Adapting Software Engineering Design Patterns for Ontology Construction

STANISLAV USTYMENKO and DANIEL SCHWARTZ
Department of Computer Science
Florida State University
Tallahassee, FL 32306-4530
USA
ustymenk@cs.fsu.edu, schwartz@cs.fsu.edu

Abstract: - In this paper, we present an argument for designing metadata schemata with design patterns. Design patterns are structured descriptions of solutions to some class of problems, and are used extensively in various stages of object-oriented software engineering. We present a use case of collaborative construction of metadata for a digital library. We explore design challenges this scenario presents and then adapt a pattern called **Composite** from a standard software engineering design patterns reference to address parts of these challenges. Additionally, we propose a new design pattern called **History** suggested by a collaborative metadata construction scenario and applicable to a wider class of problems in metadata design.

Key-Words: - Design Patterns, Knowledge Engineering, Object-Oriented Design, Semantic Metadata, Web Ontology

1 Introduction

Efforts to standardize semantic languages gave metadata engineers direct access to modeling paradigms at higher levels of abstraction. Metadata schema engineering can borrow extensively from a now relatively mature field of object-oriented (OO) software engineering (SE) [9], since it is in many ways analogous to the domain modeling activity of object-oriented SE processes. This is captured by the extensive use of the Unified Modeling Language (UML) in communications among developers of the newly emerging Semantic Web [17, 18], as well as by the ongoing effort to map out Web Ontology Language (OWL) semantics in UML [2, 4, 10]. The UML and object-oriented metaphors are seen as both useful tools in metadata schemata and semantics vocabulary engineering and ways to utilize skill sets already acquired by an army of software engineers.

Object-oriented metadata techniques found acceptance in many research communities in Information Technology. A prime example of such acceptance arises in the Learning Systems field and, particularly, the Learning Objects (LO) community, concerned with building software for use in education. Here, the interest in object orientation is fueled by promise of modularity and reuse (cf. [12]). Several metadata schema standards have been developed [13, 14]. Some authors raise justified objections regarding the appropriateness of the object oriented technologies as a proper model of learning technologies [15], citing redundancy and the vague analogies used. Nevertheless, the basic notion of an object is a useful abstraction.

Our paper discusses metadata schemata not limited to learning objects and concentrates on the use of object-oriented patterns as a design technique, leaving appropriateness of the model to further research, as a particular application domain might dictate. A preliminary version of this work was presented as [20].

The object-oriented design community had adopted a concept of **design patterns**, borrowed from architectural design [1]. Design patterns are semi-formal, systematic descriptions of the solutions for common design problems. They show a particular commonly occurring problem in a specific context and then describe a proven method for solving such a problem.

The goal of patterns is to increase the quality of design, namely maintainability and extensibility, by communicating best industry practices. Among commonly used patterns in object-oriented design are “Model-View-Controller” for flexible user interfaces, and “Factory”, which suggests encapsulating object creation to a special class to avoid relying on the type information when creating objects in runtime.

One of the best known catalogues of patterns is the book [3] by Gamma, Helm, Johnson, and Vlissides. Often referred to as the “Gang of Four book” (or GoF), this became a common starting point when discussing patterns. GoF contains 23 popular patterns used in object-oriented software design.

Following is a short description of the structural pattern Composite from GoF, which we adapt below to our metadata schema needs as shown in Figure 1. In this figure, we use static model UML notation. Arrow connectors denote inheritance, while diamond connectors denote aggregation.

Name: ***Composite***

Task description: Supports tree-like structures, where an object representing a group of objects can be treated like an individual object. A classic example is the UNIX file system, where file directories are treated as files.

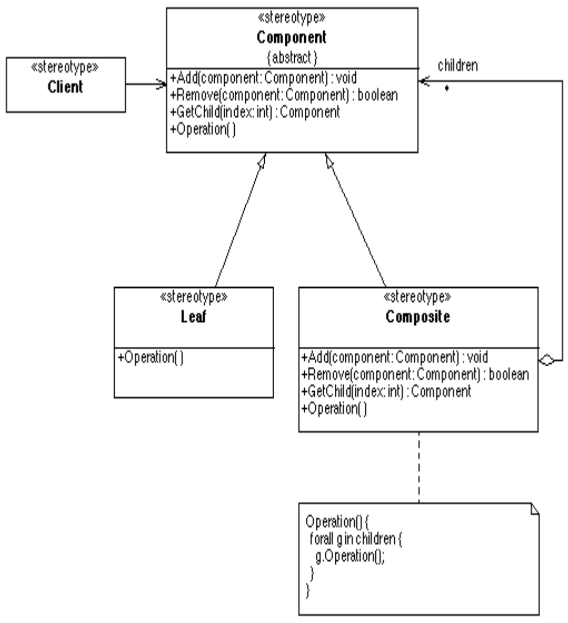
Context:
- In many applications, tree-like structures arise, where groups of elements can be elements of the bigger structure.
- All element objects have similar behaviour. As an example, consider a typical graphical user interface. All elements are rectangular areas drawn on the screen, and they all respond to user generated events. The same is true for frames and windows.

Participating classes:
- **Component**: A common superclass for Leafs and Composite, having an interface that allows an object to be a component of the Composite.
- **Leaf**: Subclass of Component representing simple objects.
- **Composite**: A Component that .contains collection of other Components. Thanks to polymorphism, both Leaves and Composites can be components of a Composite object.

Results:
- It is easier to add new types of components.
- A client program's architecture is simplified: the same logic can work with simple and composite objects.
- Any client that works with simple objects can also work with composites.

Related patterns: The GoF patterns Decorator, Flyweight and Iterator.

Metadata schemas, like software in general, have two facets. First, they are meant to naturally represent those aspects of a specific domain that are important for users. Second, they are dictated by the architecture of the software system they are part of and constrained by various technological requirements. Design patterns are meant to address the second facet, addressing nonfunctional requirements like extensibility, maintainability and performance [21]. Ontology languages, namely OWL, present a class of nonfunctional requirements related to reasoning and decidability, addressed by patterns not discussed here.


«stereotype»
Client
«stereotype»
Component
{abstract}
+Add(component:Component) : void
+Remove(component:Component) : boolean
+GetChild(index:int) : Component
+Operation( )
children
*
«stereotype»
Leaf
+Operation( )
«stereotype»
Composite
+Add(component:Component) : void
+Remove(component:Component) : boolean
+GetChild(index:int) : Component
+Operation( )

```
Operation() {
 forall g in children {
   g.Operation();
 }
}
```

Figure 1. Composite pattern.

A typical pattern includes the following elements: (1) name, (2) context, (3) task description, (4) solution description, and (5) results. Examples appear in the following Section 3. Pattern catalogs, like the GoF, organize patterns according to various categories, creating a valuable resource for both novice and experienced practitioners.

There are already examples of design patterns for semantic metadata creation. The Semantic Web Business Practices task force of the W3C has created documents to describe different approaches to using classes as property values and representing n-ary relationships in OWL and the Resource Description Framework Semantics (RDFS) [7, 8]. These documents state problems and provide several alternative solutions, clearly describing the consequences of using them. Thus they provide excellent examples of metadata design patterns. These examples address rather low-level, common problems and rely on specific features of RDFS and OWL. They fall into the design patterns category often called **language idioms.**

This paper concentrates on higher-level, language-independent design patterns. We present a realistic use case of semantic metadata in the context of a collaborative information system we are developing. The metadata schema developed through the course of this paper is intended to demonstrate this concept. Completeness and soundness were sacrificed for simplicity where necessary.

2 Case Study: Managing Knowledge Structures for a Digital Archive

2.1 Background

Indexing information in a digital archive requires extensive metadata schemas, enabling accurate categorization for navigation and search. Such a metadata system would include concepts organized in a taxonomic hierarchy, thesaurus, or more complex knowledge structure utilizing ontology semantics. One solution has been proposed to this end, namely SKOS [16], an RDF [11] vocabulary for publishing taxonomies.

One obstacle for utilizing ontology schemas for large and/or evolving archives, such as digital repositories of research papers [6], is defining an ontology of concepts to satisfy the needs of the majority of users, where such users might have conflicting perspectives on overlapping areas of inquiry. Ideally, the metadata created should reflect perspectives of different groups while spanning all the content, recognizing links between alternative conceptualizations. At the same time the system should maximize the quality of the ontology, keeping inconsistencies to a minimum.

It is our position that this task is best left to the evolving community of users to carry out. A Web-based system is to be created that allows users, individually and through collaboration, to define ontologies to classify documents in a collection. Users will form online communities (see, e.g., [22, 23]) around domains of interest, contributing to ontology engineering through discussions and collaborative editing. The resulting ontologies then can be combined on overlapping concepts and used to improve search in the repository. A user's association with ontologies will provide the context necessary for personalizing his or her search and browsing experience, identifying items of particular interest for the user. The collaborative environment can also be used to create communities of practice, supporting informal knowledge exchange.

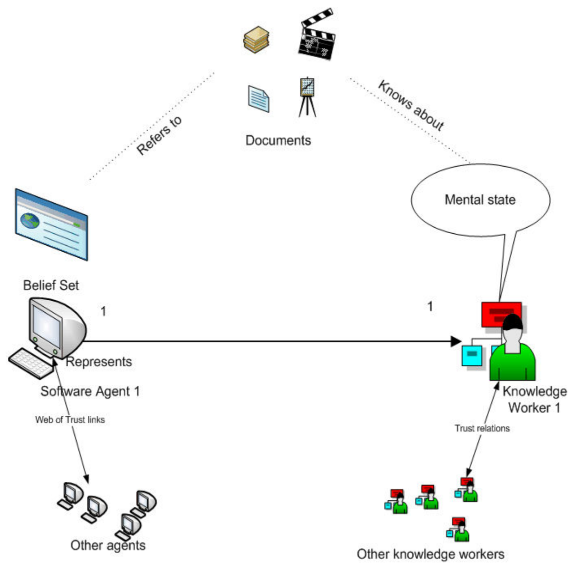


Figure 2. System overview.

2.2 System overview

The overall system is comprised of automated agents that act on behalf of the human users. Software agents are broadly defined as software entities capable of obtaining information from their environment, reacting to it, communicating with other agents, and achieving a goal of performing some complex task on behalf of their users. To perform this task, agents must acquire an understanding of their environment, and, to effectively communicate, they must subscribe to a common language with agreed upon semantics. This is illustrated in Figure 2.

The main actors in the system are human users and computational (software) agents. Each agent represents one user, and each user has one and only one agent (unless a user has a reason to have two or more agents with different beliefs). Software agents interact within an artificial network of trust relations (the web of trust) that approximate attitudes each user has towards other users.

Similarly, we introduce information artifacts agents that manipulate documents, topic categories and relations between them. A belief set of an agent aims to approximate the user's epistemic attitudes (knowledge and belief) towards these artifacts.

In our model, each agent represents one user of the system. Agents act as both information suppliers and consumers. As a consumer, an agent's goal is to find documents relevant to a user's interests. As an information supplier, agents make their own documents available for other agents.

We assume that it is in the best interests of the human users to make their documents easily searchable, and that they will provide rich metadata for this purpose.

The users of our system are knowledge workers. They engage in consuming, producing and disseminating knowledge. The tangible unit of knowledge is called a document. There is no requirement that the document is in fact a text-based file available electronically. A paper, a book, a movie, painting or sculpture can be considered a document. As far as the information system is concerned, a document is anything that can be uniquely identified and referenced and is treated as an atomic informational entity. For the remainder of this paper, we will also use the term "document" for document records.

The user has mental attitudes towards documents and other users. We refer to these attitudes as knowledge, belief, and trust. Mental attitudes change over time. We refer to the set of such attitudes at a given time as the user's mental state. References to documents a user is aware of, as well as a topic taxonomy that reflects the user's beliefs, are included in his/her mental state.

Users move from one mental state to another by performing cognitive actions: they can either acquire a new belief from another trusted individual or derive it through some inference step. We can think of these actions as occurring sequentially. An agent's inference activity can be represented as a finite set of discrete time steps, each associated with some cognitive action.

To facilitate search and browsing, a user associates documents with categories or topics: sets of documents that share some common theme. He creates a taxonomy by defining relations between topics. The most common type of relation is the subtype-supertype relation.

A software agent models mental states through its *belief set*. A belief set is a set of statements in some formal language, designed to express mental attitudes towards facts. In [19], we developed a formal logic satisfying this requirement. We can identify three kinds of beliefs that need to be represented in a belief set:

- Topic taxonomy, expressed as a set of logic-based statements with associated belief attitudes.
- Set of document references, together with statements linking documents and topics.
- Set of statements reflecting the agent's trust relationships with other agents.

At any given time, a human mind can harbor inconsistent beliefs. Inconsistency might not be apparent until, through cognitive actions, a person arrives at an absurd conclusion. When this occurs, a rational thinker re-examines his beliefs, rejecting those that are less supported or otherwise less entrenched. The artificial mind and belief set of an agent should be able to support such behavior.

On each time step, an agent is free to choose a cognitive action among legal inference rules or to acquire beliefs from other agents' beliefs. In effect, an agent adds to or modifies his belief set. This choice is guided by considerations not completely determined by rules of logical inference, namely, the agent's goals and desires. An agent's actions include the following:

- Responding to queries for documents.
- Exposing a new document created or discovered by the user. This includes classifying the document into one or more of the taxonomic categories. This action introduces a new named individual into the domain description, effectively expanding the language.
- Getting a previously unknown document into the agent's belief set. The new document is among documents exposed by some other agent.
- Discovering facts about taxonomic classification of a particular document. An agent requests taxonomic facts from other agents, with the goal to create a complete description of the document. In this process, the agent modifies its taxonomic beliefs.
- Inferring new beliefs from previously acquired beliefs. This action may be triggered by any of the previous actions.
- Modifying currently held beliefs to remove a contradiction.

2.3 Domain Model

The internal metadata schema must support dynamic evolution of the taxonomy in the community, means to track changes and user or group ownership for the individual pieces of metadata as well as larger structures (thesauri or ontologies), and methods for merging/combining concepts and relationships from individual and group ontologies. Common access and ease of modification can enable consensus building in the communities, while support for pluralism (i.e., different perspectives) helps to serve everyone's needs in the best way possible.

Types of information a proposed metadata schema must reflect include:

- Concepts
- Concept labels
- Relationships

- Taxonomies/Thesauri
- Users
- User groups

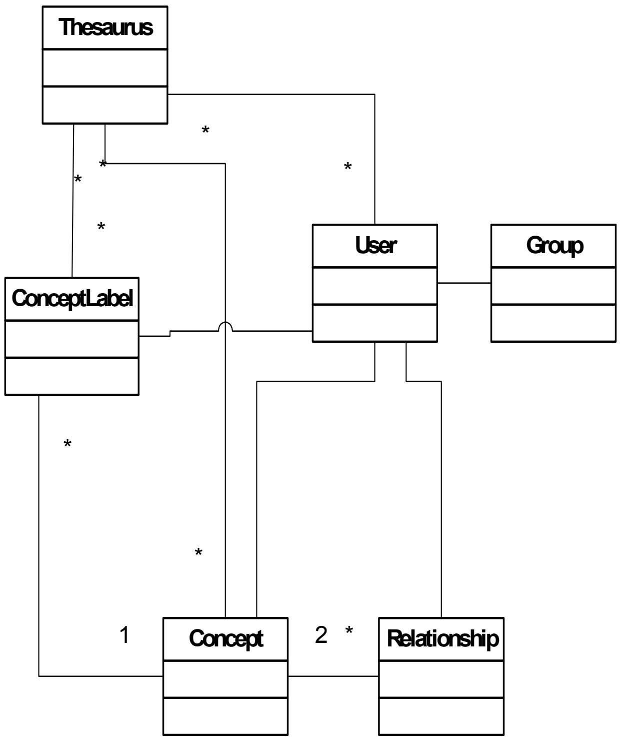


Figure 3. Initial domain model.

A UML diagram illustrating the model is shown in Figure 3. Select object-oriented design patterns can be applied to this model to help achieve the stated requirements. Those patterns can be chosen from existing catalogs (e.g., GoF) and adapted to accommodate metadata language idioms. This adaptation entails making minor changes that reflect the circumstance that metadata engineering has slightly different goals from software engineering. For example, in object-oriented software engineering, objects typically have associated methods, whereas in metadata engineering, they don't.

Our goal is to create a schema that supports this vocabulary and allows for a straightforward physical implementation using mature technology (e.g., relational database, XML storage, or ontology management system). From the previous discussion we can derive two major requirements for the model of the bibliographic domain:

- Personification. Knowledge elements must be explicitly associated with agents. Our data model contains all the knowledge of the multiagent system, so that every agent has access to the knowledge of other agents. At the other hand, each individual knowledge base (or belief set) is clearly identified.
- Dynamics. The belief revision mechanisms mentioned above rely on explicit representations of the agent's belief evolution in time.

.

Our domain model implements these requirements by explicitly representing users (agents) and by adding a timestamp to each taxonomy element. Thus, all historic data is always present in the system. All knowledge elements are identified both by the timestamp and the user.

At any given time, different users might have different views to the same model. The view is affected by the user's group membership and relationships between groups. This allows for great flexibility and is a great asset in a collaborative environment where global consensus is unlikely or undesirable. (We assume this is true for the bibliographic system we describe).

The model described above adequately reflects the domain and can be used when developing the software implementation. Direct mapping from objects to database tables facilitates the design of object-to-relational mappings and thus the initial development process can be relatively straightforward. However, several potential problems can be readily identified:

- Adding new types of statements, classes of individuals, and relationships require redesign of a database scheme.
- Developers may want to allow a user to perform certain common operations with domain objects, such as create, delete, share with others, and combine in taxonomies.

It is conceivable that a taxonomy can have other taxonomies as parts, though the practicality of such an approach demands additional investigation. Objects are subject to change tracking and permission management, and in this regard there is no difference between Documents, Statements, and Taxonomies.

3 Metadata Design Patterns

3.1 Pattern "Composite"

The ***Composite*** pattern is a structural pattern from the GoF book. It composes objects into tree-like structures for representing part-whole hierarchies, while allowing uniform treatment for both atomic and composite objects. GoF Chapter 4 defines the Composite pattern as follows:

Name: ***Composite***

Task description: Support composing objects into tree-like structures for representing Part-Whole relationship. It allows clients to uniformly treat individual and composite objects.

Context:

- Representation of a part-whole hierarchy is required.
- The same services must be supported by both atomic and complex objects.

Participating classes:

- **Component:**
  - Declares common interface for objects.
  - Contains appropriate default method definitions, common for all classes.
- **List:**
  - Represents leaf nodes of the hierarchy and has no child nodes.
  - Defines primitive nodes' behavior.
- **Composite:**
  - Defines behavior for objects that contain other objects.
  - Stores (links to) child components.
  - Implements functionality specific to child node manipulation.
- **Client:**
  - Manages objects through the **Component** interface.

Results:

- A hierarchy containing both primitive and composite objects.
- Client architecture is simplified. Clients can work with individual and composite objects in the same way, thus simplifying the client code.
- Maintainability is enhanced by simplifying the task of adding new Component classes. The client code that works with existing components will need little or no modification to support new subclasses of type **Leaf** or **Composite**.
- Common design is enforced.
- It is difficult to define constraints on the type of objects that can be parts of a composite. Sometimes, it is beneficial to only allow specific classes of Leafs for a given Composite.

Related patterns: Decorator, Adapter, Iterator, Visitor, Chain of Responsibility.

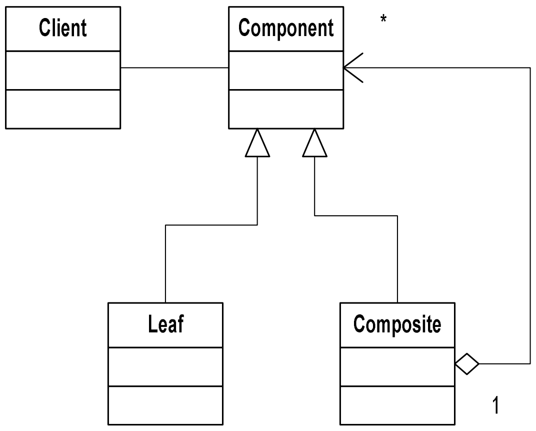


Figure 4. Composite pattern**.**

Objects that form a part-whole hierarchy in the digital libraries application domain include concepts, relationships, labels, and taxonomies. We want to allow a user to perform certain common operations with such objects: create, delete, share with others, and combine to form taxonomies. It is conceivable that a taxonomy can have other taxonomies as parts, though practicality of such an approach demands additional investigation. Objects are subject to change tracking and permission management, and in this regard there is no difference between simple objects like Concept Link and containers like Taxonomy.

.

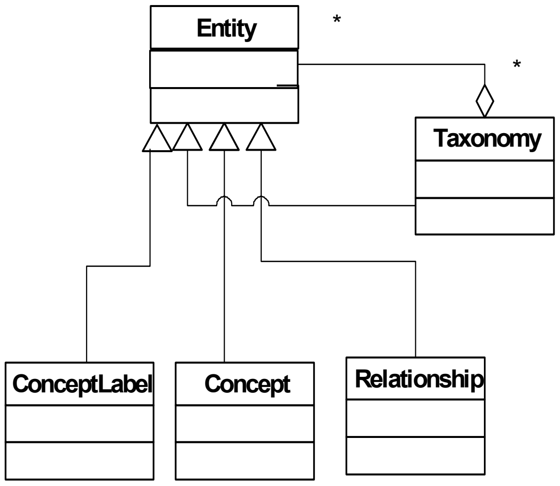


Figure 5. Composite pattern unifies classes through common superclass**.**

To facilitate uniform treatment, the pattern recommends defining a common superclass, Entity, for the domain classes involved. Concepts, Relationships and Labels are related appropriately. A thesaurus, in essence, is defined as a collection of

Entities. Using the vocabulary of the Composite pattern as used in the GoF book, Entity is the **Component** class, Concepts, Relationships and Labels are **Leafs**, and Taxonomy is the **Composite.** It is the responsibility of an Entity to define data for common functionality like versioning, although obviously it does not define a message interface (as metadata are not object-oriented software components and do not define behavior).

3.2. Pattern "History"

This pattern is new and is being introduced here for the purposes of the digital libraries application. Evolution of the metadata in a digital library scenario involves numerous parties modifying the conceptual structure (taxonomy and documents being classified). This involves tracking those changes and allowing undo capability. A user may choose to only allow his model to be editable by certain individuals, thus filtering out changes made by others. In general, our system calls for multiple versions of metadata. We want to define the simplest architecture possible to easily track those changes.

This can be achieved by introducing the new class we call **Event**. It decouples changes from the entities to which the changes are being made and allows managing changes as first-class objects. The pattern is defined as follows:

Name: ***History***

Task description: Support versioning and change management in metadata.

Context:
- Changes occur by inserting and deleting elements of metadata describing the domain or changing an element's content (e.g., its textual description).
- Metadata consists of atomic elements. Taxonomy data (the nodes and links) are the characteristic example.
- Different versions might co-exist.

Participating classes:
- **Entity:** Metadata element representing an atomic piece of metadata to manage. In the context of our application, an Entity might represent a concept, relationship, or label.
- **Subject:** Metadata element representing a user-entity (e.g., an individual user or a group) initiating the change.
- **HistoryEvent:** Metadata element representing a change that has occurred to the model.

Results:
- Versions are defined both by the timestamp and the user. At any given time, different users might have different views of the same model. The view is affected by the user's group membership and relationships between groups. This allows for great flexibility and is a useful asset in a collaborative environment where global consensus is unlikely or undesirable. (We assume this to be true for the bibliographic system we describe.)
- Events are managed as first-order objects.
- Querying for versions can become computationally expensive.
- While the schema adds very little in complexity, actual metadata can become difficult for humans to analyze. This can become a problem where human readability is a requirement. This problem can be alleviated by providing appropriate utility software.

Related patterns: The GoF pattern ***Command*** is close to ***History*** in structure and intent, but it focuses on behavior aspects not relevant to metadata. Unlike ***History, Command*** does not address collaboration and multiple simultaneous versions.

The model for the ***History*** pattern is depicted in Figure 6.

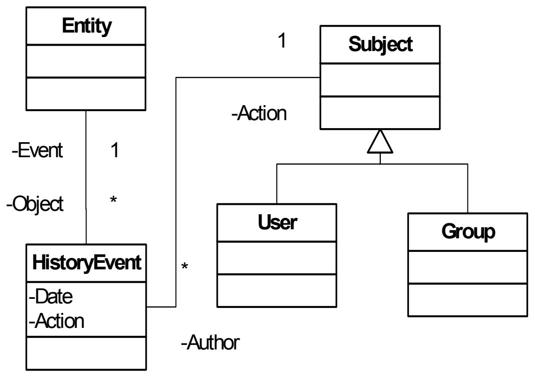


Figure 6. History pattern enables collaboration.

4 Applying Patterns and Implementing Software

The described patterns are combined, resulting in the metadata schema depicted in Figure 5. Note how the two patterns complement each other in this design: the ***Composite*** pattern allows treating taxonomy elements uniformly, while ***History*** solves change management issues for unified metadata pieces. Each pattern facilitates the other in achieving its objectives.

This schema supports a dynamic metadata management system. Metadata is stored in a relational database. Classes are implemented as tables. Inheritance is implemented as one-to-one relationships. Stored procedures are used for querying Taxonomy classes, allowing the merging of the Entity table with appropriate subclass tables (in database parlance, *select* statements with join operations).

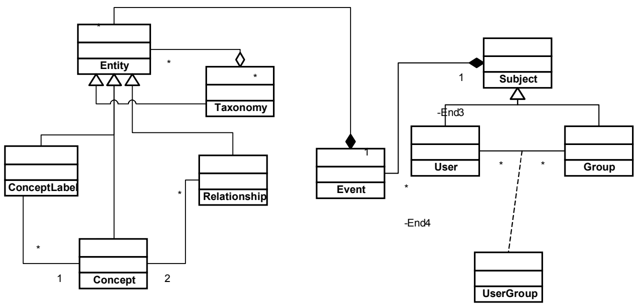


Figure 7. Refined conceptual model for metadata.

The system will be coded using Enterprise Java technologies and will employ a Web-based user interface. This includes a taxonomy browsing interface, full text search facility for bibliographic data, collaboration facilities, and an administrative module. In creating the software architecture, we use object oriented design patterns. The pattern based metadata architecture supports object-oriented design of software. Business logic components mapped to the concept taxonomy follow the classic Composite architecture, adding behavior to metadata structure. The History pattern supports multi-user collaboration logic and versioning. Software components for metadata creation will be designed using Model-View-Controller architecture. The system will support well-defined metadata vocabularies for interoperability with similar systems. Web services can be created to export vocabulary data using the well-known SKOS [16] schema. Bibliographic data conform to Dublin Core vocabulary. The internal pattern-based metadata schema is sufficiently rich in semantics to yield useful mappings to SKOS, while providing adequate flexibility for supporting the information system's object-oriented architecture.

To function as a component of the wider Semantic Web infrastructure, the system should support well-defined metadata vocabularies for interoperability. To achieve this, the data must be exportable to de-facto standard RDF/S and OWL format. Web services are created to export taxonomy data using SKOS RDF schema [16]. Additional bibliographic data will conform to Dublin Core vocabulary. Social network data is exposed as FOAF [25] graphs.

The internal pattern based metadata schema is sufficiently rich in semantics to yield useful mappings to the vocabularies mentioned, while providing adequate flexibility for supporting the information system's object-oriented architecture and reasoning services. A complete mapping of the data model's elements into the established dictionaries will be provided. Such services can facilitate incorporation of the information accumulated in the system into general-purpose Semantic Web search tools, e.g. Swoogle [24]. Similarly, an extension to our system may facilitate opportunistic expansion of the system's knowledge base using RDF crawling.

4 Conclusions

In this paper, we presented an argument for designing metadata schemata with design patterns. We described a simplified scenario of a semantic information system for collaborative metadata management for a digital library. We explore design challenges this scenario presents and then adapt a pattern called Composite from a standard software engineering design patterns reference [3] to address parts of these challenges. Additionally, we created a new design pattern, History, suggested by the collaborative metadata scenario and applicable to wider classes of problems in metadata design.

Using object-oriented approaches and patterns has shown to be a useful metaphor for metadata construction. Along with improving metadata architecture maintainability and extensibility, using patterns for metadata facilitates software design that uses similar principles.

Further research can take different directions:

- First, an attempt should be made at expanding the catalogue of patterns for semantic metadata design. The goal is to design an expandable framework of orthogonal design patterns suitable for creating complete maintainable and flexible metadata architectures. Such patterns may be adapted from existing software engineering patterns or may be new ones created specifically for metadata engineering.
- Second, sufficient experience should be accumulated applying these patterns in real-

world information systems. This should provide us with lessons learned to improve the pattern language.

- Last, ties between object-oriented paradigm and formal semantics for metadata such as envisioned in the ongoing Semantic Web project must be researched.

References:

[1] C. Alexander, S. Ishikawa, M. Silverstein, M.Jacobson, and I. Fiksdahl-King, and S.Angel, *A Pattern Language*, Oxford University Press, New York, 1977

[2] D. Djurić, D.Gašević, V.Devedžic, and V. Damjanović, UML Profile for OWL, *Web Engineering: 4th International Conference, ICWE 2004*, Munich, Germany, July 26-30, 2004, *Proceedings,* Springer-Verlag GmbH, 2004.

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

[4] W. Chang, *A Discussion of the Relationship between RDF-Schema and UML*, Group Note, WWW Consortium, 1998.

[5] A. Rector and C. Welty, eds, *Simple part-whole relations in OWL Ontologies*, W3C Editor's Draft, 24 Mar 2005.

[6] NCSTRL – Networked CS Technical Reports Library., http://www.ncstrl.org.

[7] N. Noy and A. Rector, eds., *Defining N-ary Relations on the Semantic Web*, W3C Working Draft, 21 July 2004.

[8] M. Uschold and C. Welty, eds. *Representing Classes as Property Values on the Semantic Web*, W3C Working Group Note, 5 April 2005.

[9] J. Yoder and R. Razavi, Metadata and Adaptive Object-Models, *Object-Oriented Technology: ECOOP 2000 Workshops, Panels, and Posters*, Sophia Antipolis and Cannes, France, June 2000, *Proceedings,* Springer-Verlag GmbH, 2000.

[10] S. Brockman, R. Volz, A. Eberhart, and P. Loffler, Visual modeling of OWL DL ontologies using UML, *ISWC 2004 3rd InternationalSemantic Web Conference,* Hiroshima Prince Hotel, Hiroshima, Japan, 7-11 November 2004.

[11] F.Manola and E. Miller, eds., *RDF Primer*, W3C Recommendation, 10 February 2004.

[12] I. Douglas, 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,* Reno, NV, October 10-13, 2001.

[13] IMS (Instructional Management Systems) Project from Educause. http://www.imsproject.org/.

[14] ADL *Sharable Courseware Object Reference Model*, SCORM. http://www.adlnet.org/.

[15] M.Sosteric and S.Hesemeir, When learning object is not an object: A first step towards a theory of learning objects, *International Review of Research in Open and Distance Learning*, October 2002.

[16] A.Miles and D. Brickley, eds*, SKOS Core Vocabulary Specification*, W3C Working Draft, November 2, 2005.

[17] W3C Semantic Web Activity, http://www.w3.org/2001/sw/

[18] T. Bernes-Lee, *Semantic Web Road map*, W3C Draft, September 1998.

[19] S. Ustymenko and D.G. Schwartz, An Agent-Oriented Logic for Reasoning about Belief and Trust, *Proceedings of 30th IEEE Annual International Computer Software and Applications Conference*, Chicago, IL, September 17-21, 2006, pp. 321-326.

[20] S. Ustymenko and D.G. Schwartz, Applying Object-Oriented Metaphors and Design Patterns in Defining Semantic Metadata, *Proceedings of the First On-Line Conference on Metadata and Semantics Research (MTSR'05)*, November 21-30, 2005.

[21] Fazal-E-Amin, Ansar Siddiq, Hafiz Farooq Ahmad, Aspect Design Pattern for Non Functional Requirements, in *Proceedings of 7th WSEAS Int. Conf. on Applied Computer and Applied Computational Science (ACACOS '08)*, Hangzhou, China, April 6-8, 2008, pp. 141-145

[22] Pierre Maret, Julien Subercaze, Jacques Calmet, Peer to Peer Model for Virtual Knowledge Communities, in *Proceedings of the 7th WSEAS International Conference on Artificial Intelligence, Knowledge*

Engineering and Data Bases (AIKED'08), University of Cambridge, Cambridge, UK, February 20-22, 2008, pp. 365-370

[23] Serena Pastore, Social Networks, Collaboration and Groupware Software for the Scientific Research Process in the Web 2.0 World, in *Proceedings of the 7th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases (AIKED'08),* University of Cambridge, Cambridge, UK, February 20-22, 2008, pp. 403-408

[24] Li Ding, Tim Finin, Anupam Joshi, Rong Pan, R. Scott Cost, Yun Peng, Pavan Reddivari, Vishal C Doshi, and Joel Sachs, Swoogle: A Search and Metadata Engine for the Semantic Web, in *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management,* November 09, 2004

[25] Dan Brickley, Libby Miler*, FOAF Vocabulary Specification 0.91*, Namespace Document 2 November 2007 - OpenID Edition, http://xmlns.com/foaf/spec/