Applying Object-Oriented Principles to the Analysis and Design of Learning Objects

Chrysostomos Chrysostomou and George Papadopoulos
Department of Computer Science, University of Cyprus, Nicosia, Cyprus
chrisc@cs.ucy.ac.cy
george@cs.ucy.ac.cy

Abstract: This paper forms part of a broader work examining the application of Object Oriented (OO) principles to the design and development of e-Learning material and its use within Learning Content Management Systems (LCMS). The preceding qualitative research has demonstrated the benefits of creating an OO methodology for the analysis, design and development of Learning Objects (LOs). Such benefits mainly include the high reusability, adaptability, standardization and time and cost effectiveness of LO development and use.

This paper focuses on defining a model for Object Oriented Learning Object (OOLO) analysis and design, as a first step towards a more extensive OO e-Learning methodology that will extend to OOLO implementation and use. The Unified Modeling Language (UML) notations are used to represent the relevant OO concepts, such as class structure, inheritance, aggregation, polymorphism, etc. The notations may be extended to reflect the specific needs of LOs. The main idea concentrates on developing a predefined hierarchy of OOLOs, which will be something similar to the Application Programming Interface (API) of OO languages such as Java. The existing OOLOs can be used as they are or they can be extended, through inheritance mechanisms, to create new ones. The OOLOs can then be combined using aggregation or other relationships to design, on the fly, larger learning units such as courses.

Planned further work includes the definition of an appropriate language (probably a hybrid OO and markup language) for implementing the OOLOs and the relationships between them, as well as relevant tools to enable the design and development of e-Learning content incorporating the proposed methodology.

This work is expected to enable e-Learning course developers and instructors to easily design and create standardized, highly reusable and adaptable e-Learning material, by extending and assembling existing LOs that encapsulate content, metadata, standards and operations, minimizing in this way the need to get involved with time-consuming and repetitive activities such as application of standards and metadata entry.

Keywords: e-Learning design methodologies, object-oriented learning objects, UML, learning object inheritance hierarchies, metadata inheritance, learning object assembly

1. Introduction

Preceding e-Learning research has shown that in the last few years e-Learning is being driven towards modularization of learning content (Chrysostomou & Papadopoulos 2005). The idea was to divert from the traditional and inflexible e-Learning courses, towards more flexible and reusable learning resources. Towards this cause the term Learning Object (LO) has been coined, referring to self contained learning resources serving a single objective and being able to be used and reused for learning (Polsani 2003). The concept of the LO is very similar to the concept of the software object that has prevailed in the Software Engineering (SWE) world. This association between LOs and OO SWE has been supported, in the last few years, by a number of researchers such as Poulton, Douglas, Downes, Robson and others. However, a number of technical and cultural difficulties relating to the development and use of LOs are still to be overcome in order to achieve combined high reusability and development efficiency in this area. Previous work (Chrysostomou & Papadopoulos, 2007) has shown that the development of an OO model that can be applied to the design and development of LOs can assist in overcoming a number of those difficulties and can provide a large number of benefits to e-Learning. This paper aims in setting up the ground for such a model, by demonstrating how OO modeling notations can be used to represent LOs and the relationships between them, defining a basic hierarchy of Object Oriented Learning Objects (OOLOs), and demonstrating how these notations and the predefined hierarchy can be utilized to easily design an OOLO based course.

In section one of this paper, a reference is made to the difficulties relating to the development and use of LOs as well as the ways in which an OOLO model can assist in overcoming those difficulties. Section two, describes the relevant parts of the Sharable Content Object Reference Model (SCORM), for the purpose of revealing the way in which the proposed model will link to this widely accepted suite of learning technology standards. In section three, the Unified Modeling Language (UML) notations are used to apply OO concepts to the design of LO classes and the relationships (inheritance, aggregation, etc.) between them. Section four proposes a sample hierarchy of LO classes, which can form the basis of creating LOs by using or extending the predefined LO classes. The notations and sample hierarchy of LO classes are put in use by designing a sample course consisting of existing LO classes from the sample hierarchy and new LO classes created by extending or aggregating the existing ones. Finally, section five summarizes the current work and lays down a plan for tasks that need to be carried out in order to extend the model into the actual development of OOLOs.

2. How can object orientation assist in overcoming difficulties in the development and use of LOs

Development and use of LOs entails a number of difficulties that need to be overcome in order to make LOs an attractive and efficient means for creating and delivering learning content. Development difficulties mainly arise from the complexity and the knowledge required to create LOs, whereas utilization difficulties are mostly based on the dependency of learning content on the context for which it is developed as well as the predefined teaching style that each LO may be bound to. These difficulties and the ways in which an OOLO model may assist in overcoming them, have been discussed in Chrysostomou & Papadopoulos (2007). These difficulties are:

- LO development requires extensive knowledge (LO theory, standards, XML, etc.).
- Each LO has to be created from scratch or existing LOs have to be modified in order to produce the desired content.
- Similar LOs are often created from scratch causing unnecessary waste of time, money, effort and possible risk of incompatibilities.
- For each new LO a large number of metadata has to be entered manually.
- Each new LO is untested leading to unreliability of new LOs.
- LOs are often bound to specific context and cannot be easily reused.
- The internal structures of a LO is often confusing.
- Pre-written learning content usually follows a specific form that does not suit each individual's teaching style making people unwilling to use it.

The aim of this work is to overcome as many as possible of the above difficulties by applying an OO methodology in the design and development of LOs. In other words this is an attempt to apply the lessons we learned from the development of the OO SWE paradigm, to the design and development of LOs. In brief, the idea is based on the definition of the structure of a LO in an OO way (i.e. similar to a class in OO SWE) encapsulating attributes and operations (Robson 1999) and also enabling the application of inheritance (Daniel & Honggang 2003) as well as other OO concepts such as abstraction, aggregation, polymorphism, etc. A basic hierarchy of OOLOs should also be defined, through which new LOs can be created either by using or extending existing LO classes. This hierarchy is expected to form a library of LO classes, similar to the Application Programming Interface (API) provided by OO programming languages like Java. Following is a number of benefits that may be realized by adhering to an OOLO model that will assist in overcoming the difficulties mentioned earlier:

- Due to inheritance:
  - New LOs can be developed faster, easier and with less cost by extending existing LOs.
  - New LOs will inherit the properties of their predecessors including metadata (minimizing metadata input).
  - New LOs will be based on existing tested ones (more reliable LOs).

- Existing LOs will already follow the appropriate standards and consequently the developer will not need to deal with ensuring application of standards.
- All new LOs will be created following a common (standardized) structure and they will consequently demonstrate increased interoperability.
- The hierarchy can be extended in a way that best suits specific domains or organizations.
- LOs can be better maintained (e.g. easier modification of LOs by modifying the LO class they inherit from).

- Due to aggregation:
  - LOs can easily be combined to form larger learning contents, which can also be reused when necessary.
- Due to abstraction and encapsulation:
  - LOs will not be bound to any specific context and hence be highly reusable.
  - The developer does not have to know the internal structures of the LOs (less knowledge needed to develop LOs).
- Due to polymorphism:
  - LOs will have the ability to be used in a variety of contexts.
  - Instructors will be able to use the LOs with a variety of teaching styles or apply to them their own preferred style.

3. The relation of an OOLO model and SCORM

The SCORM (Sharable Content Object Reference Model) is a suite of standards developed by the Advanced Distributed Learning (ADL) consortium. SCORM includes a number of standards developed by a variety of organizations (e.g. IEEE, AICC, IMS), relating to e-Learning and LOs. Part of SCORM is the SCORM CAM (Content Aggregation Model) that mainly deals with the components that make up SCORM conformant learning content. Following is a brief description of the two main parts of the SCORM CAM, the *SCORM Content Model* and the *SCORM Content Packaging Specification*. The purpose of this description is to enable us to determine which SCORM components our OOLO model applies to, and to ensure there are no unwanted conflicts between our model and SCORM.

3.1 The SCORM content model

The SCORM Content Model describes the components that are used to create a learning experience from learning resources. The most basic form of a SCORM component is an Asset. Assets are basically the raw material by which learning content is created. An Asset can be a text file, a web page, an image, a video file or any other kind of media (figure 1). An Asset may also consist of a number of other Assets (e.g. a web page that includes images).

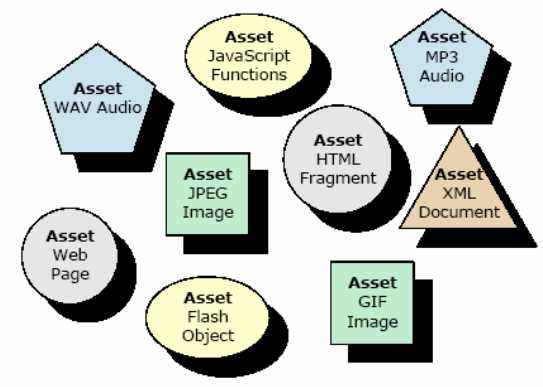

Figure 1: Examples of SCORM assets (ADL, 2004)

The next level of SCORM components is called a SCO (Sharable Content Object). A SCO is made up from one or more Assets (figure 2) and it represents the lowest level of granularity of a learning resource that can be tracked by a Learning Content Management System (LCMS). A SCO is what we would generally describe as a LO and it should offer the characteristics of a LO including: serving a single learning objective, independency from context, reusability, etc. Assets and SCOs are collectively described in SCORM as *resources*.

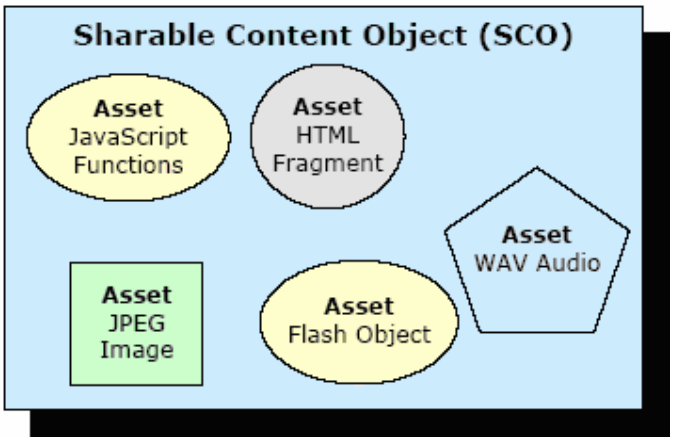

Figure 2: SCO (ADL, 2004)

Further up in the SCORM Content Model hierarchy is the Content Organization. The Content Organization is a map that represents the use of Assets and SCOs within a hierarchy of Activity Items and the relationships between these items, forming a learning experience (figure 3).

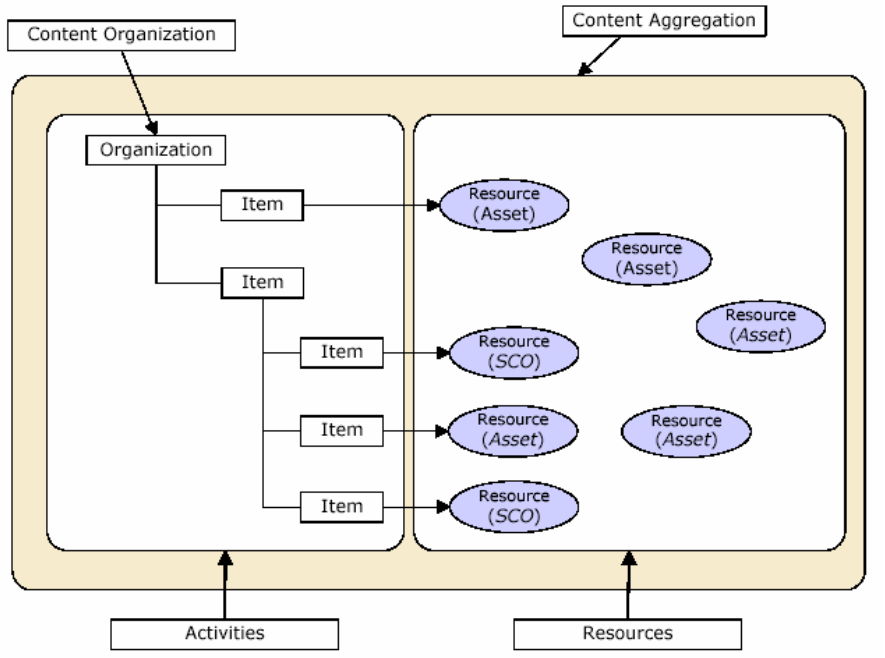

Figure 3: Content Organization (ADL, 2004)

3.2 The SCORM content packaging specification

The SCORM Content Packaging Specification describes the way in which SCORM content is packaged in order to provide a standardized way to structure and exchange learning content between learning systems and tools. According to the specification a SCORM Content Package (figure 4) consists of:

- The *manifest* file, which is an XML document describing the package's meta-data, structure and associated resources.

- The *physical files* making up the learning content.

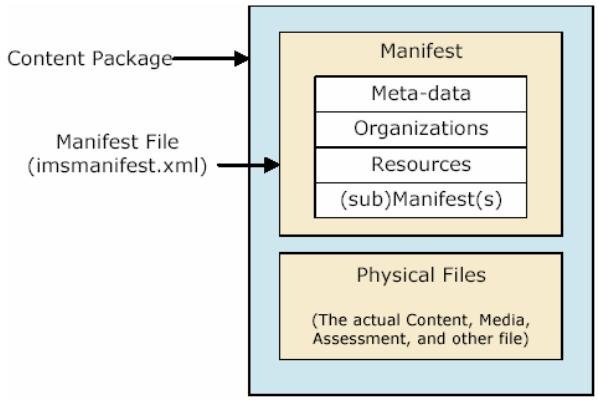

Figure 4: Content Package (ADL, 2004)

The manifest consists of the following:

- *Meta-data*: describing the content package as whole.
- *Organizations*: describing the structure of the learning resources.
- *Resources*: references defining the learning resources included in the package.
- *(sub)Manifest(s)*: describing nested units of instruction.

From figure 3 we can see that the term “resources” refers to either Assets or SCOs. We can consequently deduce that the “resources” part of the manifest file is basically defining the Assets and SCOs that participate in the package (this is also mentioned in SCORM CAM section 3.4.1.21). In the manifest file the set of resources that are part of a content package is represented by the parent element <resources> that acts as a container for a number of <resource> elements that contain the actual reference to the relevant resources (figure 5). The SCORM CAM (section 3.4.1.20) specifically defines that “there is no assumption of order or hierarchy of the individual <resource> elements that the <resources> element contains” (ADL, 2004).

```
<manifest>
  <metadata/>
  <organizations/>
  <resources>
    <resource identifier="RESOURCE2" type="webcontent" href="intro1.htm">
      <file href="intro1.htm"/>
    </resource>
    <resource identifier="RESOURCE3" type="webcontent" href="content1.htm">
      <file href="content1.htm"/>
    </resource>
    <resource identifier="RESOURCE4" type="webcontent" href="summary1.htm">
      <file href="summary1.htm"/>
    </resource>
  </resources>
</manifest>
```

Figure 5: <resources> and <resource> elements of the manifest file.

From the preceded analysis of the SCORM CAM we can deduce that an OOLO model relates only to the development of the actual resources that are referenced by the <resource> elements in the manifest files of a content package. Therefore, reference to LOs hereafter will refer to either SCOs or Assets. Since our model will deal with the design, development and structure of resources, it will not undesirably affect or come in conflict with the SCORM specifications, as these in general, deal with higher level structures of learning content. However, if desirable, such a model can minimize the <resource> elements definitions within the manifest file, as LOs that demonstrate OO functionality will offer aggregation capabilities and consequently a number of resources will be able to be referenced as a single resource.

4. OO notations for representing LOs and LO relationships

As a first step towards the development of OOLOs, appropriate notations must exist to enable the design of OOLOs and the relationships between them. From the OO theory we can borrow a number of concepts that can be applied to LOs to offer the desired functionality. These concepts have been discussed in Chrysostomou & Papadopoulos (2007), and the benefits that can be achieved from their application were outlined in section one of this paper. The main OO concepts that can be applied to LOs are: class structure, inheritance, aggregation, and polymorphism. Class packages are also shown in this section as a way of packaging together related LO classes. The Unified Modeling Language (UML) is used in the design of OO systems and it has also been used in the past to represent SCORM conformant learning content (Hu, 2005), therefore, UML notations will be used to demonstrate the application of the OO concepts to LOs. Learning related examples are used to make the suitability of the UML notations to OOLOs more obvious.

4.1 LO class structure

Similarly to the representation of software objects, each LO must be represented as a class that consists of the following:

- A class name.
- Attributes: these should include all the relevant metadata, references to content elements and any other required attributes. The metadata requirements that are mandatory for each type of LO (SCO or Asset) are specified in the SCORM Metadata Application Profile Requirements (SCORM CAM section 4.5.2).
- Operations: any operations that can be performed by or on the specific LO such as invoke, add, delete, change the value of a specific attribute, etc.

A class is represented by a class diagram, which is a rectangle that is divided into three horizontal sections. The top section includes the class name. The middle section includes the attributes and the bottom section the operations. Figure 6 demonstrates how a SCO representing an introductory lecture on Artificial Intelligence could be designed. Note that only a sample of the normally required attributes and operations are used in the example.

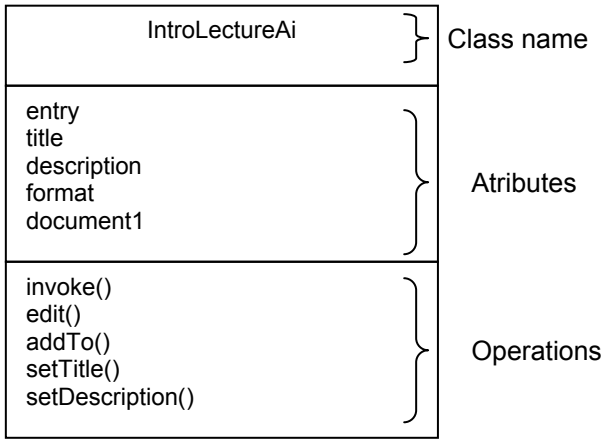

Figure 6: Class diagram for a LO class representing a lecture

4.2 Inheritance

A LO class (subclass) can be created by extending the functionality of an existing class (superclass). The subclass will inherit all the properties (attributes and operations) of the superclass and more properties can be added to it. A learning related example could be a *Question* superclass from which subclasses *EssayQuestion* and *MultipleChoiceQuestion* may derive (figure 7):

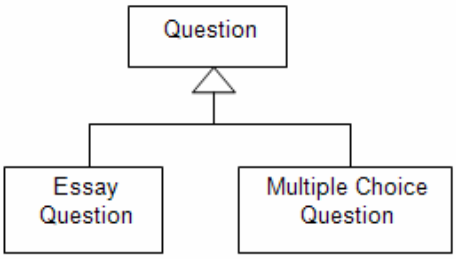

Figure 7: Inheritance class diagram

4.3 Aggregation

A LO class can be created by combining a number of instances of an existing class. For example, class A is created using a number of instances of class B. The number of instances of class B that participate in class A is specified by the multiplicity that appears on top of the connecting line. A learning related example could be a test that is made up of one or more essay questions (figure 8a):

Figure 8a: Aggregation class diagram

The following example (figure 8b) represents a *Test* class that consists of at least one but no more than three instances of an *EssayQuestion* class, and it could also include zero up to ten *MultipleChoiceQuestions*:

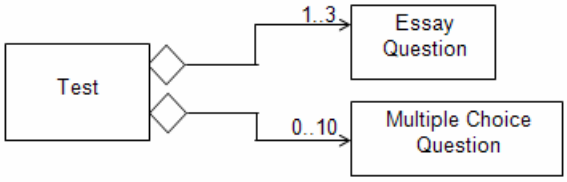

Figure 8b: Aggregation class diagram with a class deriving from multiple classes

4.4 Polymorphism

Polymorphism is the ability of a single component to take more than one form. Specifically, in the case of classes and their operations, polymorphism refers to the ability of an operation of a subclass to override the functionality of the same operation of the superclass. In the example that follows (figure 9) a *Document* superclass exists that includes a *display()* operation. A document may be in the form of a text document (i.e. ".doc") or in the form of a web page (i.e. ".html"). For this purpose two subclasses (*TextDoc* and *WebDoc*) are created by extending the functionality of the *Document* class. Each one of the subclasses includes its own *display()* operation. In the case of the *TextDoc* class, the *display()* operation will display the document as text (i.e. as a ".doc" file). In the case of the *WebDoc* class, the *display()* operation will display the document as a web page (i.e. as a ".html" file).

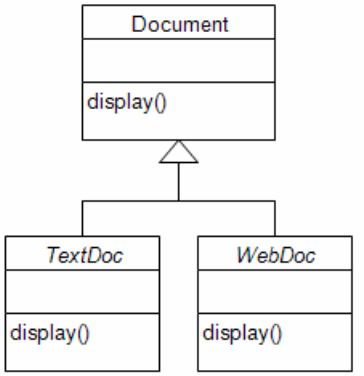

Figure 9: Inheritance class diagram demonstrating the polymorphic behavior of the method display()

4.5 Packages

A package is a way of grouping together a number of classes that are in some way related to each other. For example a package called *Assessments* may group classes that relate to assessments, such as *Assessment*, *Examination*, *Assignment, Task* etc. Although LOs at a higher and more abstract level tend not to be bound to specific context, it is possible to create LOs for use in specific domains (e.g. Mathematics). For this purpose, a package may also be used to group classes that belong to the same domain. If changes in a class that belongs to one package may have an effect to a class in another package we can show that through a dependency diagram. For example, in the diagram below (figure 10) we can see the *Assessments* and the *Maths* packages. A change on the *Task* class may affect the *Exercise* class, as the *Exercise* class may inherit from the *Task* class. The dependency between the two diagrams is shown with the broken line arrow.

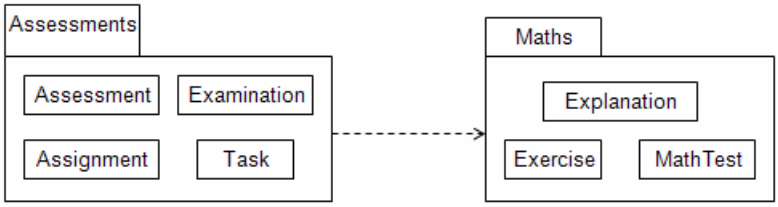

Figure 10: Packages of LO classes

5. Designing learning content based on OOLOs

In order to design learning content that consists of OOLOs, a predefined hierarchy of such objects should exist. In this section we devise such a hierarchy for the purpose of demonstrating how it can then be applied, in combination with the notations described earlier, for the purpose of designing larger learning contents, such as courses. The hierarchy of OOLOs shown here is kept rather simple and used for demonstration purposes, although it could form the basis for the development of a full scale hierarchy. The design of a sample course is then demonstrated, based on the predefined OOLO hierarchy.

5.1 A hierarchy of predefined OOLOs

In the previous section it has been demonstrated how OO concepts and design techniques can be applied to LO design. In this section those techniques will be used to design a predefined hierarchy of LO classes. The predefined hierarchy of classes is a concept widely applied in OO Software Engineering. An example would be the Java API (Sun Developer Network) which provides a predefined hierarchy of Java classes. The highest level class is called *Object* and it includes all the basic functionality that any Java class should have. All the other classes inherit, directly or indirectly, from the *Object* class, forming in this way a predefined hierarchy of classes. Most of these classes may be used as is or extended to serve more specific purposes. Related classes are grouped into packages to enable easier classification and management.

The LO hierarchy should serve similar purposes as the Java API, with application (of course) to the e-Learning domain. The highest level class is called *LearningObject* and it includes all the attributes and operations that are common between all LOs. Subclasses that inherit from the LearningObject class include the *TeachingObject* and the *AssessmentObject*. These top level classes are abstract and actual LOs cannot be created from them. Lower level classes will include *Lecture*, *Seminar*, *Workshop*, *Examination*, *Assignment*, *Task* etc. These can further be extended to form even lower level classes like *PracticalExamination*, *TheoryExamination,* etc.

It should be pointed out that the hierarchy demonstrated in this paper (figure 11) is a sample hierarchy, mostly used for demonstration purposes, but also for providing the basis for a more extensive and enhanced LO hierarchy. Any of the classes in the hierarchy may be extended to form new classes and consequently extend the hierarchy both horizontally and vertically.

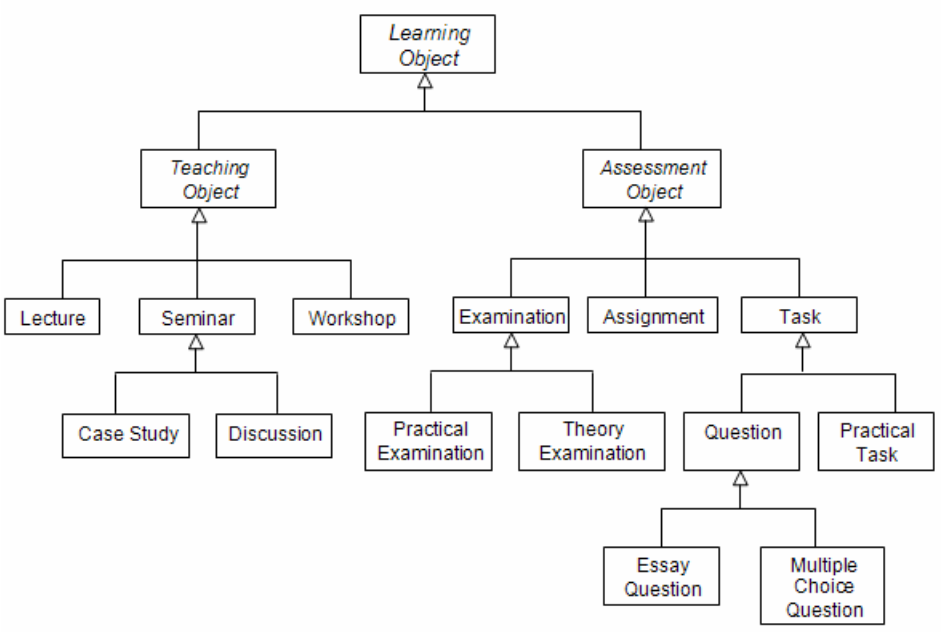

Figure 11: A hierarchy of LO classes

5.2 Course design based on the OOLO hierarchy

Assuming the existence of the above hierarchy of classes we can proceed to design an e-Learning course based on OOLOs by making use of the notations described in the previous section. Let us consider a rather traditionally structured course that is made up of the following:

- Twelve weeks of lectures (one lecture each week).
- Twelve seminars: Each seminar could be a case study or a discussion; there must be at least two but no more than four case studies in each course and the remaining seminars should be discussions (e.g. web forums, online chats, etc.).
- One assignment.
- Two theoretical examinations (a midterm examination and a final examination).

To design the course we start from an abstract class called *Course* and then we use classes from the basic LO hierarchy from figure 10 to show the LO classes that make up the whole course (figure 12). The *Course* class is made up from twelve *Lecture* objects, twelve *Seminar* objects, one *Assignment* object, and two *Examination* objects. For simplicity purposes we consider the *Lecture* as a simple class and therefore we do not analyze it further. Each one of the twelve *Seminar* objects, is either a *CaseStudy* or a *Discussion*. Multiplicity appears on the *Seminar* superclass, to show the total number of seminars in the course, but also on each one of the subclasses. The total number of *CaseStudy* objects and *Discussion* objects in the course should be twelve, which is the total number of seminars. The *Assignment* class is made up from at least one but no more than five

PracticalTask objects. The course finally includes two *TheoryExamination* objects, from which one is a *MidtermExamination* and the other one is a *FinalExamination*. The *MidtermExamination* and the *FinalExamination* classes, although they do not already exist in the predefined hierarchy, they can easily be created by extending the functionality of the *TheoryExamination* class. The *MidtermExamination* consists of one up to three *EssayQuestion objects* and it may include up to ten *MultipleChoiceQuestion objects*. The *FinalExamination* consists of one up to five *EssayQuestion objects* and at least ten up to twenty *MultipleChoiceQuestion objects.*

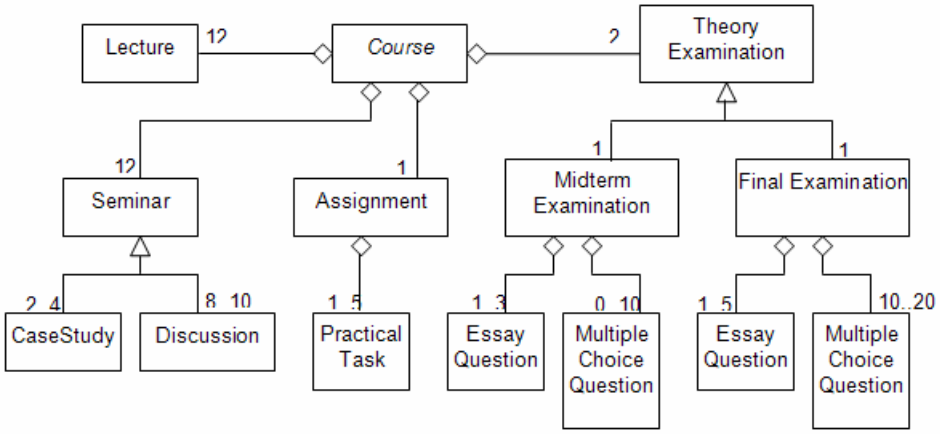

Figure 12: A course designed from LO classes and class aggregations

The above example aims in demonstrating the process of designing a course based on a predefined hierarchy of LOs. The example demonstrates the ability to use predefined LO classes, to use aggregations of classes and multiplicities to design complex LOs but also to add new LO classes by extending existing ones.

6. Conclusions and future work

This work has demonstrated the ability to apply Object Oriented concepts and design principles to the design of LOs. These concepts and principles were applied to “resource” components (Assets and SCOs) of the SCORM model to ensure interoperability with this widely accepted and standardized model but also to provide an extension to its functionality that would provide additional reusability of the learning resources. OO concepts such as classes, inheritance, aggregation and more were used for the design of OOLOs through the application of UML notations to the learning domain. A sample hierarchy of OOLOs was developed and it was used to demonstrate the design of a sample course.

This paper concentrated on the application of OO design to LOs. Naturally, the next step would be to provide the means for implementing LOs in this OO way and also to implement the OOLO relationships that are described in this paper. In order to be able to proceed with this implementation an appropriate development language should exist. The IEEE Learning Technologies Standards Committee (LTSC) has developed the IEEE P1484.12.3 standard (IEEE LTSC, 2005) that defines an XML Schema Definition Language Binding for Learning Object Metadata (LOM), which basically defines the XML elements that can be used to represent LOM elements. Since XML is used to represent the metadata, it is expected that it should also be used in creating the LOs. However, the OOLO language should provide OO functionality, i.e. provide attributes and operations within the LO class structure as well as inheritance, aggregation, polymorphism etc. An XML based language with OO features, the o:XML (Klang, 2002), is expected to be used towards this purpose. The o:XML, can act as both a markup language and an OO programming language and it can be used to develop the desired OOLO language. o:XML has an open specification and it offers a lot of the desired functionality as it supports attributes, procedures, functions, inheritance, polymorphism and more. It is also dynamic, as o:XML programs can generate XML code during their execution, a characteristic that could prove very useful for flexible and adaptable OOLO development.

Tasks to follow this work will therefore include:

- The specification of an appropriate language that will enable the implementation of the OOLO model.
- The implementation of the OOLO class, in such a way to provide encapsulation of attributes and operations as well as extension mechanisms to enable inheritance.
- The implementation of all the OOLO relationships described in this paper.
- The implementation of a sample library of hierarchically related OOLOs that can be used to create LOs.
- And finally, the development or adaptation of appropriate tools to enable the design and development of OOLOs based on the proposed model.

The outcome of this work is expected to enable a learning content developer to easily and efficiently design learning content by applying OO design principles and notations, to make use of existing LOs, to efficiently create new highly reusable and reliable LOs by utilizing existing ones and finally to flexibly create larger learning contents, i.e. courses, that will best suit their individual needs.

Acknowledgements

The work described in this paper has been carried out partially within the framework of the project UNITE: Unified e-Learning Environment for the school, Contract Number: 026964, partially funded by the European Commission under the 6th Framework of the IST (Information Society Technologies) Program.

References

ADL (2004), *SCORM*, [online] http://www.adlnet.gov/scorm/index.aspx

Chrysostomou C., Papadopoulos G. (2005) "An evaluation of e-Learning technologies and trends: Establishing an object-oriented approach to learning object design and development", *First International Conference on E-Business and e-Learning (EBEL'05)*, Amman-Jordan, [online] http://www.psut.edu.jo/EBEL/pdf/16.pdf

Chrysostomou C., Papadopoulos G. (2007) "Towards an Object-Oriented model for the design and development of Learning Objects", *International Journal on e-Learning,* Awaiting publication in Volume7, Issue 2.

Daniel, B., Honggang W. (2003) "Developing a schema for Learning Object based on Object Oriented Model of object Inheritance", Proceeding of the 3rd IEEE International Conference on Advanced Learning Technologies, [online] http://csdl.computer.org/comp/proceedings/icalt/2003/1967/00/19670439.pdf

Douglas I. (2001), "Instructional Design Based on Reusable Learning Objects: Applying Lessons of Object-Oriented Software Engineering to Learning Systems Design", 31st ASEE/IEEE Frontiers in Education Conference, USA, [online] http://citeseer.ist.psu.edu/524398.html

Downes S. (2004), "Object Oriented Learning Object", Montreal, 26 November 2004, [online] http://www.downes.ca/files/1

Hu, S. C. (2005), "Application of the UML in modelling SCORM-conformant Contents", College of Computing and Informatics, Providence University, Tawain, [online] http://csdl2.computer.org/comp/proceedings/icalt/2005/2338/00/23380200.pdf

IEEE LTSC (2002),1484.12.1-2002. IEEE Standard for Learning Object Metadata, [online] http://ieeeltsc.org/

IEEE LTSC (2005),1484.12.3-2005 IEEE Standard for Learning Technology- XML Schema Definition Language Binding for Learning Object Metadata, [online] http://ieeeltsc.org/

Klang, M. (2002), *Object Oriented XML*, [online] http://www.o-xml.org

Polsani, R. P. (2003), "Use and Abuse of Reusable Learning Objects", *Journal of Digital Information*, Volume 3, Issue 4, Article No. 164, [online] http://jodi.ecs.soton.ac.uk/Articles/v03/i04/Polsani

Poulton, C. (2005), "Applying Principles of Software Engineering Design to the Development of Reusable Learning Objects", Department of Electronics and Computer Science. University

of Southampton [online]
http://www.ecs.soton.ac.uk/~cmp301/comp6009/IRP%20cmp301.pdf
Robson, R. (1999), "Object-oriented Instructional Design and Web-based Authoring", *Journal of Interactive Learning Research*, [online]
http://www.eduworks.net/robby/papers/objectoriented.pdf
Sun Developer Network, *Java 2 Platform Standard Edition 5.0 API Specification,* [online]
http://java.sun.com/j2se/5.0/docs/api/index.html