology

The Content Ontology is used to capture knowledge about some Learning Object. It describes the Concepts in the Object like the target and the prerequisite Concepts and also the relationships between those Concepts (Figure 5). In the editor, Authors can import a predefined ontology, edit it, or create a simple hierarchy of classes from scratch.

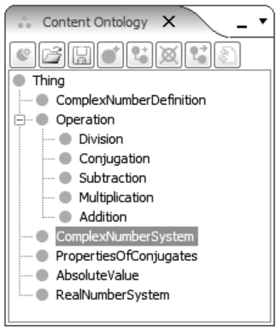

**Figure 5: Content Ontology Example**

Step 2 – Creating the Concept Graph

The Concept Graph relates Concepts with the is-prerequisite-of relation. Authors can drag & drop classes from Content Ontology to Concept Graph and create edges among them. During building the Concept Graph, Authors will discover the coexistence of many conceptualizations that one educational objective can have.

In Figure 6 *Complex Number System* Concept requires the knowledge of *Properties of Conjugates* and *Absolute Value*. The later, requires *Operation. Operation* Concept needs *Complex Number Definition*, and *Definition* about *Real Number System* Concept.

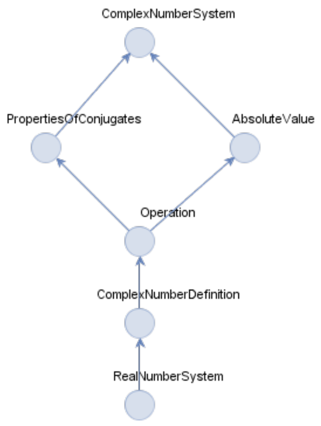

**Figure 6: Concept Graph example**

Step 3 – Creating the KRC Graph

The KRC Graph is built on top of Concept Graph. Editor duplicates Concept Graph Nodes and edges as KRC components. Only prerequisite Nodes are not contained. Authors can associate Learning Objects on each KRC Node. Objects may be contained in the project, or may be created as new (Resource or Product). The selected Node illustrates its target Concept and from the Content Ontology are inferred all the equivalent Concepts, if any.

The Editor prompts notices in a problem panel when Authors add Objects with prerequisite Concepts that do not exist in the active Object. In the example (Figure 7), *Operation* Concept is acquired after the successful execution of Resources of *Conjuration* (con), *Addition* (add), *Division* (div), *Subtraction* (sub) and *Multiplication* (mul). Notice that *Real Number System* Concept is not included in KRC (it won't be taught). Thus it is a prerequisite Concept of this Learning Object.

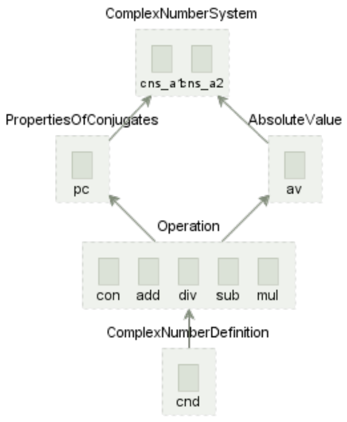

**Figure 7: KRC Graph example**

Step 4 – Creating the Execution Graph

The X Graph is built on top of the KRC. Each Object that is associated with a KRC Node becomes an X Graph Node. An X Graph Node is defined as a Learning Act (a Learning Act is associated with one CROP Object or with a Physical Location of a Resource Object), Control, or Dialogue Node. Learning Acts are grouped by their target Concept, a property that the associated KRC Node implied, defining X sub-Graphs. Authors can further group X Nodes and X sub-Graphs defining sequence and parallel groups. Edges of X Graph impose the sequence of the execution of Objects. X Graph Edges connect X Nodes, X sub-Graphs or groups. KRC edges are not transferred to XGraph, as they connect unit concepts, not groups or sub-graphs, but their prerequisite-of restrictions still stand. E.g. by default the XModel cannot execute *Addition* Object if Learner does not know about *Complex Number Definition,* and Authors do not need to explicitly state it again. Authors are allowed to add further edges, Controls and Dialogues in order to restrict the learning process.

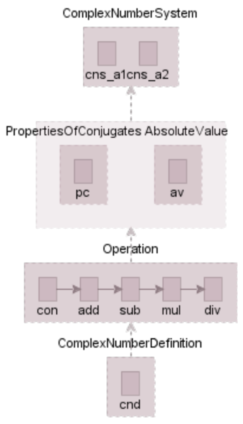

**Figure 8: X Graph example**

Keep in mind that Objects that are associated to a KRC Node do not have any sequence restriction. If strong restrictions need to be applied, it may imply that Authors should consider refactoring Content Ontology and Concept Graph by splitting or merging Concepts. Every Product can have one or more XGraphs. Resources' XGraphs are simple: by default they contain only one Learning Act that directs to the Physical Location of the actual resource.

In Figure 8 XGraph only specifies the order of the Resources in the Operation group of Nodes: first Learners learn about *Conjuration,* then *Addition,* then *Subtraction,* then *Multiplication* and finally about *Division*.

Step 5 – Creating the XModels

Finally, Authors can define the Execution Models (XModel). One XGraph can be associated to one or more XModels. Learner's goal is to conquer the root Node of the XGraph. To succeed that, Learner has to previously attain every node that is connected to. Apart from having to succeed on prerequisite Concepts, the sequencing rules, the Dialogues and the Controls, no other restrictions take place on the way of how a Learner will walk on the XGraph. On the one hand, Discrete Mathematics literature lists several well-structured algorithms for traversing a graph, e.g Depth First, or Breadth First. On the other hand, further priorities can be set, in case the next x step is still ambiguous. E.g. the preference of executing an assessment resource over a support resource, or the preference of executing a Learning Object that is associated to a video resource, over a text resource.

To support the above CROP associates an XModel with an execution algorithm, a priority list and a verbose level. Verbose level controls how detailed or general the information messages are going to be, i.e. whether the XManager will inform the reason of the next given Learning Object versus another to the Learner.

Generally, Authors can revise the initial Learning Object. The editor will synchronize the changes made on graphs and on the ontology through action listeners. E.g. if Author deletes a Concept Graph Node, the editor will delete the associated KRC Node and edges and subsequently the associated execution nodes and edges on X Graph.

## 3.3 More supported tools

The editor also supports further related functionalities, such as:

a. the export of owl comments and axioms of Concepts from Content Ontology in text files for creating learning material (text),
b. a simple LOM viewer of the Object, that is partially populated given the current CROP Object state,
c. a simple quiz editor for creating Assessment Resources,
d. a simple text editor for editing Support Resources,
e. image, video, pdf Viewers for Support Resources,
f. a sample execution simulator (under development).

## 3.4 Implementation details and source code

CROP Editor is written in Java and uses Swing GUI widget toolkit [10] to create its user interface. Also parts of our application rely on the following tools and source code: Docking Frames [11], for organizing the editor's panels such that the user can drag & drop them, mxGraph Java Graph Visualization Library [7], OWL API for working with OWL 2 CROP ontologies [4], HermiT reasoner [9], Apache log4j logging library [12].

The source code of the editor, binaries and examples can be found on GitHub [14].

## 4. Execution Details

Both XManager and XNodeManager processes are responsible for the execution of a CROP Object. Every CROP Object has one or more X Models and with the given Learner Profile the XManager selects the most appropriate Model. Each XModel is associated to one XGraph.

The XGraph contains restriction produced by the XEdges that the Author explicitly has inserted during the creation of XGraph and the prerequisite-of relations that are implied by the KRC Graph. The XManager given the XModel and the XGraph decides the next Execution Step (XStep), i.e. a Dialogue Node, a Control, or an X Parallel or a Sequence Group.

For the execution of a Group the XNodeManager process takes turn and every Group has its own Manager and each XNodeManager adheres to the XManager. XNodeManager decides the next XStep among Control Nodes, Dialogue Nodes, Learning Act Nodes and recursively XGroups.

For instance, on the *Complex Number System* CROP Object of our example, XManager will firstly execute the *Complex Number Definition* Group XStep. The XNodeManager calculates the next Step, which is the execution of 'cnd' Learning Act Support Resource associated to a document concerning the definition of Complex Numbers. The Operation Group is the next to follow. The corresponding XNodeManager will execute the sequence of Learning Acts as the edges impose: first with the *Conjuration*, followed by the *Addition*, the *Substation*, the *Multiplication* and finally with the *Division* Support Resources. Afterwards, the XNodeManager of the *Properties of Conjuration* and the *Absolute Value* parallel Group selects randomly the next XStep (suppose that further metadata leaves them still ambiguous e.g. they are both documents with the same execution time, level, density, etc.). The final step, which is the *Complex Number System* Group, contains two parallel Assessment Resources, and once again (suppose) the choice is done randomly.

## 5. OPEN PROBLEMS AND FUTURE WORK

We are currently working on enhancing the implementation with additional features, such as:

- The Gather feature that searches and collects Objects contained on a physical location in order to populate KRC Nodes.
- A simple IEEE LOM editor. Currently some elements can be exported from the CROP Object, e.g. the relations, classification, is part of… Other properties can be editable, e.g. the Author name.
- An embedded rich document editor for editing the actual Support Resources.
- The Import feature for importing IEEE LOM, or SCROM Objects [6], wrapped CROP Objects (SCROM compliant)
- The support further rules for avoiding inconsistences in the ontology and (regrettably) bugs fixes.

Our ultimate vision is to design a reference architecture of Semantic Learning Domains where repositories of ontologies with CROP Learning Objects exist and Services collaborate for delivering adaptive Objects to custom Learners' needs.

## 6. REFERENCES

[1] C. Hartonas. *The CROP reference architecture for learning objects in the semantic web*. Technical Report, TEI Larissa, Dept. Computer Science, 2010.

[2] C. Hartonas and E. Gana. *Adaptivity for knowledge content in the semantic web*. Proceedings of KGCM, 2008.

[3] C. Hartonas and E. Gana. *Learning objects and learning services in the semantic web.* In Advanced Learning Technologies, 2008. ICALT'08. Eighth IEEE International Conference on, pages 584–586. IEEE, 2008.

[4] M. Horridge and S. Bechhofer. *The owl api: a java api for working with owl 2 ontologies.* Proc. of OWL Experiences and Directions, 2009, 2009.

[5] IEEE, Learning Technology Standardization Committee (LTSC) - *Draft standard for Learning Object Metadata (LOM)*, 2002. Last accessed Oct 2008.

[6] ADL Initiative. *Sharable Content Object Reference Model (SCORM) 2004* 4th Edition Documentation, 2004.

[7] JGraph Ltd. *JGraphX (JGraph 6) User Manual - Version 1.11.0.0*. http://jgraph.github.com/ mxgraph/docs/manual_javavis.html, 2001. [Online; accessed 28-March-2013].

[8] A. Kritsimalis. *Role modeling for cdl specifications of web services.* Master's thesis, TEI Larissa Dept Computer Science & Staffordshire University, 2010.

[9] B. Motik, R. Shearer, B. Glimm, G. Stoilos, and I. Horrocks. *Hermit OWL Reasoner*. www.hermit-reasoner.com, 2007–2013. [Online; accessed 28-March-2013].

[10] Oracle America Inc. *Swing Java Foundation Classes*. http://docs.oracle.com/javase/7/docs/ technotes/guides/swing/index.html, 1993. [Online; accessed 28-March-2013].

[11] B. Sigg. *DockingFrames 1.1.1 - Core*. http://dock. javaforge.com/dockingFrames_v1.1.1/core.pdf, 2012. [Online; accessed 28-March-2013].

[12] The Apache Software Foundation. *Apache Log4j 2*. http://logging.apache.org/log4j/2.x/, 1999–2013. [Online; accessed 28-March-2013].

[13] M. Tsiakmaki, E. Gana, and C. Hartonas. *Adaptation as an emergent property of learning service composition in the semantic web (in greek)*. Proceedings of the 2nd Conf on Computer Assisted Education, Patras, Greece, pages 889–898, 2011.

[14] M. Tsiakmaki and C. Hartonas. *JCropEditor. - Source code, binaries, examples*. https://github.com/tsiakmaki/jcropeditor.git. [Online; accessed 28-March-2013].