2006

An Ontology-Based Metamodel for Software Patterns

Scott Henninger
University of Nebraska - Lincoln, scotth@cse.unl.edu

Padmapriya Ashokkumar
University of Nebraska - Lincoln, ashokkum@cse.unl.edu

An Ontology-Based Metamodel for Software Patterns

Scott Henninger, Padmapriya Ashokkumar
Univ. of Nebraska-Lincoln
Computer Science and Eng.
Lincoln, NE 68588-0112
{ashokkum, scotth}@cse.unl.edu

1 Abstract

Patterns have been successfully used in software design to reuse proven solutions. But the complex interconnections and the number of pattern collections is becoming a barrier for identifying relevant patterns and pattern combinations for a given design context. More formal representations of patterns are needed that allow machine processing and the creation of systematic pattern languages that guide composition of patterns into coherent design solutions. In this paper, we present a technique based on Description Logic and Semantic Web technologies to address these problems. A metamodel is presented for developing pattern languages using this technology. Usability patterns are used to demonstrate how this metamodel can be instantiated to form a pattern language for that domain. Our technique provides a computational basis for building intelligent tools that utilize patterns, known best solutions to recurring problems, to support software development activities.

2 Introduction and Motivation

Software patterns represent knowledge successful solutions to recurring problems within contextual constraints [2, 12]. Patterns are increasingly being used to not only capture and disseminate best practices, but also to turn named patterns into a shared vocabulary for expressing and communicating technical knowledge [1, 20]. Beginning with the seminal Gang of Four book on design patterns [12], software patterns have expanded to many other domains, including software architecture [7], process [3], analysis [11], usability [6, 13], and product lines [8], to name just a few.

Pattern usage is currently practiced informally, often through folklore and textbooks, and at best being embedded in hypertext systems supporting semantic-free "related-to" relationships [13, 14]. But the continued proliferation and interconnected nature of patterns presents a number of problems that have not been adequately addressed. First is the sheer number of patterns available. One source states there are 250 patterns for Human-Computer Interaction alone [20], and still misses a number of usability pattern collections. Second, the interconnected nature of patterns makes it very difficult to understand the potential interactions, necessary couplings, contradictions, and inconsistencies amongst the different patterns. Used in an informal manner, this barrier is a constraint-based problem that people are particularly inept at understanding and applying. Third, there are a number of pattern collections currently in print or on Web-based resources. Not only do these collections differ in format, they also differ in emphasis, have overlapping content, and often contain contradictory information. Adding to the complexity is the fact that many patterns have multiple variants with subtle differences [9]. Fourth, each pattern is itself a piece of potentially complex knowledge that requires a degree of mastery [20] to apply to specific problem contexts.

These dimensions of scale, complexity, heterogeneity, and required expertise conspire to create barriers to the effective use of pattern technologies. Addressing these and other issues related to "pattern creep" will require increasing levels of formality and associated technologies to help people deal with the aforementioned issues. In particular, the development of pattern *languages* (as opposed to loosely coupled pattern collections [14]), which are systematic means of organizing patterns to provide holistic design solutions [1, 24] will require levels of formal representation that have yet to be applied to pattern technologies. There is therefore a need to formalize the representation of patterns and apply rigorous reasoning techniques to support the usage of patterns in the software development process.

The overall motivation of this research is to construct formal frameworks for pattern-based software development tools. A step in this direction is to represent patterns in formal ontologies representing not just pattern collections, but pattern languages – interconnected patterns that lead to a systematic solution of a given software development problem. The intent of these languages is to form an infrastructure for the development of intelligent tools that utilize patterns to develop cost effective software development solutions with high levels of quality.

To achieve this goal, technologies and solutions are needed that the process of constructing common vocabularies that unify the currently fractured state of pattern representations. We utilize ontologies, defined as a formal explicit specification of a shared conceptualization [23] within a domain of interest, to implement shared vocabularies that help represent and organize pattern languages. Specifically, the ontology implementation technology we utilize is the Web Ontology Language (OWL) [17] and Semantic Web technologies [4, 19]. OWL-DL is utilized to formally define patterns and relationships between patterns. OWL-DL is founded on decidable fragments of first order logic and axiomatic definitions that can be used by Reasoners to infer new facts and check the consistency of the resulting ontologies and infer new facts. In addition to the use of formal logic-based definitions, Semantic Web technologies utilize distributed representations of ontologies and various standards (W3C recommendations) that allow interoperability between distributed data stores.

In this paper, we describe a metamodel for describing patterns. Similar to the Meta-Object Facility (MOF), purpose of this metamodel is to provide the building blocks for developing pattern languages in specific domains. We demonstrate this process by using our metamodel to create a pattern language for usability design.

In the following, ontologies and the Web Ontology Language (OWL) are briefly introduced. This is followed by a presentation of out usability pattern metamodel and an example pattern language based on this metamodel. The paper concludes with a discussion of contributions, related work, and future directions.

3 Ontology-Based Approaches to Pattern Languages

The use of ontologies to represent pattern languages is a marriage of two complementary philosophies. An objective of pattern languages is to provide the means for professionals to use a common vocabulary about design and other issues. The Semantic Web and its technologies, such as OWL, implement techniques for formally defining ontologies through shared vocabularies, axiomatic definitions, and formal logic to further support machine-based automated reasoning. In the following, we briefly explain the reasoning capabilities of OWL and capabilities we use to create intelligent pattern languages.

3.1 OWL Description Logics

The Web Ontology Language (OWL) [17] builds on the Resource Description Framework (RDF) and RDF Schema [18] to create a frame-based knowledge representation language with axiomatic constructs for logic-based expressivity. OWL includes vocabulary for describing properties and classes. OWL constructs allow the construction of class taxonomies and properties act as predicates that represent an RDF triple between two classes. OWL builds on this infrastructure with richer typing of properties, relationships between classes (e.g. disjointness), class constructors (e.g. union, composition), enumerated classes, cardinality (e.g. “exactly one”), equality, and characteristics of properties (e.g. symmetry, transitivity. OWL also adds axiomatic constructs for greater expressiveness, including quantifications (e.g. existential, universal), qualified cardinalities, and general axiomatic definitions of class membership through complex expressions.

OWL properties are predicate that operate on subjects (domains) and map to an object (range). Range values can be restricted through various axiomatic class construction operators. For example, if you want to state that a Login pattern is the set of pattern instances that are a subclass of a UIAuthenticate pattern and all values of the hasWidgets property come from the TextBox and Button classes, then you would state (in DL form):

$$\text{Login} \sqsubseteq \text{UIAuthenticate} \sqcap \forall \text{hasWidgets}.(\text{TextBox} \sqcup \text{Button})$$

As part of the W3C standard, OWL requires a XML serialization of as a common representation that can be used across different OWL editors and reasoners. For this specific example, the XML is shown in Figure 1 (using the Protégé OWL ontology editor [22]), where the class Login is defined as an anonymous class consisting of the intersection between ‘UIAuthenticate’ and a restriction on the property ‘hasWidgets’ where all values (range values) are from either the ‘TextBox’ or ‘Button’ classes (union).

3.2 OWL and Pattern Languages

In addition to OWL DL, Semantic Web technologies offer a number of features that support creating an effective infrastructure for patterns that is capable of automated reasoning:

- **Distributed representations:** Since OWL is built on top of RDF and XML [15], Uniform Resource Indicators (URIs) are used to support common vocabulary in distributed files. A URI defines a unique namespace (using the same syntax as URLs) and concept within the namespace, thus assuring that two patterns using the same URI are referring to the same OWL element.
- **Well-defined semantics.** The description logic defined by OWL-DL allows precise definitions of concepts, as demonstrated in the example above. Patterns are therefore understood by both human and machine processable [5].

```
<owl:Class rdf:ID="Login">
  <rdfs:subClassOf>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
     <owl:Class rdf:ID="UIAuthenticate"/>
     <owl:Restriction>
      <owl:onProperty>
       <owl:ObjectProperty rdf:ID="hasWidgets"/>
      </owl:onProperty>
      <owl:allValuesFrom>
       <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:ID="TextBox"/>
         <owl:Class rdf: ID="Button"/>
        </owl:unionOf>
       </owl:Class>
      </owl:allValuesFrom>
     </owl:Restriction>
    </owl:intersectionOf>
   </owl:Class>
  </rdfs:subClassOf>
  </owl:Class>
```

Figure 1: XML Serialization of OWL-DL.

- **Automated reasoning:** Because patterns defined in OWL ontologies are formally defined, automated reasoning is possible that infers relationships between patterns and classifies pattern instances. In addition OWL reasoners support consistency checking based from formal axioms that help pattern language builders create well-defined and consistent ontologies.
- **Rule-based reasoning.** In addition to reasoning capabilities, rules can cover a wide range of behavior, such as specifying that under a specified set of circumstances that a rule should be applied. This additional expressiveness is enhanced by the formal and well-defined semantics of OWL.
- **Rule-based and semantic search.** In addition to specifying pattern attributes and relationships, rules can be used for precise specification of matching criteria. Intelligent search based on semantic relationships is also possible, enhancing the ability to find patterns that fit developer needs.
- **Heterogeneous representations.** A current problem with pattern collections is the wide range of attributes, descriptions, and terminology used to describe the patterns. While our metamodel (Section 4) is designed to facilitate more homogeneous representations, OWL axioms provide a number of constructs to establish the equivalence of concepts and properties as well as incremental differences between concepts in distributed pattern ontologies.
- **Web technology compatible:** W3C OWL requires XML serializations, making it compatible with existing Web standards such as http server, Web Services and can be detected and indexed by search engines.

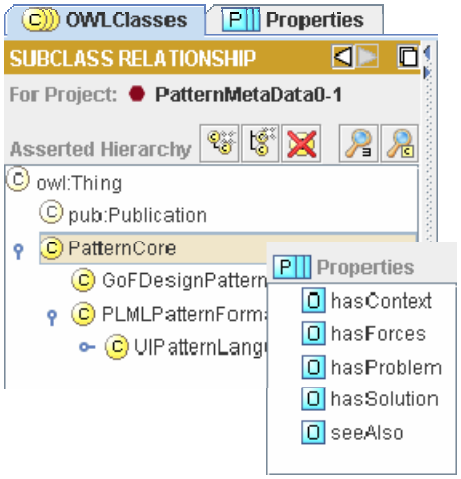

Figure 2: The Core Software Pattern Metamodel. Viewed in the Protégé OWL ontology editor.

4 Usability Design Pattern Metamodel

The purpose of a metamodel is to define the basic building blocks and rules for constructing well-formed models within some domain of interest [21]. A pattern metamodel will therefore provide the basic building blocks for creating types of patterns, or pattern languages. For example, Figure 2 shows the base (core) metamodel pattern. This metamodel states that all models need to have the 'hasContext', 'hasForces', 'hasProblem', 'hasSolution' and 'seeAlso' properties.

The metamodel for describing UI Design Patterns was derived from the Pattern Language Markup Language PLML [10]. Additional properties are included to help describe supporting evidence for a given pattern [16]. PLML was defined using XML DTD technology to describe user interface patterns. In the following, extension to PLML are described that take advantage of QOL features to create semantically meaningful pattern descriptions and relationships between patterns that facilitate computational reasoning.

A major weakness of current pattern representations is the lack of semantic or typed relationships between patterns. While most pattern representations will have a "related-to" [12] or "seeAlso" [10] link between patterns, this is little more than a type of hyperlink that carries no semantic meaning beyond "this pattern is related to that one".

The usability patterns metamodel (see Figure 3) builds on the PaternCore and PMLPatternFormat pattern metamodels (see ① in **Error! Reference source not found.**). The metamodel consists largely of defined pattern properties, see ②. The 'requiresPattern' property

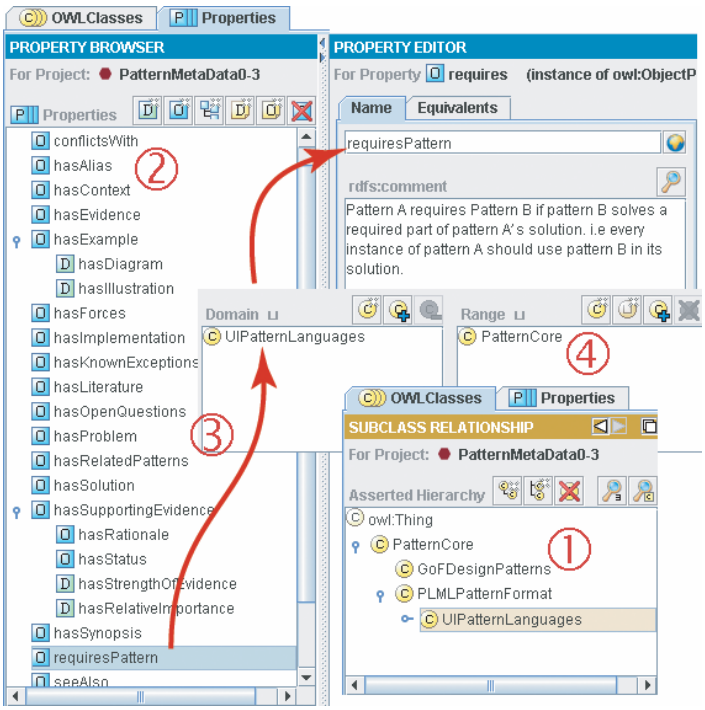

Figure 3: Property and Class Views of the Usability Pattern Metamodel.

is chosen, showing the pattern description and its Domain and Range (see ③ and the associated arrows). The purpose of requiresPattern is to provide a relationship between patterns when one pattern is required when another is used. As shown in Figure 3, a pattern of type UIPatternLanguage (the Domain) can require a pattern from any other pattern concept (the Range), represented by the root pattern concept, PatternCore (see ① and ④ in Figure 3). The range of values for this property can be further restricted for a more precise definition of the type of pattern needed (as demonstrated later).

The metamodel defines several types of pattern relationships with meanings specifically intended for the domain of usability patterns. These include:

- *uses*: Pattern A uses pattern B if the usage is optional In other words not every instance of pattern A uses pattern B, some variants do and some don't [26].
- *requires*: Pattern A requires Pattern B if pattern B solves a required part of pattern A's solution. I.e every instance of pattern A should use pattern B in its solution [26].
- *alternative*: Pattern A is an alternative to pattern B is the they have the same problem and context but different solution.
- *conflictsWith*: Pattern A conflicts With pattern B is they should not e used together in a design.
- *hasKnownExceptions*: Known contexts in which a pattern should not be used.

Some pattern annotations are applicable to certain pattern types but not for others. For example, the hasImplementation property is useful to describe design patterns but not usability patterns. These types of pattern annotations are accommodated in the metamodel by using OWL domain construct. The domain of the property is defined to be the most general class to which it is applicable. For example, hasContext is defined for the class Software Patterns as all Software Patterns have a context in which they are applicable whereas hasImplementation is defined for the class of Design Patterns.

4.1 Metadata Attributes

In addition to metadata attributes such as name, author, version and others, the following attributes are most useful for describing patterns and relationships between patterns.

- *hasProblem*: describes the need of an actor (person/system) for which the pattern was designed. For example, the Problem of an Ecommerce Website is to provide online users a virtual shopping experience. The actor here is the Ecommerce Website owner (company/individual).
- *hasSolution:* provides a set of instructions to solve the problem described in the Problem attribute.
- *hasContext:* a set of "applicable" design conditions in which the pattern can be most usefully ("naturally") applied
- *hasRationale:* provides principled reasons from behavioral psychology, cognitive psychology or another science for why the solution is effective.
- *hasForces:* discusses constraints and tradeoffs in choosing the solution suggested in the pattern.

4.2 Using the Metamodel

The domain and range definitions of the metamodel properties are defined to be as general as possible. This allows other models to adapt the metamodel elements for specific needs. For example, suppose a pattern language was being built for Web-based browser interfaces. When creating an instance of the metamodel for this purpose, the designers could restrict the range of 'requiresPattern' to include only Web-based interfaces.

Even after creating a specialized metamodel, additional restrictions, such as the 'hasWidgets' example in Section 3.1 can be used to refine the semantic definition of a given pattern relationship. Suppose one wants to represent the concept that when a large hierarchical Web site is being developed, any interface that uses the "SelectAction" pattern must use either the "Bread

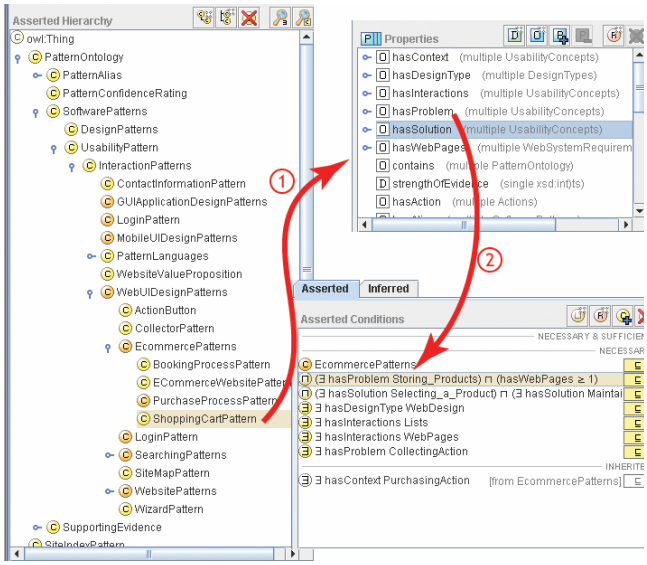

Figure 4: OWL description of a usability design pattern.

crumbs" pattern or a combination of the "Double tab" pattern and the "Main Navigation" pattern [25]. This means using the SelectAction pattern could be defined by the range restriction:

$$\text{SelectAction} \sqsubseteq \exists \text{requiresProperty}(\text{BreadCrumbs} \sqcup (\text{DoubleTab} \sqcap \text{MainNavigation}))$$

This means there exists at least one 'requiresPattern' property with a pattern from the BreadCrumbs or the DoubleTab and MainNavigation patterns. In this manner, the selection of one pattern can lead directly to the suggested use of another and so on until a partial solution to the user interface is found through a composition of patterns representing best practices for user interface design.

5 A Usability Pattern Instance

Figure 4 shows part of an instance of the usability pattern metamodel specifically designed for Web-based e-commerce. This shows a "Shopping Cart" pattern as a solution to the problem of storing products from multiple web pages that a user has selected items from. This is represented by the statement in the line pointed to by arrow ② with the property restriction ($\exists$hasProblem Storing_Products) $\sqcap$ (hasWebPages $\geq$ 1). This describes the problem though a combination of existential quantification on the hasProblem property and a cardinality restriction on the number of Web pages in the problem statement (hasWebPages).

The ShoppingCartPattern utilizes many of the properties from the metamodel (a partial list is shown by the arrow marked ①), including the hasSolution property. Following arrow ② reveals that in addition to the metamodel definition for hasProblem, the pattern uses a number of other properties to define the pattern and requirements on potential solutions.

6 Reasoning With OWL Models

Describing pattern attributes using this formal medium facilitates automated inferences of relationships between patterns instead of manually defining them as is done in text based pattern languages. For example, any pattern that has the same problem as the Shopping Cart pattern can be inferred to be an alternative pattern as long as the contexts are same and the solutions are different. Further, if these restrictions are stated as Necessary and Sufficient conditions, a Reasoner can infer classification. Although not shown here, if the restriction "$\exists$hasDesignType WebDesign" were stated as the only Necessary and Sufficient condition for membership in the ShoppingCartPattern concept, then all new patterns specifying one or more relationships of type WebDesign would be inferred to be a Shopping Cart Pattern. More complex restrictions can be used to precisely define class membership as deeply as deemed necessary by ontology designers.

7 Conclusions and Future Work

Design patterns serve a twofold purpose of providing a structured representation of best practices and supplying a shared vocabulary to express and communicate design knowledge. In this paper, we have shown how formally defined ontologies using the Semantic Web can be used to support these dual purposes in a distributed computational environment capable of automated axiomatic reasoning.

The general goals of this approach are to create a formal framework for creating interconnected pattern languages for usability knowledge. The metamodel described in this paper takes a step in that direction by creating an infrastructure upon which pattern languages and pattern-based tools can be built.

While a main goal of patterns is to form a vocabulary that helps developers communicate better, too many pattern collections have been created that draw little or no relationships between each other, in essence creating islands of patterns that sometimes contradict, duplicate, or are inconsistent with one another. We see context as one of the main organizing features of patterns. Usability issues and decisions are often, if not always, context-sensitive. Capturing this context is just the first step at making usability design a more stable or scientific endeavor.

Continued research is needed to further understand the complexities of creating repositories of usability patterns and applying them proactively in the software development process. We have only taken small steps in this direction, and hope that future validation and use of our approach provide more information of usability

knowledge and the contextual factors that impact this knowledge.

8 References

[1] C. Alexander, "The Origins of Pattern Theory: the Future of the Theory, and the Generation of a Living World," OOPSLA 1996 Keynote Address, http://www.patternlanguage.com/archive/ieee/ieeetext.htm, 1996.

[2] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, 1977.

[3] S. Ambler, Process Patterns: Building Large-Scale Systems Using Object Technology. Cambridge University Press, 1998.

[4] T. Berners-Lee, "Semantic Web Roadmap," W3C Semantic Web Vision Statement, http://www.w3.org/DesignIssues/Semantic.html, 1998.

[5] T. Berners-Lee, J. Hendler, O. Lassila, "The Semantic Web," Scientific American, May 2001, 2001, on-line at: http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2.

[6] J. Borchers, A Pattern Approach to Interaction Design. John Wiley & Sons, 2001.

[7] F. Buschmann, R. Meuiner, H. Rohnert, P. Sommerlad, M. Stal, Pattern-Oriented Software Architecture. John Wiley Sons, 1996.

[8] P. Clements, L. M. Northrop, Software Product Lines: Practices and Patterns. Addison-Wesley, 2001.

[9] J. Dietrich, C. Elgar, "Towards a Web of Patterns," Proc. Semantic Web Enabled Software Engineering (SWESE), 117-132, Galway, Ireland, 2005.

[10] S. Fincher, "Perspectives on HCI patterns: concepts and tools (introducing PLML)," Interfaces, 56, pp. 26-28, 2003, on-line at: http://www.bcs-hci.org.uk/interfaces.html.

[11] M. Fowler, Analysis Patterns. Addison-Wesley, 1997.

[12] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.

[13] S. Henninger, "An Organizational Learning Method for Applying Usability Guidelines and Patterns," in Engineering for Human-Computer Interaction (revised papers, EHCI 2001), vol. LNCS 2254, Springer, 2001, pp. 141-155.

[14] S. Henninger, P. Ashokkumar, "An Ontology-Based Infrastructure for Usability Design Patterns," Proc. Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland, pp. 41-55, 2005.

[15] M. Klein, "XML, RDF, and Relatives," IEEE Intelligent Systems, 15(2), pp. 26-28, 2001.

[16] S. J. Koyanl, R. W. Bailey, J. R. Nall, "Research-Based Web Design & Usability Guidelines," Communications Technology Branch, National Cancer Institute & US Dept of health and Human Services, http://www.usability.gov/pdfs/guidelines.html, 2003.

[17] D. L. McGuinness, F. van Harmelen, "OWL Web Ontology Language Overview," W3 Consortium, http://www.w3.org/TR/owl-features/, Last Updated February 10, 2004.

[18] E. Miller, R. Swick, D. Brickley, "Resource Description Framework (RDF)," W3 Consortium, http://www.w3.org/RDF/, Last Updated March 9, 2006.

[19] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, G. Schreiber, D. Wood, D. Connolly, "W3C Semantic Web," World-Wide Web Consortium, http://www.w3.org/2001/sw/, Last Updated March 15, 2006.

[20] S. Montero, P. Díaz, I. Aedo, "A Semantic Representation for Domain-Specific Patterns," in Int'l Symp. on Metainformatics, U. K. Wiil, Ed., Springer-Verlag, LNCS 3511, 2005, pp. 129-140.

[21] OMG, "MetaObjectFacility(MOF) Specification, v1.4," Object Management Group, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF, Last Updated April, 2002.

[22] Stanford Univ., "Protégé Project," Stanford Medical Informatics, http://protege.stanford.edu/, Last Updated March 9, 2006.

[23] R. Studer, V. R. Benjamins, D. Fensel, "Knowledge Engineering: Principles and Methods," Data and Knowledge Engineering, 25, pp. 161-197, 1998.

[24] J. Tidwell, "The Gang of Four are Guilty," http://www.mit.edu/~jtidwell/gof_are_guilty.html, 1999.

[25] M. van Welie, "Patterns in Interaction Design," http://www.welie.com/, Last Updated 05-25-2005, 2005.

[26] W. Zimmer, "Relationships Between Fesign Patterns," in Pattern Languages of Program Design, J. O. Coplien, D. C. Schmidt, Eds., 1995.