

## Semantic Models for XML Schema with UML Tooling

David A Carlson<sup>1</sup>

<sup>1</sup> David Carlson & Associates, Inc.  
71 White Swan Ct.,  
Kalispell, MT 59901 USA  
[www.XMLmodeling.com](http://www.XMLmodeling.com)  
[dcarlson@xmlmodeling.com](mailto:dcarlson@xmlmodeling.com)

**Abstract.** Design and use of XML Schemas are essential tasks in the software engineering process for many systems, but duplication and different naming and design rules in these schemas are major roadblocks for systems integration. An ontology-driven design approach is described that uses semantic models during the creation and management of XML schema vocabularies. UML models are used for model-driven design of schemas and semantic annotations are used to associate UML elements with OWL concepts. Specific examples from financial services industry schemas illustrate the issues and benefits of semantic integration. A design tool named *hyperModel* implements some of these modeling activities and enables editing and browsing both UML models and OWL ontologies with a common interface of dynamically created class diagrams.

**Keywords:** UML, OWL, XML Schema, XSD, Modeling, Eclipse, Financial Services, FpML.

### 1 Introduction

Design and use of XML Schemas are essential tasks in the software engineering process for many systems. The importance of schemas is not limited to those organizations that embrace Service Oriented Architecture (SOA), but they are integral to the service contract of all applications and organizations that exchange information using XML documents. However, the relative ease of writing XML Schemas is both a blessing and a curse; we now have hundreds of schemas written by industry associations, IT departments, and individual developers. Semantic models may be the key to managing and integrating these XML vocabularies and making them part of a unified software engineering architecture.

A Semantic model is an *ontology* that represents a formal explicit description of a domain. In contrast to XML Schemas that define types based on syntax and structure, ontologies define classes as sets of individuals that satisfy required axioms. Ontologies range from simple taxonomies to deep knowledge about subjects such as medicine and chemistry. The recent adoption of the Web Ontology Language (OWL) by the W3C [12] is an important step in bringing the use of ontologies into Internet-oriented distributed systems.

Others have written about Ontology Driven Architecture and its potential for raising productivity of system integration [10]. The research described in this article focuses on the role of ontology in designing and deploying XML vocabularies. We propose that ontology-driven development of XML Schemas will lead to improved reuse and faster, higher quality integration. We also apply concepts and tooling from Model Driven Architecture (MDA) by

using UML models as the platform for visualizing and integrating XML Schema and OWL ontologies.

Specific examples from the financial services domain are used to illustrate:

- Designing and visualizing both XML Schemas and OWL ontologies using UML class diagrams.
- Assigning semantic annotations (OWL references) at design time in UML.
- Generating semantic annotations into XML Schema definitions.
- Semantic classification, discovery, and validation of models and XML instances.
- Integration of all these activities within a design workbench, named *hyperModel*, that is implemented in the Eclipse IDE platform.

Section 2 describes the use and benefit of semantic models applied to the design and management of XML Schemas. Section 3 introduces specific examples from the financial services domain. Section 4 describes *hyperModel*, an integrated tool for ontology-driven development. All processing of OWL ontologies in *hyperModel* is supported by the Eclipse EODM open-source project. The EMF Ontology Definition Metamodel (EODM) is an implementation of RDF(S)/OWL metamodels from the OMG's Ontology Definition Metamodel (ODM) using the Eclipse Modeling Framework (EMF) [11][4].

## 2 Semantic Models for XML Vocabularies

XML vocabularies are usually specified using XML Schemas. However, the schema language is limited to basic structural descriptions and, while focusing on the schema language expressions, developers often lose sight of the more important goal: creating a robust vocabulary for communicating between applications and organizations. The Unified Modeling Language (UML) can be used to overcome many of these limitations when designing XML vocabularies. By placing emphasis on the information model and visualizing the models using class diagrams, designers and business users are able to focus on the content instead of the syntax.

The UML also has benefits while designing OWL ontologies and when associating OWL concepts with XML Schema structures. The class diagrams provide a common visual presentation of schema structures and ontology concepts, and the shared language enables the linkage of semantics and implementation in a unified tool environment.

### 2.1 Modeling XML Vocabularies

Using UML for modeling the design of XML applications and generating XML Schemas [1] has been applied in production systems development for more than five years. A principal goal of this work is to build libraries of reusable business components that can be shared across applications and communicated with end-users via UML class diagrams.

Whenever possible, the constructs of XML Schema are mapped to corresponding elements in the UML. A profile of stereotypes is used to extend the UML for schema-specific definitions, for example to differentiate between elements and attributes in complexType definitions. This profile is applied when importing XML Schemas into UML and when



The development of component libraries based on CCTS is still in early stages. In all usage that we have seen thus far, the object class and property terms are assigned as part of component naming, but no taxonomy or ontology of the terms themselves is constructed. Semantic models can provide benefits for CCTS similar to those described in this article for UML and OWL ontologies.

## 2.2 Web Ontology Language (OWL)

The Web Ontology Language OWL is a semantic markup language for publishing and sharing ontologies on the World Wide Web [12]. In ontologies, the logical relationships are made explicit through OWL class definitions and other formal statements. This not only makes it easier for other human users of models to understand the specifically intended meaning, but also means that other tools can use the definitions transparently. An ontology differs from an XML Schema in that it is a knowledge representation, not a message format.

Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. An OWL ontology may include descriptions of classes, properties and their instances. An OWL *restriction* describes the class of all instances that fulfill a specific condition on a property. The key power of OWL is that classes can be defined by combining multiple restrictions and other classes. For that purpose, OWL provides logical operands to build intersections (and), unions (or) and complements (not) of other classes [16].

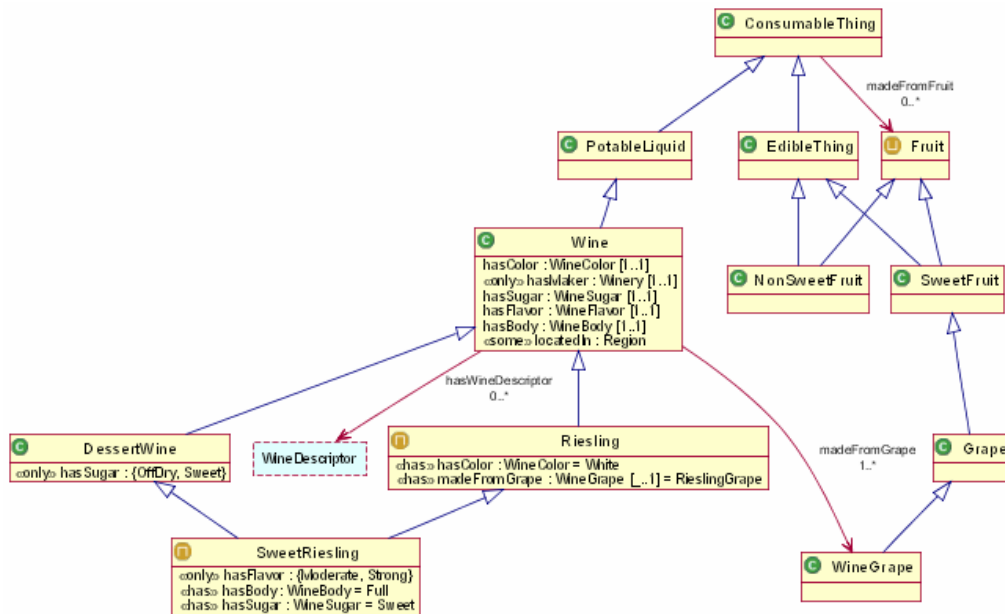
Work is underway to define an Ontology Definition Metamodel (ODM), plus a UML profile and mapping between OWL and UML [11]. Through this UML mapping, we can create class diagrams of OWL ontologies in a standardized notation. As part of the research described in this article, we have created a prototype application that renders class diagrams directly from OWL, without actually transforming the ontologies into UML models. We intend to align this notation with the ODM specification after it is finalized.

The class diagram shown in Figure 2 illustrates part of an ontology about food and wine that is described in the OWL Guide [12]. In it, we can easily see that a Wine must be made from one or more WineGrapes, which are kinds of SweetFruit. And SweetRiesling is a kind of wine that is an intersection of Riesling and DessertWine. This intersection class also defines other restrictions on the property values for hasFlavor, hasBody, and hasSugar. Thus, SweetRiesling is not a structural type definition as in XML Schema, but it is a semantic definition of what it means to be a sweet Riesling wine and how this wine differs from other kinds of wine.

Applications can exploit other tools to handle and analyze OWL models. One family of such tools is called reasoners. A *reasoner* is a service that takes the statements encoded (asserted) in an ontology as input and derives (infers) new statements from them. In particular, OWL reasoners can be used to:

- Reveal subclass/superclass relationships among classes
- Determine the most specific types of individuals
- Detect inconsistent class definitions

OWL reasoners are based on Description Logics (DL), which is a field of research that has studied a particular subset of first order logic that is proven to be computationally feasible for practical reasoning systems. One popular OWL reasoner, called Pellet, has an open-source Java implementation and is integrated with several OWL design tools [13].



**Fig. 2.** Class diagram view of an ontology about food and wine.

As described in the following sections, we can use OWL ontologies and DL reasoners to assist design time activities while modeling XML Schemas, and runtime semantic validation and mapping of XML messages.

### 2.3 Semantic Annotations

A standard mechanism is needed for associating semantic concepts with their realization in UML models, XML Schema types, and Web Service interfaces. The W3C Semantic Annotations for WSDL (SAWSDL) specification defines how to add annotations to XML Schema and WSDL components that refer to concepts defined in an ontology [14]. It also defines an annotation mechanism for specifying the structural mapping of XML Schema types to and from an ontology. In SAWSDL, these semantic annotations are implemented as XML attributes added to definitions in WSDL services or XML Schemas.

This specification emphasizes Web Service definitions, but it also includes annotations that can be applied to XML Schema definitions that are used without the services. This article describes how these semantic annotations are used in both XML Schemas and in UML models of those schemas to bind their definitions to a semantic model.

### 2.4 Benefits of Ontology and XML Schema

XML Schemas and OWL ontologies are useful on their own, but additional benefits are possible when using them in combination. These benefits are realized during design time while

creating schemas and at runtime when exchanging XML documents in service-oriented systems.

**Semantic Classification.** XML Schemas and their instances may be classified into taxonomic hierarchies defined by concepts in one or more ontologies. These taxonomic categories can be specified as either *primitive* or *defined* concepts. Primitive concepts can be used in a model to represent classes of individuals that users are expected to be able to classify naturally. Defined concepts can be used in a model to represent subclasses of the primitive ones that the system will be able to classify if needed [8]. For example, financial assets might be explicitly classified into primitive concepts for Cash, Bond, or Equity, whereas any schema type that includes a postal code property is classified as a kind of defined concept named PostalAddress.

Classification of complexType and simpleType schema definitions will assist in establishing equality or subsumption relationships between definitions from different schemas. For example, related kinds of addresses (postal, email, geographical coordinates), parties (individual, organization, government), and financial assets (cash, bond, equity) are inferred using an OWL reasoner. Existing schemas have many ways of defining and naming address structures, but all of these types can be classified as kinds of Address.

**Message or Component Discovery and Reuse.** One of the primary benefits of schema type classification is to assist designers in discovering and reusing existing components from XML vocabulary libraries. Concept discovery mechanisms are among the common benefits of using ontologies and description logics [8].

One of the principal reasons for work on industry standards such as the Universal Business Language (UBL) and the UN/CEFACT Core Components [2] is to enable reuse and easier integration of business systems. Those specifications describe component repositories and associated component metadata that will assist discovery. Use of domain ontologies with these vocabulary repositories may significantly improve the ability to achieve their goals.

**Unify XML Naming and Design Rules.** Many companies and standards groups create a Naming and Design Rules (NDR) specification that requires very specific styles for all XML Schema definitions [9]. Several of these specifications are related to work originating at OASIS, but there are significant differences between NDRs. Some NDRs require that XML Schema elements use short truncated names to reduce the size of XML documents in high volume transaction systems or when the XML vocabularies were based on older EDI systems. For example, CxlRegRsn is used instead of CancellationRejectReason (in FIX financial schemas), or types are named CE, CD, and ED (in HL7 healthcare schemas). Such XML vocabulary names carry little human readable meaning for someone who is not intimately familiar with these particular standards.

Semantic models can be used to associate shared concepts with these vocabulary terms. The ontology concepts enable translation of specialized NDR naming standards to a business terminology that is both human readable and machine processable by other applications.

**Semantic Validation of Messages.** Semantic classification also has benefits for runtime validation of XML messages in service oriented systems. Message validation using XML Schemas is limited to simple structural integrity, and that validation is often limited by schemas that define most content as optional. Alternatively, business rules for semantic validation might require that all XML message components that are a kind of Party must include a kind of Address somewhere in their content. This form of semantic validation rule would be written with reference to the ontological concepts and thus applied to any XML messages whose content can be classified into those concepts.

**Mapping Between Message Vocabularies.** Given the proliferation of XML Schemas, often containing very similar information, it is sometimes necessary to transform XML document instances from one schema to another. This involves mapping element names, changing structure, and transforming content values (e.g. format of dates or units of measure).

When the types in each schema are associated with shared semantic models, then the mapping rules can be written in terms of the semantic content of messages and not limited to simply mapping element names between each alternative schema [15]. This kind of semantic mapping is one of the primary goals of the W3C Semantic Annotations for WSDL specification [14].

### 3 Financial Services Example

We have applied UML model-driven design techniques to XML vocabularies from many industries, including banking, healthcare, manufacturing supply chain, and others. Based on that experience, we believe that the financial services industry is an especially good domain for applying semantic models to realize the benefits outlined in Section 2.4. Financial transactions depend on precise description of complex instruments, and there are a number of XML messaging standards that contain similar concepts that have been implemented with different XML naming and design rules.

Four examples of financial industry standards are:

- **Financial Information eXchange (FIX) Protocol:** Messaging standard for the real-time electronic exchange of securities transactions.
- **Financial products Markup Language (FpML):** XML specification for swaps, derivatives, and structured financial products.
- **Interactive Financial Exchange (IFX):** XML specification for electronic bill presentment and payment, business to business payments, business to business banking (such as balance and transaction reporting, remittance information), automated teller machine communications, consumer to business payments, and consumer to business banking.
- **Market Data Definition Language (MDDL):** XML specification for exchanging data necessary to account for, analyze, and trade financial instruments. MDDL defines common terms that provide a standard vocabulary so market data may be exchanged unambiguously between exchanges, vendors, redistributors, and subscribers.

It is possible to automate the transformation of these XML Schemas into OWL ontologies, but this would lead to a fragmented, suboptimal ontology design. We would end up with four ontologies that recreate the same independent descriptions using a different language. Instead,

we need to identify and describe the essential concepts used in these vocabularies and then associate the message terms from each schema into a common semantic model.

We have only just begun to design this ontology and cannot report results in this paper. Our strategy for creating the ontology is as follows:

1. Harvest key concepts from the schema definitions while looking for reusable abstract types and terms. The FpML schema is especially good in this regard. See for example the schema class diagram shown in Figure 1, where general types are defined for schedules and date intervals. These may yield a useful ontology of scheduling concepts needed to describe financial transactions. Similarly, FpML includes a generalization hierarchy for kinds of financial assets, although these are limited to the scope of messages included by FpML.
2. Organize an upper ontology of basic concepts that can be used to classify the terms used in each schema. We initially defined concepts for Party, PartyRole, ContactPoint, Asset, and Trade. Then the primitive and defined subclasses are chosen to describe the types used in these messaging standards.
3. Focus on the *essential semantics* that can be used to differentiate and classify terms used in these schemas. Don't attempt to duplicate all elements and types used in each schema. Identify the properties and restrictions that define a term and distinguish it from other related terms, as opposed to the many informative elements and operational data contained in large XML schemas.
4. Define ontology concept names using common industry terminology. In particular, don't use schema element naming that is based on implementation concerns such as the FIX element name truncation (e.g. CxlRegRsn is used instead of CancellationRejectReason). The resulting ontology and semantic annotations will provide a translation of FIX element names to more meaningful terminology.

## 4 *hyperModel* Tool Implementation

The *hyperModel* design tool has been under development and in active use for five years. Until now, its focus has been on model-driven design and visualization of XML Schemas, and bi-directional transformation between XML Schema and UML models. *hyperModel* is now being extended with a proof-of-concept implementation for OWL modeling, semantic annotations of UML models and XML schemas, and semantic classification of model elements.

The design objectives of *hyperModel* are:

- Use Eclipse Modeling Framework (EMF) for model-driven design
  - Metamodels defined by current industry standards: UML 2.1, OWL, and XML Schema
- Common UML style interface for viewing and editing UML and OWL models
- Visualization of large models using an interactive browsing metaphor with class diagrams
- UML profile for semantic annotations that bridge UML and OWL models
- Wizard-driven transformations for importing XSD into UML models and generating XSD from UML, including support for semantic annotations

#### 4.1 Metamodels for UML2, OWL, and XML Schema

Eclipse provides open-source implementations of these three metamodels, where all use EMF Ecore as their meta-metamodel. EMF Ecore is aligned with the OMG standard Essential MOF (EMOF) and provides complete and robust Java implementations generated from Ecore models. Because the UML2, OWL, and XSD metamodels are all based on Ecore, they share a common model reflection API and a growing library of Eclipse tools that support any Ecore model.

The Eclipse UML2 and XSD metamodels are already used in several large vendor commercial products, so they are complete, robust, and well supported by the Eclipse community. All processing of OWL ontologies in *hyperModel* is supported by the Eclipse EODM open-source project [4]. The EMF Ontology Definition Metamodel (EODM) is an implementation of RDF(S)/OWL metamodels from the OMG's Ontology Definition Metamodel (ODM) with additional support for parsing, inference, model transformation, and editing. The EODM project is still maturing and the ODM specification was voted for adoption just prior to this article's publication, but its common basis in EMF Ecore enables rapid integration with other EMF models in the *hyperModel* design tool.

#### 4.2 UML Style Interface

There are dozens of UML tools available and they are in common use by many developers. As a result, many users are familiar with the basic style of editing UML models using a tree view of the model's packages, classes, and attributes, plus a class diagram for visualizing and/or editing a model. *hyperModel* is able to open and edit any UML model that is compatible with the Eclipse UML2 metamodel implementation [5].

On the other hand, OWL models and OWL editing tools use specialized notation and terminology that are unfamiliar to most developers. With *hyperModel* we are exploring the idea of viewing and editing a subset of OWL ontology definitions using a familiar UML style interface, while still working on a native OWL model and representation, i.e. without actually translating the ontology into a UML model. A UML style presentation is applied to an OWL model. When a user familiar with OWL wants to edit more advanced definitions, he/she can drop down into a complete native OWL editing environment integrated within the same tool and editing the same model instance.

The screen in Figure 3 shows the Model Explorer view where any combination of UML models and/or OWL ontologies may be opened simultaneously. In this example, we are browsing a UML model of the FpML messaging standard that was imported from XML schemas, plus an OWL ontology is open along with two standard data type libraries.

The class diagram is created dynamically by selecting one or more classes in the Model Explorer, or by double-clicking on a class in a larger diagram to drill-down into a more specific subset. We refer to these dynamic diagrams as *dynagrams*. For large class diagrams, the toolbar includes options for controlling the depth and breadth of expansion while following model associations and laying out the diagram. The editor includes a history of recent diagram views and back/forward arrows so that you can explore a large model with a browser-like style. The diagrams shown in Figures 1 and 2 were also created in this view and then saved to PNG files.

The tabbed Properties view below the diagram can be used to display or edit properties of the current selection in either the class diagram or the Model Explorer. Another view is under

development, not shown here, where a user can examine and navigate the class inheritance hierarchy of any class selected in the Model Explorer or class diagram.

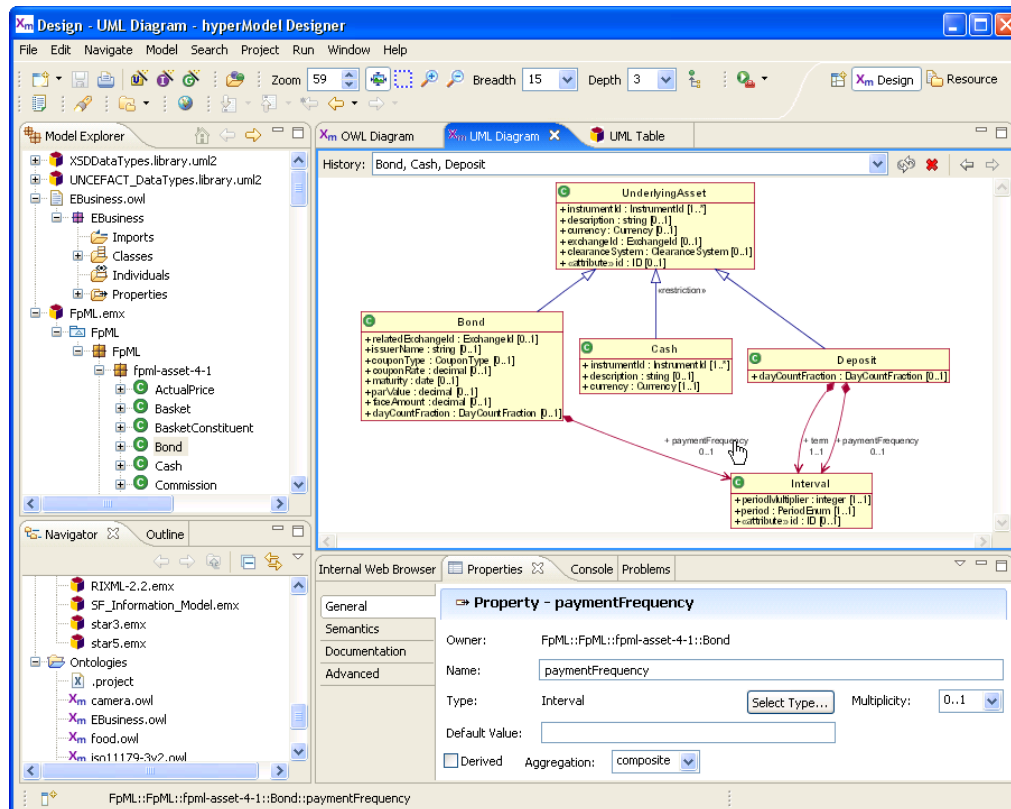
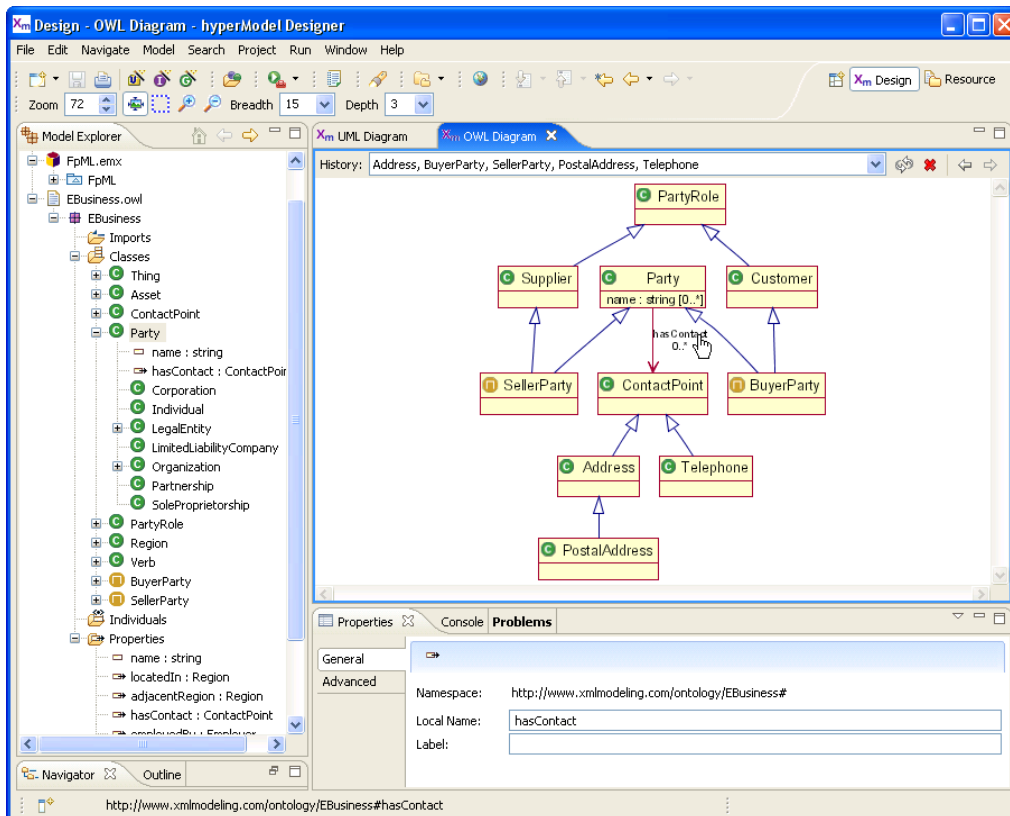


Fig. 3. *hyperModel Designer* showing a UML model imported from FpML schemas.

The objective of *hyperModel* is to provide a common user interface style when working with either UML or OWL models. Figure 4 shows a screen while editing the *EBusiness.owl* ontology definitions. This is a very simple ontology but it illustrates the same use of Model Explorer, class diagram, and Properties views. The UML style presentation in the Model Explorer view is still under development, but you can see how we include an OWL property definition under the class(es) that are in the property's domain (under Party). We also include property restrictions under other defined classes, where the property is shown with its narrowed definition (e.g. restrictions on cardinality, type, or value). This presentation of restrictions has a valid interpretation in the UML2 specification where it is called a property *redefinition*. But to reemphasize, this presentation of OWL is only a rendering of the native OWL definitions and not a transformation into a UML model. More advanced OWL definitions that are not easily mapped into UML are preserved, even if they are not shown in this UML style view.



**Fig. 4.** *hyperModel* class diagram for a simple e-Business OWL ontology.

For users who want to see the native OWL perspective, the Model Explorer lists all property definitions under a “Properties” folder, seen at the bottom of the explorer view in Figure 4. Similarly, OWL individuals are listed in the explorer. When completed, this tool will include an OWL tab in the Properties view where all details of an OWL definition can be seen, created, and edited.

The OWL class diagrams are also rendered directly from the ontology definitions and, as with the UML model diagrams, are created dynamically while exploring concepts in an ontology. We will align the class diagram notation with that defined in the ODM specification, including stereotypes and tags on the classes and properties [11]. Other research on using UML diagrams with OWL predates the ODM specification [17, 18] and influenced the standard UML mapping and profile stereotypes in ODM. *hyperModel* follows a similar approach as described by this prior work, but extends it with dynamic diagrams and UML style presentation of native OWL definitions.

We added one enhancement over UML notation described in the prior work by using a notational convention for displaying partial cardinality restrictions on properties. For example, as seen in the Riesling class in Figure 2, the ‘madeFromGrape’ property is a restriction on the same property as defined for its superclass Wine, but only the maximum cardinality is restricted to 1. We could show the effective restriction on this property, which is 1..1 because the inherited property is 1..\*. But for an ontology designer looking only at this class, it would

not be clear where each restriction is declared. We display the UML class diagram with property multiplicity of ‘[...1]’ to signify that only the maximum cardinality is restricted and the underscore indicates that the minimum cardinality is inherited. Also, we show a restricted object property within the class attribute partition (instead of as an association) to avoid diagram clutter and also as a hint that it is restricted from an inherited property.

The class hierarchy view (not shown) will present a class subsumption hierarchy for a selected OWL concept. As with other OWL views, it is rendered directly from the OWL model using APIs from EODM. However, unlike the class hierarchy view of a UML model, the OWL view shows a class subsumption hierarchy inferred by an associated DL reasoner. This kind of class subsumption view is common with all ontology modeling tools and *hyperModel* will allow configuration and choice of several alternative reasoners.

A *hyperModel* user can move easily between any number of UML and OWL models that are open in the explorer. While editing a UML model, all classes in all open models are available when creating associations, so you can integrate a design that is divided across several model files. We are currently working on providing this same modularity for linking references across OWL ontologies and for creating semantic annotations that bridge UML elements to OWL concepts.

### 4.3 Semantic Annotations

To model the semantic references from UML elements to OWL concepts, we implemented a small UML profile with one stereotype that allows this information to be captured in a UML model extension. This profile is based on the W3C Semantic Annotations for WSDL specification that was described in Section 2.3. These semantic annotations can be created and edited in *hyperModel* using a properties view tab that was added for this purpose. As shown in Figure 5, the Semantics tab is visible whenever the current selection is on a UML named element. Whenever the user enters text in any of these three fields, the corresponding UML profile stereotype is added automatically.

Figure 5 illustrates the integration of UML and OWL editing. The user is browsing a class diagram of a UML model imported from FpML schemas, while in the Model Explorer view the concept hierarchy of the EBusiness ontology is displayed. The user has assigned a semantic annotation from the UML class named *UnderlyingAsset* to the OWL concept named *Asset*.

In this example, the *EBusiness.owl* ontology is loaded from the user’s local file system. However, an important use case for semantic integration is reference to *shared ontologies* that are developed and maintained by a community of users. As with most OWL modeling tools, an ontology can be loaded into *hyperModel* by entering the remote URI that identifies a shared ontology. That ontology is loaded and rendered in a familiar UML style and used to assign semantic annotations to an application specific UML model.

This is a proof-of-concept implementation for assigning semantic annotation to UML elements. Future versions of *hyperModel* will enhance this interface with drag-and-drop capability for assigning OWL concepts to UML elements. We will also add support for hybrid class diagrams that combine UML classes and OWL concepts in one diagram, distinguished by color and class icons, with dependency relationships illustrating the semantic assignment.

Another enhancement under development is a new search dialog and search results view (similar to those provided in Eclipse IDE for Java) that support semantic classification and component reuse as described in Section 2.4. A similar ontology enabled search feature is described in [19] but *hyperModel* will integrate the searching into Eclipse IDE views.

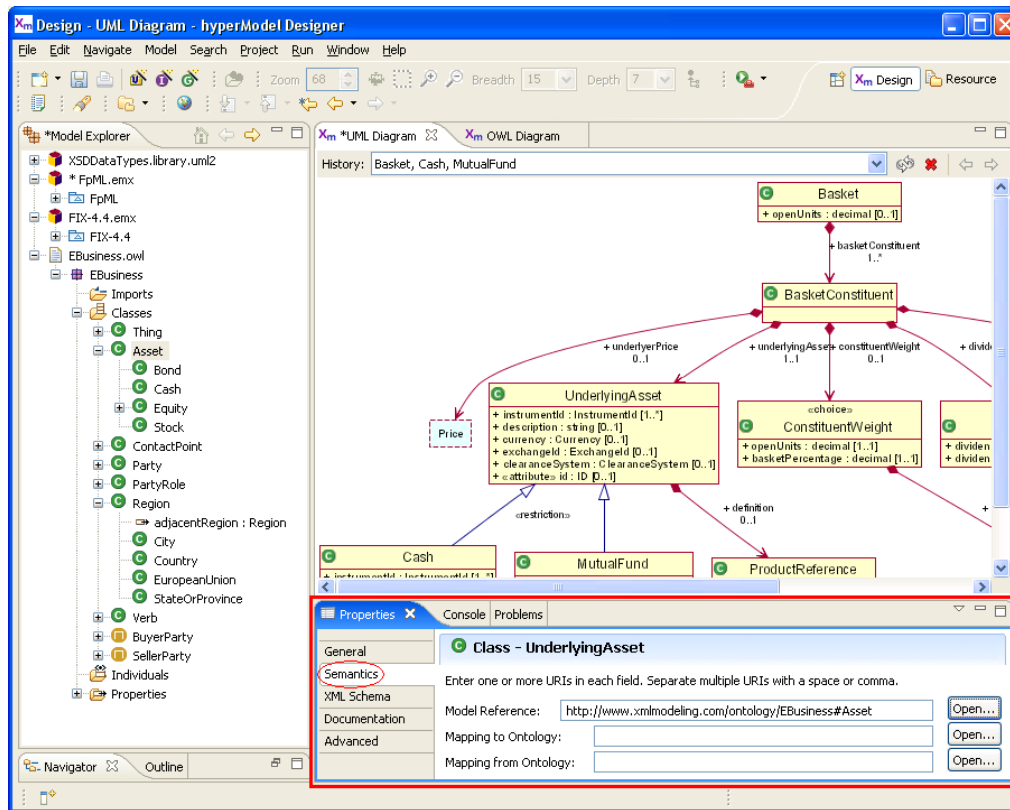


Fig. 5. Adding semantic annotations to a class in the FpML model.

## 5 Conclusions

In current development practice the design of XML Schemas is often disconnected from a coherent software engineering process and not aligned with other related schemas created within the same organization or across industry standards groups. The use of semantic models may help to bridge these gaps and provide a focal point for shared semantics and component reuse.

We are working toward the goal of demonstrating and fulfilling the benefits outlined in Section 2.4 that enable semantic management and design of XML vocabularies. An ontology-driven software engineering process for XML vocabularies would provide:

- Design-time classification and consistency checking
- Discovery and reuse XML message components
- Runtime semantic validation of XML messages
- Runtime message mapping based on semantic content

In the financial services application we will start with OWL concept taxonomies and demonstrate benefits for component classification, discovery, and reuse. Then, through an

iterative development process, we will identify and model the essential semantics that distinguish alternative kinds of financial assets, trades, and party roles. These enhanced ontologies will be applied to classify XML messages based on their element content and not only on explicit concept assignments.

Additional features of the integrated design environment in *hyperModel* are still under development. In particular, we will extend *hyperModel* to cover semantic classification, discovery, and validation of models and XML instances. This enhancement will require integration with an OWL DL reasoner. To accomplish this, we are designing an Eclipse extension point for declaring available reasoners. In Eclipse, the use of extension points in plug-in configuration files is a primary mechanism for modularity and extensibility of the integrated tool platform. We expect to build OWL reasoner integration for the open-source Pellet library [13] and for the more general DIG interface [3].

## References

- [1] D. Carlson. Modeling XML Applications with UML: Practical e-Business Applications, Addison-Wesley, 2001.
- [2] UN/CEFACT Core Components Technical Specification (CCTS), November 2003, [http://www.unece.org/cefact/ebxml/CCTS\\_V2-01\\_Final.pdf](http://www.unece.org/cefact/ebxml/CCTS_V2-01_Final.pdf)
- [3] DL Implementation Group (DIG) Description Logic reasoner interface, <http://dl.kr.org/dig/index.html>.
- [4] EMF Ontology Definition Metamodel (EODM) Eclipse open-source project, <http://www.eclipse.org/emft/projects/eodm>
- [5] Eclipse UML2 metamodel open-source project, <http://www.eclipse.org/uml2>
- [6] Financial Information eXchange (FIX) Protocol, <http://www.fixprotocol.org>.
- [7] Financial Products Markup Language, <http://www.fpml.org>.
- [8] Horrocks, I., McGuinness, D., and Welty, C. Digital Libraries and Web-Based Information Systems. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *Description Logic Handbook*, Chapter 14. Cambridge University Press, 2003.
- [9] Naming an Design Rules (NDR), XML Cover Pages, <http://xml.coverpages.org/ndr.html>
- [10] P. Tetlow, J. Pan, D. Oberle, E. Wallace, M. Uschold, and E. Kendall. Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering, <http://www.w3.org/2001/sw/BestPractices/SE/ODA/>.
- [11] Ontology Definition Metamodel (ODM), <http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf>.
- [12] M. Smith, C. Welty, and D. McGuinness (editors). OWL Web Ontology Language Guide, <http://www.w3.org/TR/owl-guide/>, February 2004.
- [13] Pellet open-source OWL DL reasoner, <http://www.mindswap.org/2003/pellet/>.
- [14] J. Farrell and H. Lausen (editors). Semantic Annotations for WSDL, <http://www.w3.org/2002/ws/sawSDL/spec/>, August 2006.
- [15] M. Uschold and M. Gruninger. Ontologies and Semantics for Seamless Integration, SIGMOD Record, Vol. 33, No. 4, December 2004.
- [16] H. Knublauch, D. Oberle, P. Tetlow, and E. Wallace (editors). A Semantic Web Primer for Object-Oriented Software Developers. <http://www.w3.org/TR/sw-oosd-primer>, March 2006.
- [17] K. Baclawski, et al. Extending the Unified Modeling Language for Ontology Development.
- [18] S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual Modeling of OWL DL Ontologies Using UML. In Proceedings of ISWC 2004, pp. 198-213, 2004.
- [19] W. Rungworawut and T. Senivongse. Using Ontology Search in the Design of Class Diagram from Business Process Model. Transactions on Engineering, Computing and Technology, v12, March 2006.