SWOLO: A SWEBOK-based Ontology for Learning Objects

Imran A. Zualkernan

American University of Sharjah, UAE, izualkernan@aus.edu

***Abstract* — Re-usable learning objects (RLO) have recently gained popularity as the primary organizational structure in software that manipulates and renders e-Learning content. Cisco defines a comprehensive domain independent methodology for design and authoring of an RLO. This paper presents a new ontology called SWOLO that extends Cisco's methodology for building learning objects for Software Design. SWOLO is based on the IEEE Software Engineering Body of Knowledge. A case study of an existing learning object from the open courseware initiative (OCW) demonstrating the use of this ontology is also presented.**

***Index Terms* — Software Engineering body of knowledge, Learning objects, software design**

I. Introduction

Re-usable learning objects (RLO) are often used as the organizational primitives for computer programs delivering online learning content [1][2][3]. The concept of a "learning object" is derived from object-oriented design. As the name implies, a learning object is typically an abstract data type whose structure represents the various learning components. While a learning object can takes many forms, much like object-oriented design, the primary motivation behind using learning objects is to structure the software delivering learning content in a manner such that the content can be easily re-used in various e-learning applications.

Cisco [1] provides a widely used methodology and framework for authoring and organizing an RLO. This paper extends Cisco's learning object methodology by incorporating elements from the Software Engineering Body of Knowledge [4] to construct a methodology that is specifically tied to Software Design.

II. Cisco's Learning Object Design Ontology

Fig. 1 shows a simplified ontology [5] for Cisco's methodology. As Fig. 1 shows, in this ontology, each lesson is represented by an RLO. Each RLO, in turn, has *Learning Objectives*. Each RLO also contains a number of *Reusable Information Objects* (RIO). Each RIO consists of *Content Items*, *Practice Items* and *Assessment Items*. A *Content Item* typically consists of the learning materials while *Practice* and *Assessment* items represent quizzes and exams. A *Practice* item typically represents a formative assessment where the objective is to stimulate the learning process. There are two types of *Assessment Items*; post- and pre-. Pre-assessments formally evaluate a student's understanding before attempting the lesson while the post-assessment measures their performance after the lesson.

Each RIO also has learning objectives. The *Cognitive Level* of an RIO represents the desired level of competence and identifies how the learner will remember or use the skills and knowledge they are acquiring. The Cognitive Level is generally based on David Merrill's work and more widely on Bloom's taxonomy [6]. Bloom's taxonomy provides a well-known framework outlining generic categories of levels of learning for cognitive tasks ranging from Knowledge, Comprehension, Application, Analysis, Synthesis and Evaluation. Each successive level of learning requires a higher level of understanding. For example, Knowledge level understanding of the software engineering concept of "coupling" may only require one to recall what coupling is and to simply list the types of coupling (e.g., stamp, data, common etc.). A Synthesis level of understanding of coupling, on the other hand, may require students to construct a design that minimizes particular types of coupling.

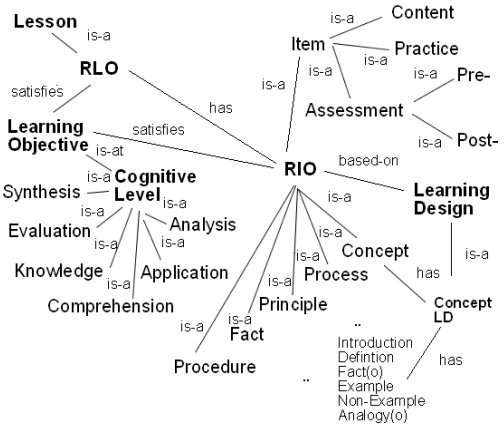

Fig. 1. Ontology for Cisco's Learning Object Methodology

As Fig. 1 shows, from an organizational perspective, Cisco classifies an RIO into five categories; *Concepts*, *Facts*, *Procedures*, *Processes* and *Principles*. For example, "Router" is a *Concept* to be explained. Similarly, "Guidelines for Designing a Multilayer Switched Network" is an example of a set of *Principles* used for designing networks. For each type of object, Cisco also defines a *Learning Design* template. A *Learning Design* template describes what an explanation for a particular type of RIO may contain. For example, the Learning Design for a *Concept* RIO consists of

Introduction, *Definition*, *Fact*, *Example*, *Non-Example* and an *Analogy*; *Fact*, *Example* and *Analogy* are optional elements. Similarly, the *Learning Design* for a *Process* RIO consists of *Introduction*, *Fact*, *Staged Table*, *Block Diagram* and a *Cycle* chart. This means that any RIO explaining a process will contain a section on introduction and depending on the type of information object, may contain the description of a fact, a staged table, a block diagram or a cycle.

The primary premise of this paper is that rather than using the generic organizational principles of "Concept", "Principle," etc., a learning object for Software Design should instead employ an ontology based on the Software Engineering Body of Knowledge (SWEBOK) [4]. The same argument can be extended to other part of the software engineering such as requirements or maintenance.

III. SWOLO – A LEARNING ONTOLOGY BASED ON SWEBOK

According to SWEBOK, "Design is defined in [IEEE610.12-90] as both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process." (pp. 3-1). The broad components of the design ontology articulated in SWEBOK are as follows:

1. **Design Issues** – SWEBOK includes the key issues in design to be Concurrency, Control and handling of events, distribution of components, Errors and exception handling, Interaction and presentation and Data persistence.
2. **Enabling Techniques** - These principles are common to all software engineering techniques. SWEBOK defines enabling techniques to be Abstraction, Coupling and Cohesion, Decomposition and Modularization, Encapsulation/Information hiding, Separation of Interface and Implementation, and Sufficiency, Completeness and Primitiveness.
3. **Software Design Strategies** - Design strategies include Function-Oriented structured Design, Object-oriented Design, Data-structured-design, Component-based Design and others.
4. **Software Design Notations** - Software design notations often involve using different types of representations. Some examples from SWBOK include ADLs, class and object diagrams, component diagrams, CRCs, deployment diagrams, ER-Diagrams, IDLS, Jackson's Methodology and structural charts. The dynamic representations include activity diagrams, collaboration diagrams, data-flow diagrams, state diagrams, sequence diagrams, formal specification languages and pseudo-code and PDLs.
5. **Software Structure and Architecture –** is a description of the sub-systems and components of the system. According to SWEBOK, this also includes Architectural structures and viewpoints, Architectural styles, Design patterns and frameworks.
6. **Software Design Quality Analysis and Evaluation** – includes software quality descriptions like quality attributes, quality analysis and evaluation techniques (including reviews, simulations and prototyping), and measures like function-oriented and object-oriented measures.

SWOLO (Software Engineering Ontology for Learning Objects) is based on the design ontology implicit in SWEBOK. SWOLO adds learning design components specific to software engineering to each of the SWEBOK concepts to arrive a significantly different ontology than Cisco's.

The main components of SWOLO are shown next by reverse engineering the structure underlying an existing "lesson" or a learning object. The purpose of the case study is to show that SWOLO provides a natural and a better representation for organizing software that manipulates and delivers learning for Software Design.

IV. CASE STUDY

This case study uses the "lecture 2" module in the"6.170 Laboratory in Software Engineering" (2001 version) course from MIT's open course initiative [7]. This module is available is a PDF file on the OCW website. This randomly selected module was analyzed to determine how well the SWOLO ontology fits the Software Design concepts being explained.

A. Learning objectives

This module identifies its learning objectives to "introduce some notions for talking about parts and how they relate to each other" and "identifying the problem of coupling and showing how coupling can be reduced." The objectives for this learning object in SWOLO are show in Fig 2. For example, LOB1 represents the first learning objective and this learning objective was first mentioned on line 2 in the PDF file of the module.

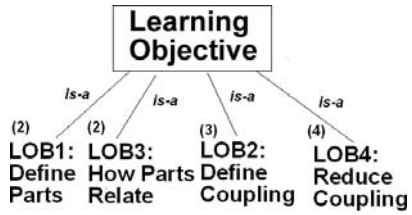

Fig. 2. Instances of Learning Objectives for the sample module in SWOLO

The rest of the module introduces a number of RIO's related to these learning objectives.

B. Enabling Techniques

The module explains the two common software engineering enabling techniques of Decomposition and Decoupling.

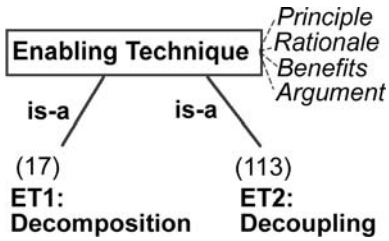

Fig. 3. Instances of Enabling Technique RIO's in SWOLO

As Fig. 3 shows, the module contains two instances of the Enabling Technique RIO. The first instance (ET1) is introduced early on (line 17) while the second one (ET2) is introduced much later (line 113) in the module. In SWOLO, an Enabling Technique has four facets of Learning Design; *Principle*, *Rationale*, *Benefits* and an *Argument*. For example, the *Principle* behind the *Decomposition* RIO shown in Fig. 4., is to break up a program into a collection of parts. The *Rationale* in this particular RIO is "correctness" and "stability". The *Benefits* are division of labor, reuse, modular analysis and localized change. Finally, the *Argument* is provided using Dijkstra's N parts example and Simon's description of the two clock makers.

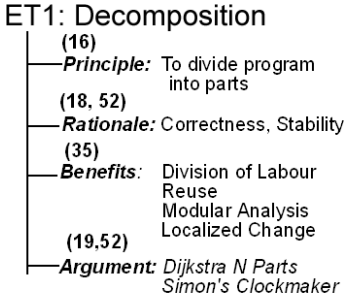

Fig. 4. Decomposition RIO as an instance of Enabling Technique in SWOLO

C. Design Strategy

The module also discusses the difference between top-down and parallel design strategies. As Fig. 5 shows, this particular module includes two instances of the Design Strategy RIO; Top-Down Design and Parallel-Design.

In SWOLO, the Learning Design for Design Strategy is specified by a *Process*, *Limitations* and an *Example*. Fig. 6 shows one such RIO abstracted from the module. The RIO about Top-Down design clearly incorporates the process of design itself and its limitations. In addition, splitting the software design for a Browser into *ReadCommand*, *GetPage* and *DisplayPage* is provided as an example of Top-Down design.

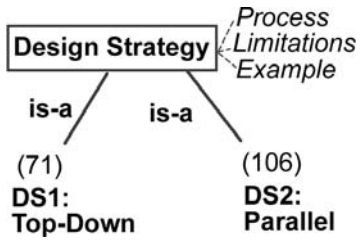

Fig. 5. Two instances of the Design Strategy RIO's in SWOLO

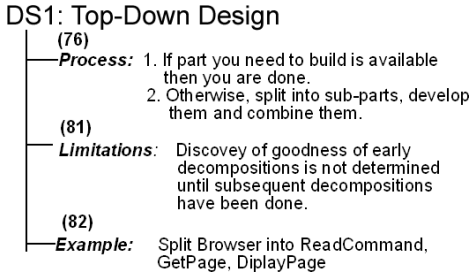

Fig. 6. An Instance of the Top-Down Design RIO in SWOLO

.

D. Design Notation

As Fig. 7 shows, the module discusses two instances of Design Notation; *Uses Diagrams* and *Dependency diagrams*. In SWOLO, the learning design for a design notation has a *Definition*, *Uses*, *Limitations* and an *Example*. A *Uses Diagram* can be Tree or Layered.

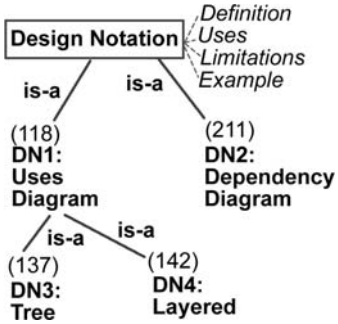

Fig. 7. Various instances of the Design Notation RIO in SWOLO

Fig. 8 shows various instances of the Design notation RIO. As Fig. 8 shows, Uses Diagram is defined to capture *uses* relationship between parts of a software design. This type of diagram can be used for reasoning, reuse and to determine the construction order in case of a change. A limitation of this type of a diagram is an explosion due to the transitive nature of the *uses* relationship. Finally, an example is provided to illustrate this particular type of design notation.

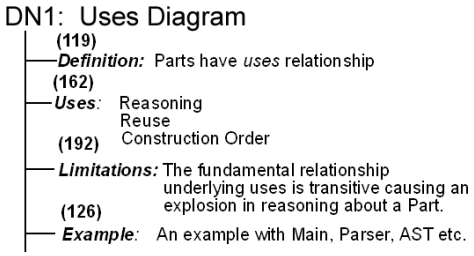

Fig. 8. Uses Diagram RIO as an instance of the Design Notation RIO in SWOLO

E. Software Structures and Architectures

As Fig. 9 shows, Decoupling Technique RIO is an instance of the Software Structures and Architectures (SSA). As Fig. 10 shows, in SWOLO, the learning design of an SSA RIO describes a common Solution to a common Problem in some Context. A Decoupling Technique solves the problem of minimizing the quantity and quality of dependencies. It does so by bringing together aspects of systems that belong together.

The module discusses four types of Decoupling Techniques; Façade, Hide Representation, Polymorphism and Callbacks. Fig. 11 shows the Learning Design for the Façade RIO. The Façade solves the problem of decoupling layers by introducing a new implementation part between two parts. In addition, layering of Protocol and Network layers in the design of an internet browser is provided as an example of Facade.

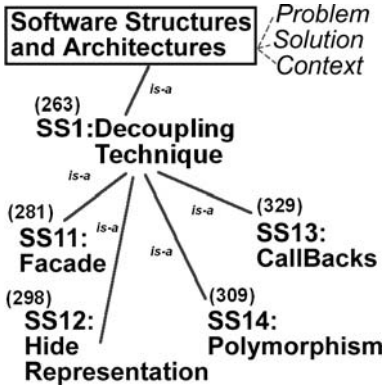

Fig. 9. Instances of the Software Structures and Architectures RIO in SWOLO

F. Software Design Quality and Evaluation

As Fig. 12 shows, Software Design Quality and Evaluation has two types of RIO; Quality Attributes and Quality Measures. *Coupling* is an instance of the Quality Attribute and *Dependence* is an instance of the Quality Measure. *Dependence* measures *Coupling*. There are two aspects of *Dependence*; quality and quantity.

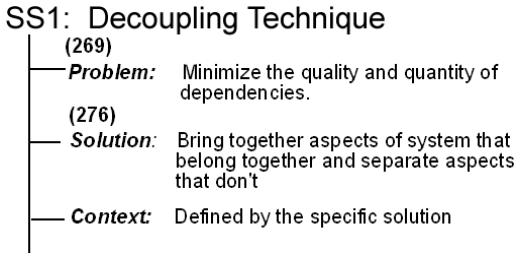

Fig. 10. Decoupling technique as an instance of the Software Structures and Architectures RIO in SWOLO

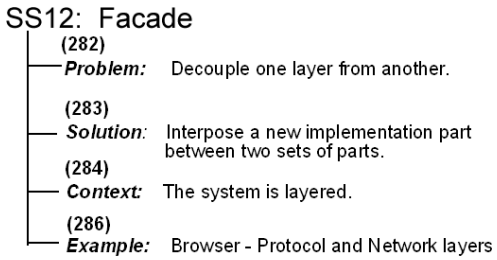

Fig. 11. The Façade RIO as an instance of the Decoupling Technique in SWOLO

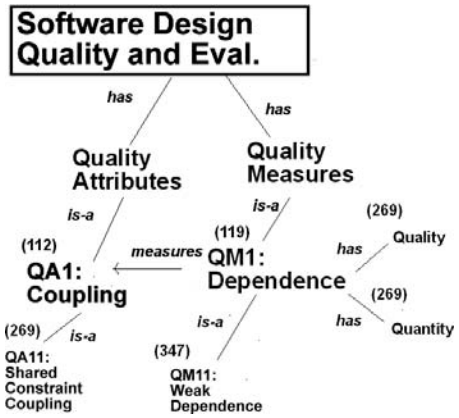

Fig.12. Various instances of Design Quality Attributes and Measures in SWOLO

V. Conclusion

This paper presented an extended form of Cisco's re-usable learning object design methodology that incorporates a new design ontology called SWOLO. SWOLO is based on SWEBOK. For an existing learning module in software design, the use of the new ontology led to a refined representation of what is being taught and the concepts in the design ontology were easily identified. In addition, unique Learning Design templates based on the software design ontology rather

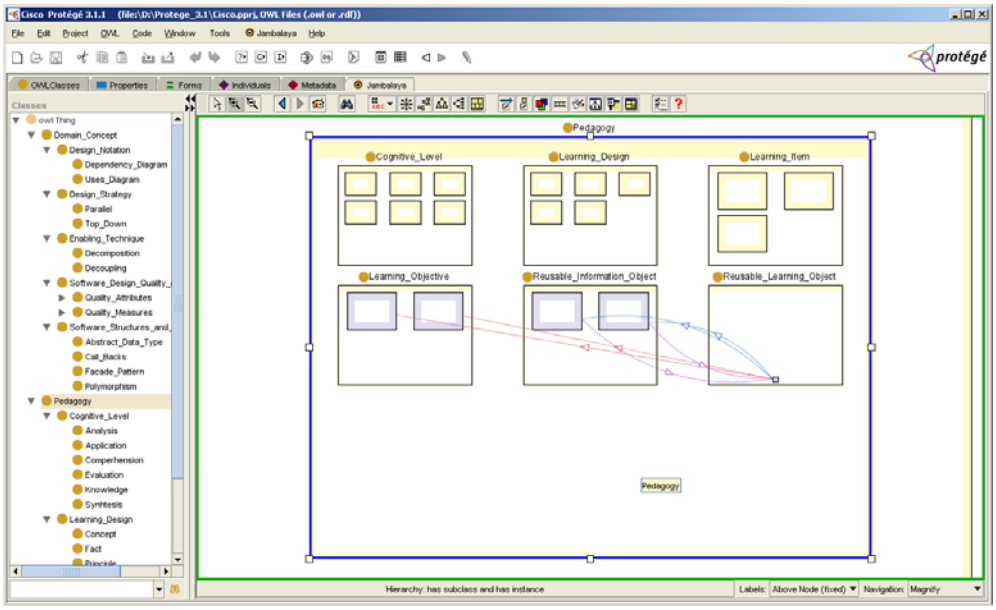

Figure 13. A prototype implementation of SWOLO Ontology in OWL using Protégé

than generic notions of a "Concept" or a "Process" were defined and identified. While not discussed in this paper, SWOLO also allows one to specify the Cognitive Level of each RIO at a finer level of detail than indicated in SWEBOK [5] (see Appendix D in [5]).

The concepts presented here can easily be extended to incorporate other components of the Software Engineering Body of Knowledge (e.g., see [8]). The ontology is currently being formalized using the Protégé toolset [9]. Figure 13 shows a prototype implementation showing the module discussed in this paper. Once the formalization of the ontology has been completed, an interesting direction may be the automatic annotation of existing learning objects in Software Design (e.g., see [10]).

Finally, this approach needs to be incorporated into existing learning object packaging and sequencing frameworks like SCORM [11]. One key contribution of this research is the introduction of Learning Design components based on the Software Design ontology. However, like Cisco's methodology, these Learning Design components are fairly primitive in that they simply specify "slots" for the various components of learning (e.g., *Problem*, *Solution* and *Context*). Of particular importance is a relationship to the emerging Learning Design Specification [12]. Therefore, rather than static slots, the Learning Design is being extended to specify a "grammar" of allowed learning processes for each component of the SWEBOK Design ontology.

REFERENCES

[1] "Reusable Learning Objects Authoring Guidelines: How to Build Modules, Lessons and Topics," *Cisco Systems*, Inc., 2003.

[2] D. Wiley, "Learning Object Design and Sequencing Theory," *Doctoral Dissertation, Brigham Young University*, 2000.

[3] www.reusability.org

[4] A, Abran, J. W. Moore, P. Bourque, and R. Dupuis, "Guide to the Software Engineering Body of Knowledge," 2004 Version, ed: IEEE Computer Society Press, 2004.

[5] A. Gomez-Perez, O. Corcho and M. Fernandez-Lopez, *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, First Edition, Springer, 2005.

[6] B. Bloom, "Taxonomy of Educational Objectives: The Classification of Educational Goals," Mackay, 1956.

[7] http://ocw.mit.edu

[8] C. Wille, R. Dumke, A. Abran, J. Desharnais, "E-Learning Infrastructure for Software Engineering Education: Steps in Ontology Modeling for SWEBOK," Proceedings of the IASTEAD International Conference on SOFTWARE ENGINEERING, Feb. 17-19, 2004, Innsbruck, Austria.

[9] "The Protégé Editor and Knowledge Acquisition System," (online: http://protege.stanford.edu/)

[10] J. Jovanovic and D. Gasvevic, "Ontology-Based Automatic Annotation of Learning Content," International Journal on Semantic Web & Information Systems, vol. 2, no. 2, 91-119, 2006.

[11] "SCORM 2004," Advanced Distributed Learning (online: http://www.adlnet.org/).

[12] "Learning Design Specification," IMS Global Learning Consortium, Inc. (online: http://www.imsglobal.org/learningdesign/index.htm