Context Mediation: Ontology Modeling using Web Ontology Language (OWL)

Philip Tan Eik Yeow

Working Paper CISL# 2004-11

June 2004

Composite Information Systems Laboratory (CISL)
Sloan School of Management, Room E53-320
Massachusetts Institute of Technology
Cambridge, MA 02142

TABLE OF CONTENTS

Context Mediation: Ontology Modeling using Web Ontology Language (OWL)

by

Philip Tan Eik Yeow

Submitted to the SMA office on May 17, 2004
In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Computer Science

ABSTRACT

The Context Interchange strategy is a novel approach in solving heterogeneous data source interoperability problem through context mediation. In the recent implementation, a FOL language is used in the modeling and implementation of the application ontologies. In this project, we close the gap between COIN and Semantic Web by adopting the use of W3C Recommendation for ontology publishing, OWL Web Ontology Language in the strategy realization. The ontological model in COIN is represented in OWL, by mapping the respective ontological concepts in COIN to its counterparts. Emerging rule markup language of RuleML is used for modeling rule-based metadata in the ontology. In conjunction with that, we have developed a prototype demonstrating the use of this COIN-OWL ontology model.

Keywords:

Context Interchange strategy, Web Ontology Language (OWL), Semantic Web, Ontology Modeling, RuleML, Context Mediation, Extended Context Interchange (eCOIN), Data integration, Heterogeneous data source interoperability

Dissertation Supervisors:

1. Prof. Stuart Madnick, SMA Fellow, MIT
2. Assoc. Prof. Tan Kian Lee, SMA Fellow, NUS

CHAPTER 1
Introduction

The Context Interchange strategy [23] is a mediator-based approach for achieving semantic interoperability among heterogeneous data sources and receivers. Using COINL, a language firstly introduced in [12], the detection and mediation of data heterogeneity was efficiently realized in a prototype implementation using the constraint logic programming system ECLiPSe in Prolog [11]. COINL is the lingua franca for the modeling and implementation of the ontology to describe the disparate and heterogeneous data sources. From the Extended Context Interchange project (eCOIN) [15], the concrete syntax for knowledge representation and context mediation has evolved from the proprietary COINL language to using FOL/Prolog.

Of late, a large number of knowledge representations scheme for ontological knowledge has been proposed by various research groups and bodies. The list goes from general-purpose Resource Description Framework (RDF) [13] for information representation on the Web, to DARPA Agent Markup Language (DAML) [1] as extension of RDF aimed at facilitating agent interaction on the Web, and special-purposed ontology, such as ebXML that aims to "enable enterprises of any size, in any global region, to conduct business using the Internet" [3]. One recent unifying effort in creating *the* ontology language for the Web is the Web Ontology Language (OWL) by the World Wide Web Consortium [21], which together with RDF, forms the major building blocks for the Semantic Web [8].

1.1 Project Motivation

With OWL being released as W3C Recommendations, our effort is aimed at bridging the gap between eCOIN and Semantic Web by adopting the use of OWL for ontology modeling in COIN.

This is inline with our broader vision in heterogeneous data integration effort. The adoption of OWL in our context interchange framework also opens the door for future interoperability with other readily available ontology. As pointed out by Kim [19], “development of decentralized and adaptive ontologies have value in and of themselves, but whose full potential will only be realized if they are used in combination with other ontologies in the future to enable data sharing”.

1.2 Thesis Organization

Our work on using OWL and RuleML for ontology modeling and implementation of eCOIN for heterogeneous context mediation is detailed in this thesis. The thesis is organized as follows. Section 2 highlights the imperative background fundamental of the subject for the understanding of the paper, together with a review of relevant work. Section 3 describes the detailed design of ontology modeling in eCOIN using OWL. Section 4 details the implementation strategy and the prototype developed. And we conclude the paper with a summary and future research direction in the final section.

CHAPTER 2
Core Technology and Framework

2.1 Context Interchange Framework

Before describing the proposed eCOIN-OWL strategy, it is necessary to provide a summary of the architecture of a Context Interchange strategy. In particular, the understanding of the Extended Context Interchange (eCOIN) [15] is imperative to motivate the proposed strategy.

Our work based and extends on the previous work in the Context Interchange effort at MIT, chiefly on the Extended Context Interchange (eCOIN) implementation. Similar to COIN [12], eCOIN is a realization of the Context Interchange strategy articulated in [22] in the form of a data model, reasoning algorithm, and a prototype implementation. However, eCOIN is an improvement over COIN in terms of semantic representations, context reasoning and mediation, and prototype implementation.

2.1.1 Context Interchange Example

We believe one of the easiest ways to understand the Context Interchange framework is via concrete example illustration. Consider we have two financial data sources: Worldscope (worldscope) and Disclosure Corporate Snapshot (disclosure).

Worldscope provides basic financial information on public companies worldwide, while Disclosure is a information directory on companies publicly traded on U.S. exchanges. Worldscope reports all the financial data information in USD, and on the scale factor of 1000, while disclosure reports the financial data information in the currency of the country of incorporation of the companies, and on the scale factor of 1.

Using these financial data sources, the users are able to post queries on the public companies of interest. For example, to retrieve the sales data of Daimler-Benz AG from the worldscope database, the user may issue the following query:

```
select WorldcAF.SALES
from WorldcAF
where WorldcAF.COMPANY_NAME = "DAIMLER-BENZ AG";
```

Using the eCOIN prototype [2], the following results is obtained:

WorldcAF.LATEST_ANNUAL_FINANCIAL_DATE	WorldcAF.SALES
12/31/93	56,268,168

On the other hand, to retrieve the data from disclosure, the following query is posted:

```
select DiscAF.LATEST_ANNUAL_DATA, DiscAF.NET_SALES
from DiscAF
where DiscAF.COMPANY_NAME = "DAIMLER BENZ CORP";
```

And the following result is retrieved:

DiscAF.LATEST_ANNUAL_DATA	DiscAF.NET_SALES
12/31/93	97,737,000,000

Here, we can note the discrepancy in data due to the difference in context of the data sources, both in the currency and the scale factor used.

In a conventional information system, to perform a join table query between Worldscope and Disclosure, these context disparities has to be resolved manually and encoded in the SQL query. Using COIN, these context discrepancies (different company name format, date format, financial data currency type and scale factor) are mediated automatically and queries such as the follow can be posted without knowing the actual context:

```
select WorldcAF.TOTAL_SALES, DiscAF.NET_INCOME
from DiscAF, WorldcAF
where DiscAF.COMPANY_NAME = "DAIMLER-BENZ"
and DiscAF.COMPANY_NAME = WorldcAF.NAME
and WorldcAF.LATEST_ANNUAL_DATA = "12-31-93";
```

This automated context reasoning and mediation capability is the essence of the Context Interchange strategy.

2.1.2 COIN Ontology Model

The Context Interchange framework employs a hybrid of loosely- and tightly-coupled approach in data integration in heterogeneous data environment. The Context Interchange framework was first formalized by Goh et. al in [16] and further realized by Firat [15]. The Framework comprises three major components:

- The *domain model*, which is a collection of rich types, called *semantic types*. The domain model provides a lexicon of *types*, *attributes* and *modifiers* to each semantic type. These semantic types together define the application domain corresponding to the data sources which are to be integrated.
- The *elevation theory*, made up of an array of *elevation axioms* which define the data types of the data source, and its correspondence with the semantic types in the domain model. Essentially, this maps the primitive types from the data source to the rich semantic types in the application domain.
- The *context theory* comprising declarative statements which either provide for the assignment of a value to a modifier, or identify a *conversion function* which can be used as the basis for converting the values of objects across different contexts.

These three components forms the complete description of the application domain, required for the context mediation procedure as described in [12].

From the conceptual ontology model, the axioms in the eCOIN framework are realized using logic programming in Prolog. The collection of axioms present in an instance of this framework constituted an eCOIN FOL/Prolog program. This is the native language of which eCOIN framework operates on.

2.2 Web Ontology Language (OWL)

The OWL Web Ontology Language is designed for use by applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full.

The Semantic Web is a vision for the future of the Web, in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML's ability to define customized tagging schemes and RDF's flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language what can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema.

OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web.

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.

- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

- OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. Owl Lite also has a lower formal complexity than OWL DL.
- OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.

- OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

OWL Lite uses only some of the OWL language features and has more limitations on the use of the features than OWL DL or OWL Full. For example, in OWL Lite classes can only be defined in terms of named superclasses (superclasses cannot be arbitrary expressions), and only certain kinds of class restrictions can be used. Equivalence between classes and subclass relationships between classes are also only allowed between named classes, and not between arbitrary class expressions. Similarly, restrictions in OWL Lite use only named classes. OWL Lite also has a limited notion of cardinality - the only cardinalities allowed to be explicitly stated are 0 or 1.

As a concrete example of OWL, consider the following ontology in OWL, which states that GraduateStudents are Students with a degree of either BA or BS

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="GraduateStudent">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#hasDegree"/>
        </owl:onProperty>
        <owl:someValuesFrom>
          <owl:Class>
            <owl:oneOf rdf:parseType="Collection">
              <Degree rdf:ID="BA"/>
              <Degree rdf:ID="BS"/>
            </owl:oneOf>
          </owl:Class>
        </owl:someValuesFrom>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#Student"/>
  </owl:Class>
</rdf:RDF>
```

Equivalently, the example can be described in a UML class diagram in Figure 2-1.

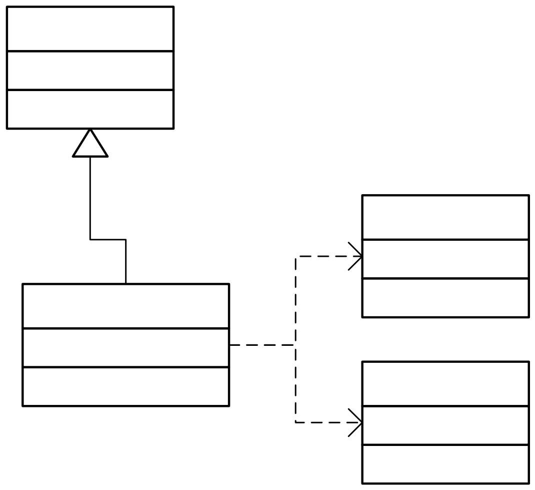


Figure 2-1: Class diagram for sample OWL ontology

2.3 Rule Markup Language (RuleML)

RuleML Initiative is a collaboration with the objective of providing a basis for an integrated rule-markup approach that will be beneficial to the committee and the rule community at large. This shall be achieved by having all participants collaborate in establishing translations between existing tag sets and in converging on a share rule-markup language. The main goal for RuleML kernel language is to be utilized as a specification for immediate rule interchange.

Rules can be stated (1) in natural language, (2) in some formal notation, or (3) in a combination of both. Being in the third, 'semiformal' category, the RuleML Initiative is working towards an XML-based markup language that permits Web-based rule storage, interchange, retrieval, and firing/application.

The XML schema definition of RuleML can be viewed as syntactically characterizing certain semantic expressiveness subclasses of the language. As eCOIN represents the ontological model in Prolog, which is in the horn-logic family, our use of RuleML is focused on the datalog and hornlog sublanguage.

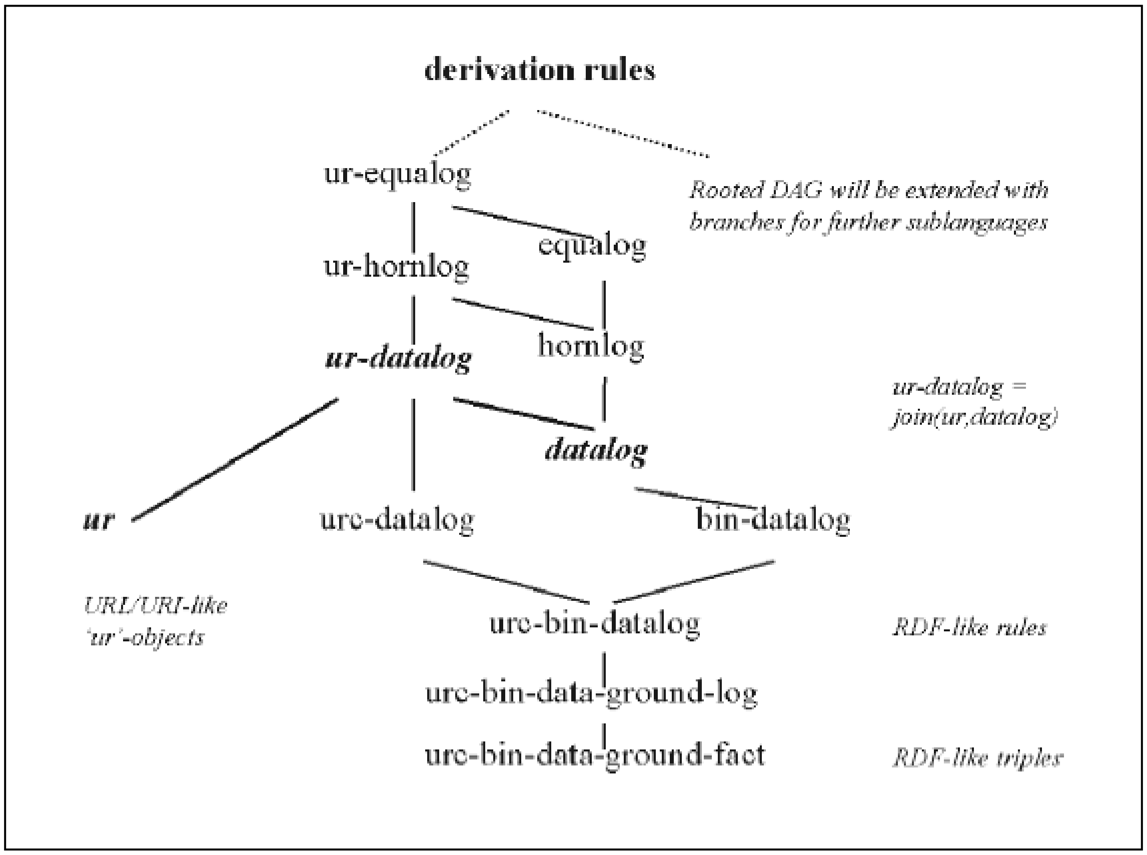


Figure 2-2: Hierarchical view of the RuleML sublanguages [9]

These two sublanguages provide a comprehensive language facility in describing rules encoded in Prolog. The XML concrete syntax of RuleML of these two classes uses the following constructs for rule representation:

Tag	**Uses**
Datalog	
`rulebase`	root element uses ‘imp’ rules and ‘fact’ assertions along with ‘query’ tests as top-level elements
`imp`	short for ‘implication’, usable on the rulebase top-level and uses a conclusion role _head followed by a premise role _body
`fact`	fact assertions are usable as degenerate rules on the rulebase top-level
`_head`	_head role is usable within ‘imp’ rules and ‘fact’ assertions; uses an atomic formula
`_body`	_body role is usable within ‘imp’ rules and ‘query’ tests; uses an atomic formula and an ‘and’
`and`	an ‘and’ is usable within _body’s to express conjunction
`atom`	atomic formulas are usable within _head’s, _body’s, and ‘and’s
`_opr`	usable within atoms as operator, uses the rel(ation) symbol
`ind`	short for ‘individual’. Individual constant, as in predicate logic
`var`	short for ‘variable’. Logical variable, as in logic programming
`rel`	relation or predicate symbol
Hornlog	
`cterm`	complex, compound, or constructor terms are usable within other cterms, tups, rolis, and atoms; uses _opc ("operator of constructors") role followed by sequence of five kinds of arguments
`_opc`	c(onstruc)tor symbol; usable within c(onstructor) terms.
`ctor`	constructor

As example of the use of RuleML, consider the following rule:

“The discount for a customer buying a product is 5.0 percent if the customer is premium and the product is regular.”

Expressed using the RuleML language constructs introduced above, this rule can be encoded as:

```
    <imp>
    <_head>
      <atom>
        <_opr><rel>discount</rel></_opr>
        <var>customer</var>
        <var>product</var>
        <ind>5.0 percent</ind>
      </atom>
    </_head>
    <_body>
      <and>
        <atom>
          <_opr><rel>premium</rel></_opr>
          <var>customer</var>
        </atom>
        <atom>
          <_opr><rel>regular</rel></_opr>
          <var>product</var>
        </atom>
      </and>
    </_body>
  </imp>
```

Or equivalently, the rule can be expressed in Prolog as follows:

```
discount(Customer, Product, 5.0) :- premium(Customer), regular(Product)
```

Note that the above rule uses only the datalog sublanguage of RuleML. As the application ontologies in COIN may involve more complex rules, our design and implementation uses both the datalog and hornlog sublanguages.

2.4 Related Work

In [20], Lee has presented a XML-based metadata representation for the COIN framework. The essence of the work lies in modeling and storing of the metadata in RDF format as the base format. A number of intermediate representations of were proposed: RDF, RuleML, RFML and the native Prolog representation used in COIN. The core ontological model of COIN in RDF format is transformed into the aforementioned intermediate representation by applying Extensible Stylesheet Language Transformation (XSLT) on the fly. Context mediation for heterogeneous data is then executed using the ontological model encoded in the COIN language. It is worth noting that the approach proposed in this work primarily deals with a single representation at a time. The intermediate ontological model is represented in RDF, RuleML or RFML individually, but not as a combination of the different formats, which is the approach taken in our proposal.

In a related work of eCOIN [15], high-level suggestions of eCOIN to Semantic Web mapping using OWL is proposed as future works direction. Firat compared and contrasted eCOIN and OWL in terms of language constructs, and presented a mapping of the domain model from eCOIN to OWL. In particular, he suggested the domain model compatibility of eCOIN and OWL, where domain model concepts of Semantic Type and Attribute can be represented trivially as Class and ObjectProperty respectively in OWL. The ‘is-a’ relationship commonly found in object-relation framework is translated into subClassOf construct in OWL.

One relevant effort in the Semantic Web/OWL space is Context OWL (C-OWL) [10], a language whose syntax and semantics have been obtained by extending the OWL syntax and semantics to allow for the representation of contextual ontologies. However, the extension focused on limited context mapping using a set of bridge rules that specify the relationship between contexts as one of the following: equivalent, onto (superset), into (subset), compatible, incompatible. The limited expressiveness of the language fails to address the contextual differences such as those possible with COIN.

CHAPTER 3
Context Interchange in OWL

One major perspective the Context Interchange strategy employs is the relational view of the data. The canonical representation of data source is the relational data model [15]. Semi-structured data, including information from web pages can be used, but they have to be first converted to relational sources, for example, using the Cameleon web wrapper engine. This aspect of the strategy is one distinct area that sets itself apart from the common usage of OWL, where ontology and data are maintained in the semi-structured format of OWL.

Intuitively, the use of OWL in COIN can be viewed as the meta-ontology layer on top of OWL, providing an extension to OWL to support context-aware ontology to the current context-oblivious ontology in OWL.

For convenience and brevity, we refer to this context modeling strategy for COIN using OWL as COIN-OWL in this text.

3.1 Approach

In eCOIN, the FOL/Prolog program formed by the collection of domain model definitions, elevation theories and context theories is used to detect and mediate context disparity and heterogeneity in a query using an abductive procedure defined in [12]. One important principle of our work is to preserve this constraint programming engine in the COIN framework.

We adopt layered architecture in the adoption of OWL in context interchange framework: (1) the domain ontology will be modeled in OWL (and its extension or relevant technology), (2) the ontology will be transformed to eCOIN FOL/Prolog as the native representation of the domain, and finally, (3) the native program will be taken as input to the abductive engine for context mediation. The high-level architecture of the approach is illustrated in Figure 3-1.

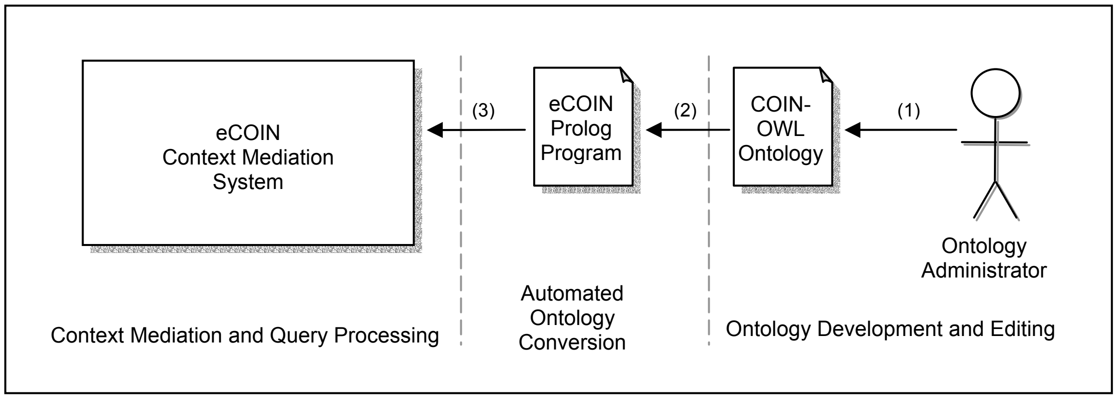


Figure 3-1: Three-tier approach for Context Interchange ontology modeling using OWL

The OWL ontology model can be viewed as the front-end of the system, where it is the main interfacing layer to the user of the eCOIN system. In the intermediate layer, the transformation from OWL to the native FOL/Prolog program will be transparent to the users. The transformation process is detailed in the later section of the thesis. With the derived program in its native FOL/Prolog format, the existing mediation engine can be reused in its entirety.

The big win of this approach is that it minimizes re-work: there is little value in reinventing the wheel, especially when the current functionality of the system provides the total capability required. At the same time, the abstraction of the middle tier of the architecture shielded the users from the actual implementation of the COIN context mediator. This componentization fulfills our aim of adoption of OWL in the framework, yet ensuring minimal impact to the existing COIN system.

3.2 OWL and Rule-based Ontology

One major challenge of the adoption of OWL in the ontology model is that the COIN ontology model encompasses a number of constructs that are not available in OWL. Constructs such as Domain Model and Elevation Axioms can be represented in OWL rather easily – conceptually, these constructs describes the relationship among the data types, and can be modeled accordingly using corresponding constructs in OWL that express relationships among classes.

The problem, however, lies in the modeling of context theory, which is the pivotal component in the framework for context interchange. The collection of context axioms in a context theory is used either to provide for the assignment of a value to a modifier, or identify a conversion function, which can be used as the basis for converting the values of objects across different contexts. Often, the expressiveness of rules is required to define the conversion of a semantic type in the source context to a different context.

In our proposed design, axioms requiring such flexibility are encoded in RuleML. RuleML allows rule-based facts and queries to be expressed in the manner similar to conventional rule language such as Prolog. The concrete representation of RuleML is XML, which fits seamlessly in our effort to standardize the ontology representation in eCOIN.

We foresee that RuleML will eventually be accepted as part of the W3C standard for Rule-based ontology in Semantic Web. The early adoption of such emerging standard promotes standardization of our effort and allows our work to be re-used by other interested parties in the Semantic Web and data/context integration space.

3.3 Notational Conventions

A number of namespace prefixes are used in the following sections as defined below. We attempt to adhere to the namespace prefix used in the OWL Web Ontology Language XML Presentation Syntax [17] for uniformity and readability. As in the OWL documentations, note that the choice of the namespace prefix is arbitrary, and not semantically significant.

Prefix	**Namespace**	**Notes**
rdfs	“http://www.w3.org/2000/01/rdf-schema#”	The namespace of the RDF Schema
owl	"http://www.w3.org/2002/07/owl#"	The namespace of OWL in RDF/XML syntax
xsd	"http://www.w3.org/2001/XMLSchema#"	The namespace of the XML Schema
coin	“http://context2.mit.edu/coin#”	The namespace of the proposed COIN-OWL in RDF/XML syntax

3.4 COIN ontology with OWL and RuleML

In this section, we will examine the modeling of the COIN ontology in OWL with respect to domain model, elevation theory and context theory. Where appropriate, additional categorization may be introduced to facilitate the presentation of the design. In addition to the abstract model, the concrete XML representation of the model is presented to illustrate the proposed implementation of the model. The complete listing of the concrete XML representation of the ontology model is available from Appendix A.

As a valid OWL ontology of COIN itself, this model can be used as a base OWL ontology to model disparate data sources for the purpose of data integration by means of context mediation. The approach for realization of this strategy is to maintain the COIN ontology as an independent OWL ontology which can be imported into other OWL ontology.

3.4.1 Domain Model

By definition, domain model provides define the taxonomy of the domain in terms of the available semantic types and modifiers to each semantic types. In addition, the notion of primitive type is used to represent the data types that are native to the source or receiver context.

OWL uses the facilities of XML Schema Datatypes and a subset of the XML Schema datatypes as its standard datatypes (or equivalently, its primitive datatypes). On the other hand, the primitive types in the COIN language consist of *string* and *number*. Trivially, the datatypes can be represented using its counterparts in OWL, namely *xsd:string* and *xsd:int, xsd:float* or *xsd:double*.

Figure 3-2 shows the class diagram of the components in the COIN-OWL domain model. We will inspect the design of each of the classes in details subsequently.

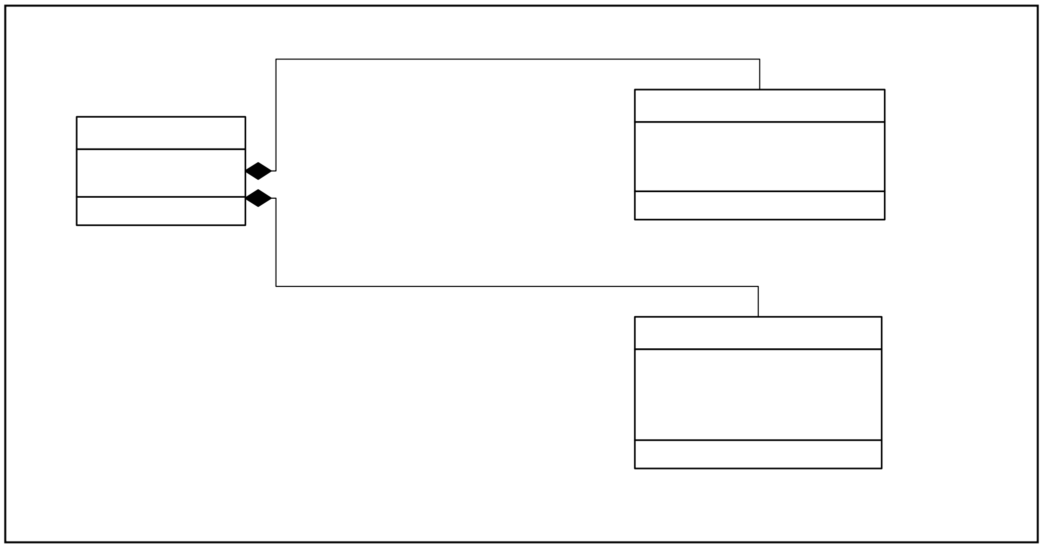


Figure 3-2: Class diagram of the Domain Model in COIN-OWL

Semantic Type

Types may be related in an abstraction hierarchy where properties of a type are inherited. For rich semantic type in the COIN framework, the *basic* type is the universal parent of which all semantic type is derived from. We proposed the use of *class* element for the modeling of a semantic type as it provides the construct for sub-classing, allowing inheritance of semantic type where needed.

Each semantic type is associated by attributes and modifiers which are modeled as ObjectProperty of the semantic type.

```
<owl:Class rdf:ID="SemanticType" />
<owl:ObjectProperty rdf:ID="Modifiers">
  <rdfs:domain rdf:resource="#SemanticType"/>
  <rdfs:range rdf:resource="#Modifier"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Attributes">
  <rdfs:domain rdf:resource="#SemanticType"/>
  <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>
```

As the running example in throughout this section, we have a semantic type called companyFinancials that represents the financial information of a company. This can be represented using the following proposed OWL constructs:

```
<coin:SemanticType rdf:ID="companyFinancials">
</coin:SemanticType>
```

Attribute

Each semantic type may contain one or more attributes, which describe the state of a semantic object[1] or the relationship between semantic types. Attribute is modeled as a class, and attached to semantic type as ObjectProperty.

```
<owl:Class rdf:ID="Attribute" />
  <owl:DatatypeProperty rdf:ID="AttributeName">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="AttributeFrom">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="#SemanticType"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="AttributeTo">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="#SemanticType"/>
  </owl:ObjectProperty>
```

[1] Instances of semantic types are called semantic objects

The companyFinancials semantic type has two attributes, namely company and fyEnding. To model this in COIN-OWL semantics:

```
<coin:SemanticType rdf:ID="companyFinancials">
  <coin:Attributes>
    <coin:Attribute rdf:ID="att_company">
      <coin:AttributeTo rdf:resource="#companyName"/>
      <coin:AttributeFrom rdf:resource="#companyFinancials"/>
      <coin:AttributeName rdf:datatype="&xsd;string">company</coin:AttributeName>
    </coin:Attribute>
    <coin:Attribute rdf:ID="att_fyEnding">
      <coin:AttributeTo rdf:resource="#date"/>
      <coin:AttributeFrom rdf:resource="#companyFinancials"/>
      <coin:AttributeName rdf:datatype="&xsd;string">fyEnding</coin:AttributeName>
    </coin:Attribute>
  </coin:Attributes>
</coin:SemanticType>
```

Modifier

Similarly, each semantic type may be modified by one or more modifiers. Each modifier in turns relates the semantic type to another semantic type.

```
<owl:Class rdf:ID="Modifier"/>
<owl:DatatypeProperty rdf:ID="ModifierName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Modifier"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="ModifierFrom">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Modifier"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ModifierTo">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Modifier"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ModifierValues">
  <rdfs:range rdf:resource="#ModifierValue"/>
  <rdfs:domain rdf:resource="#Modifier"/>
</owl:ObjectProperty>
```

`companyFinancial` semantic type is modified by two modifiers, `scaleFactor` and `currency`, which augment the `companyFinancial` semantic type to the scale factor (semantic type of `basic`) and the currency (semantic type of `currencyType`) in which the financial data is stored. Using COIN-OWL, this information is represented as:

```
<coin:Modifier rdf:ID="mod_scaleFactor">
  <coin:ModifierName rdf:datatype="&xsd;string">scaleFactor</coin:ModifierName>
  <coin:ModifierTo rdf:resource="#basic"/>
  <coin:ModifierFrom rdf:resource="#companyFinancials"/>
</coin:Modifier>

<coin:Modifier rdf:ID="mod_currency">
  <coin:ModifierName rdf:datatype="&xsd;string">currency</coin:ModifierName>
  <coin:ModifierTo rdf:resource="#currencyType"/>
  <coin:ModifierFrom rdf:resource="#companyFinancials"/>
</coin:Modifier>

<coin:SemanticType rdf:ID="companyFinancials">
  <coin:Modifiers rdf:resource=”#mod_scaleFactor”></coin:Modifiers>
  <coin:Modifiers rdf:resource=”#mod_currency”></coin:Modifiers>
</coin:SemanticType>
```

From these two examples of attributes and modifiers, we can see that references to other classes/objects can be made by either instantiating the classes directly within the parent class (as with attributes) or instantiating the classes outside of the parent class, and linked to the parent class using the rdf:resource construct. Note that both ways create semantically equivalent and syntactically well-formed OWL ontology document. This difference in concrete representation of the domain is usually caused by ontology tools that generate concrete ontology automatically, and should not be of an ontology administrator’s concern.

3.4.2 Source Sets

A COIN concept not available in OWL is the intensional description of the data sources, as OWL is used as the descriptive language only for semi-structured data on the Web. COIN, on the other hand, is designed to deal with a wide range of data sources, which makes the declarative description of the data sources indispensable for data integration and context mediation.

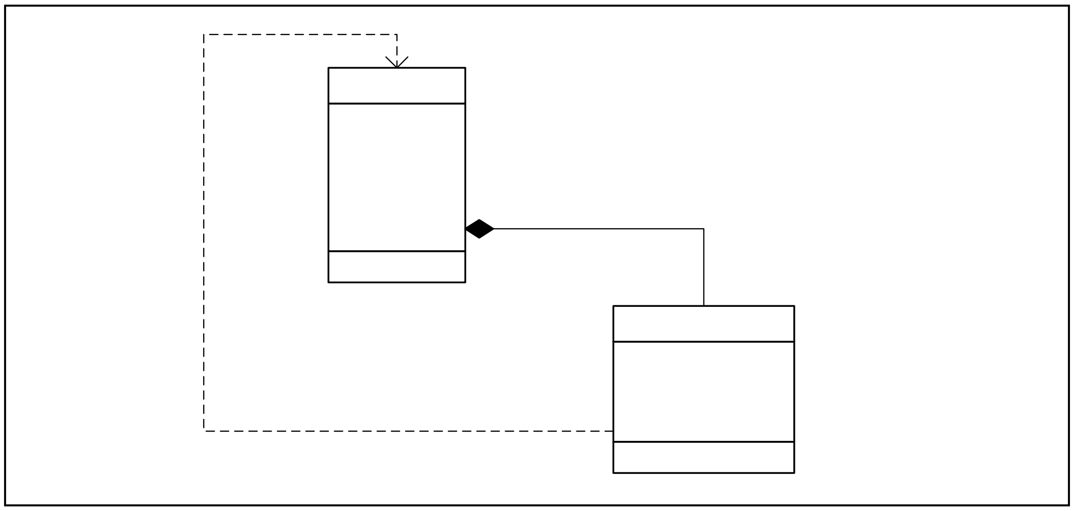


Figure 3-3: Class diagram of the Source Sets in COIN-OWL

Relation

For this purpose, we introduce the notion of a Relation class in the model. Conceptually, this construct is really part of the auxiliary ontology of an ontology – the facility describes the various data sources that consume the core ontology. This piece of information is used in the COIN query optimization and execution engine.

```
<owl:Class rdf:ID="Relation" />
<owl:DatatypeProperty rdf:ID="DataSource">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="RelationName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Type">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Bindings">
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Operators">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
<owl:ObjectProperty rdf:ID="Columns">
  <rdfs:range rdf:resource="#Column"/>
  <rdfs:domain rdf:resource="#Relation"/>
</owl:ObjectProperty>
```

Column

Following the relational data model, each relation consists of one or more columns. As each column is unique to its associated relation, we have a one-to-one relationship between Column and Relation class. Using OWL's Inverse Functional Property, we create an AssociatedRelation property on the Column class to relate each column to the relation it belongs to.

Note that using an ontology editor supporting OWL, we will not need to fill this field. The AssociatedRelation property is simply the inverse of Relation class' Columns property. This is one of the many OWL rich language feature set that we leveraged on.

The concrete OWL/RDF syntax of the Column class is presented below.

```
<owl:Class rdf:ID="Column" />
<owl:Class rdf:ID="Column" />
<owl:DatatypeProperty rdf:ID="ColumnName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Column"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="DataType">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Column"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ColumnIndex">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Column"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="AssociatedRelation">
  <rdfs:domain rdf:resource="#Column"/>
  <owl:inverseOf rdf:resource="#Columns" />
</owl:ObjectProperty>
```

To model a relation and the columns making up of the relation, we instantiate the `coin:Relation` class, and subsequently the `coin:Columns`. For example, the lookup table relation `countryIncorp` stores the company name and the country the company is incorporated in the `COMPANY_NAME` and `COUNTRY` columns respectively:

```
<coin:Relation rdf:ID="rel_countryIncorp">
  <coin:Type rdf:datatype="&xsd;string">e</coin:Type>
  <coin:RelationName rdf:datatype="&xsd;string">countryIncorp</coin:RelationName>
  <coin:Bindings rdf:datatype="&xsd;string">0,0</coin:Bindings>
  <coin:DataSource rdf:datatype="&xsd;string">view</coin:DataSource>
  <coin:Columns>
    <coin:Column rdf:ID="col_countryIncorp_COMPANYNAME">
    <coin:ColumnIndex rdf:datatype="&xsd;int">1</coin:ColumnIndex>
    <coin:ColumnName rdf:datatype="&xsd;string">COMPANY_NAME</coin:ColumnName>
    <coin:AssociatedRelation rdf:resource="#rel_countryIncorp"/>
    <coin:DataType rdf:datatype="&xsd;string">string</coin:DataType>
  </coin:Column>
  </coin:Columns>
  <coin:Columns>
    <coin:Column rdf:ID="col_countryIncorp_COUNTRY">
      <coin:ColumnName rdf:datatype="&xsd;string">COUNTRY</coin:ColumnName>
      <coin:ColumnIndex rdf:datatype="&xsd;int">2</coin:ColumnIndex>
      <coin:AssociatedRelation rdf:resource="#rel_countryIncorp"/>
      <coin:DataType rdf:datatype="&xsd;string">string</coin:DataType>
    </coin:Column>
  </coin:Columns>
</coin:Relation>
```

Constraints

Another concept contained in the source set is constraint. Constraints, consisting of integrity constraints and general constraints, are used as:

- key constraints that express the keys of a primitive relation
- foreign key constraints constraining the links between relations
- general constraints that may involve semantic conflicts

Implemented using Constraint Handling Rules (CHR), these constraints are used for query optimization in the query processing phase. In our current work, the constraints are not modeled in OWL. Instead, the rules are represented at its native format of CHR.

3.4.3 Context Axioms

One core prevailing concept in COIN is the notion of context differential and the ability to interoperate among contexts through context mediation. The fundamental component to context axioms is the definition of context itself.

Context

Context is simply modeled as an OWL class. The only property pertinent for the class is name of the context, stored in the ContextName property.

```
<owl:Class rdf:ID="Context"/>
<owl:DatatypeProperty rdf:ID="ContextName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
   <rdfs:domain rdf:resource="#Context"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

In our running example, we have different contexts for different relations, due to the source of the data. For example, the data from worldscope can be modeled as having its own context, `worldscope`:

```
<coin:Context rdf:ID="context_worldscope">
  <coin:ContextName rdf:datatype="&xsd;string">worldscope</coin:ContextName>
</coin:Context>
```

Modifier Values

The interpretation of a semantic object value that is decorated by modifiers may vary according to the values taken by the modifier. The value of the modifier is determined by prior domain knowledge, dependent on the context of the domain. This value can either be a static literal (number or string), or dynamically obtained from other attributes. This hierarchical structure translates to the need of modeling a parent ModifierValue class, with two subclasses ModifierStaticValue and ModifierDynamicValue.

```
<owl:Class rdf:ID="ModifierValue" />
<owl:Class rdf:ID="ModifierStaticValue">
  <rdfs:subClassOf rdf:resource="#ModifierValue" />
</owl:Class>
<owl:Class rdf:ID="ModifierDynamicValue">
  <rdfs:subClassOf rdf:resource="#ModifierValue" />
```

```
</owl:Class>
```

Ideally, in a strict object modeling framework, the ModifierValue class should be an abstract class, in which no object can be instantiated from this parent class. Instead, all modifier values can only either be a ModifierStaticValue or ModifierDynamicValue object.

To illustrate this, consider the currency modifier for semantic type companyFinancials. All financial data are expressed in USD in the worldscope context:

```
<coin:Modifier rdf:ID="mod_currency">
  <coin:ModifierValues>
    <coin:ModifierStaticValue rdf:ID="modval_currency_worldscope">
    <coin:ModifierContext rdf:resource="#context_worldscope"/>
    <coin:ModifierSemanticType rdf:resource="#companyFinancials"/>
    <coin:ModifierStringValue rdf:datatype="&xsd;string">USD</coin:ModifierStringValue>
    <coin:ModifierObject rdf:resource="#mod_currency"/>
  </coin:ModifierStaticValue>
</coin:ModifierValues>
```

In the datastream context, however, the financial data are stored based on the location of incorporation of the companies. This requires a cross-reference to other attributes to retrieve the required information. Using our COIN-OWL ontology model, this can be modeled as:

```
<coin:ModifierValues>
  <coin:ModifierDynamicValue rdf:ID="modval_currency_disclosure">
    <coin:ModifierContext rdf:resource="context_disclosure"/>
    <coin:ModifierSemanticType rdf:resource="companyFinancials"/>
    <coin:ModifierDynamicValues rdf:resource="att_countryIncorp"/>
    <coin:ModifierDynamicValues rdf:resource="att_officialCurrency"/>
    <coin:ModifierDynamicValues rdf:resource="att_company"/>
    <coin:ModifierObject rdf:resource="mod_currency"/>
  </coin:ModifierDynamicValue>
</coin:ModifierValues>
```

In English, this means that the value of the currency modifier for companyFinancials semantic type in the datastream context is retrieved from the values of countryIncorp, officialCurrency and company attributes.

Conversion Functions

A more complex construct available in COIN is the conversion function. In essence, conversion functions enable interoperability of semantic objects across different contexts. This is achieved by defining generic conversion rules for each semantic type that may yield different value under different contexts.

This requirement calls for a language facility that is both flexible and supports rule-based data. However, OWL lacks the ability to model rules in an extensible manner. Therefore, we recommend the use of RuleML for conversion functions modeling.

As an example, consider the simple conversion function in eCOIN's Prolog representation, that converts the month in words into numeric value (and vice versa):

```
rule(month("Jan", 01), (true)).
```

This rule can be represented using RuleML as follows:

```
<fact>
  <_head>
    <atom>
      <cterm>
        <_opc><ctor>rule</ctor></_opc>
        <cterm>
          <_opc><ctor>month</ctor></_opc>
          <ind>Jan</ind>
          <ind>01</ind>
        </cterm>
        <ind>true</ind>
      </cterm>
    </atom>
  </_head>
</fact>
```

For the complete concrete syntax of RuleML used, readers are referred to the earlier section of 2.3 Rule Markup Language (RuleML).

The figure below shows the relationship among the classes participating in the context axioms.

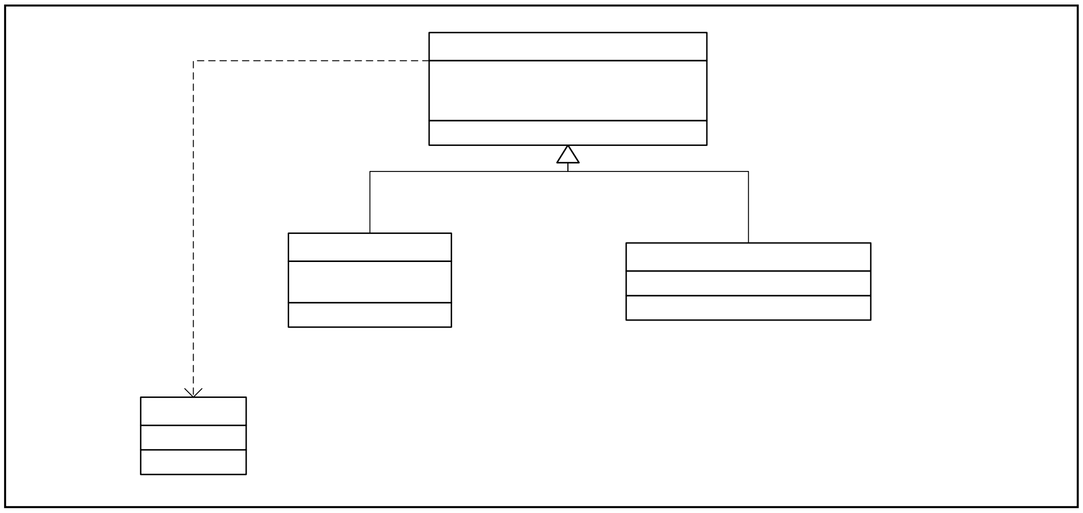


Figure 3-4: Class diagram of Context Axioms in COIN-OWL

3.4.4 Elevation Axioms

Elevation axioms are used to describe the functional relationship between data sources and domain model. Intuitively, the elevation axioms can be viewed as the mapping of the primitive relation to its semantic relation. At the lower level, each column and data cell are mapped to their semantic counter part via skolemization[2].

Skolemization

Skolemization is done as part of the primitive to semantic relation elevation process. This however, is a low level process for context mediation and resolution, and should be abstracted from the ontology editor/administrator at the application level.

Relation Elevation

For this reason, we propose a more intuitive view of the elevation process by introducing SemanticRelation and SemanticColumn classes in COIN-OWL. Intuitively, a primitive relation is

[2] In logic programming, skolemization is the process of removing existential quatifiers by existantial instantiation.

Context

-ContextNa

elevated (or mapped) to a semantic relation by mapping each of the columns to a semantic type. SemanticRelation class essentially represents the semantic relation elevated from the primitive relation, and contains the semantic mapping between the primitive column and its semantic type in its original context.

```
<owl:Class rdf:ID="SemanticRelation"/>
<owl:ObjectProperty rdf:ID="ElevationContext">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticRelation"/>
  <rdfs:range rdf:resource="#Context"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SemanticColumn" />
<owl:ObjectProperty rdf:ID="SourceColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticColumn"/>
  <rdfs:range rdf:resource="#Column"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="TargetSemanticType">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticColumn"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>
```

Attribute Elevation

Each semantic type is decorated with one or more attributes. The attributes may refer to the same semantic relation of the domain (simple attribute) or from other relations (complex attribute). Attribute elevation provides the mapping between the attribute of a semantic column and the referring semantic column.

To model the hierarchical difference between the simple and complex attributes, we leverage on the inheritance facility of OWL by using a SemanticAttribute class, and a specialized ComplexSemanticAttribute classes to model each type of the attributes. The difference between SimpleAttribute and ComplexSemanticAttribute lies in the need for a construct to express the relationship between two or more relations involved in the join definition.

```
<owl:Class rdf:ID="SemanticAttribute" />
<owl:ObjectProperty rdf:ID="SourceSemanticColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticAttribute"/>
  <rdfs:range rdf:resource="#SemanticColumn"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="TargetSemanticColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticAttribute"/>
  <rdfs:range rdf:resource="#SemanticColumn"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="ComplexSemanticAttribute">
  <rdfs:subClassOf rdf:resource="#SemanticAttribute" />
</owl:Class>
<owl:ObjectProperty rdf:ID="JoinValues">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#ComplexSemanticAttribute"/>
  <rdfs:range rdf:resource="#SemanticRelation"/>
</owl:ObjectProperty>
```

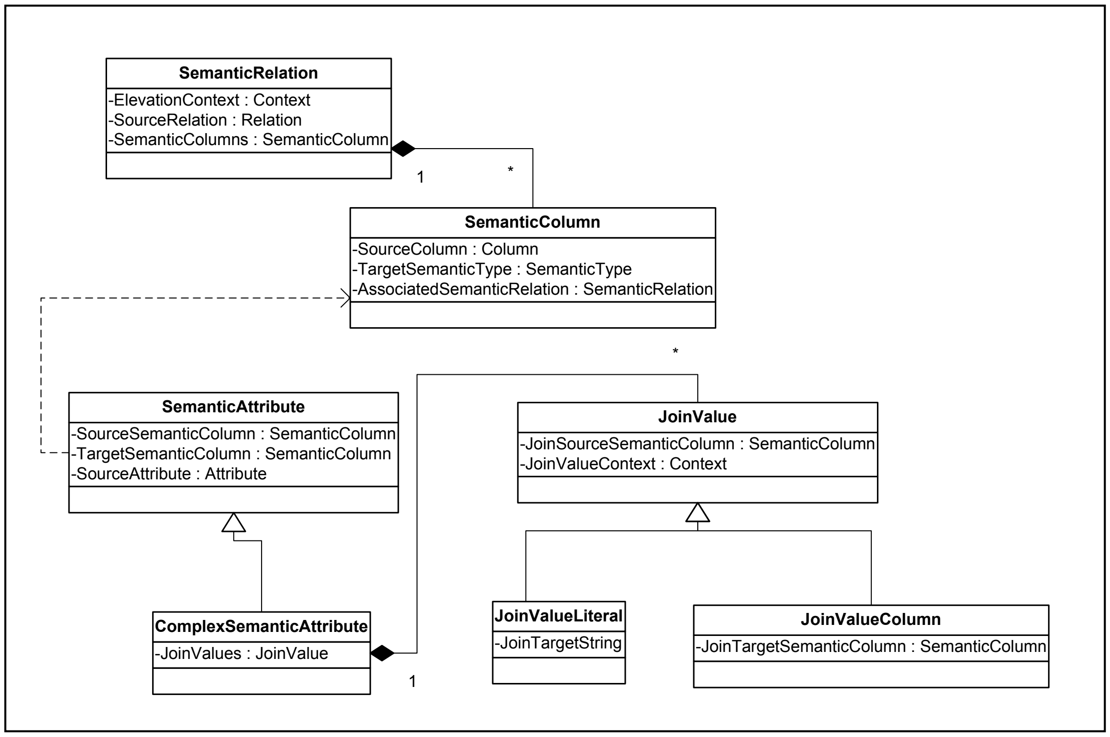

SemanticRelation
-ElevationContext : Context
-SourceRelation : Relation
-SemanticColumns : SemanticColumn
1
*
SemanticColumn
-SourceColumn : Column
-TargetSemanticType : SemanticType
-AssociatedSemanticRelation : SemanticRelation
*
SemanticAttribute
-SourceSemanticColumn : SemanticColumn
-TargetSemanticColumn : SemanticColumn
-SourceAttribute : Attribute
JoinValue
-JoinSourceSemanticColumn : SemanticColumn
-JoinValueContext : Context
ComplexSemanticAttribute
-JoinValues : JoinValue
1
JoinValueLiteral
-JoinTargetString
JoinValueColumn
-JoinTargetSemanticColumn : SemanticColumn

Figure 3-5: Class diagram of the Elevation Axioms in COIN-OWL

3.4.5 Summary of COIN-OWL Ontology Model

In this section, we revisit the artifacts introduced in the COIN-OWL ontology model from the previous sub-sections.

The following table summarizes the various OWL constructs used in the COIN-OWL ontology model.

COIN Concepts	**OWL/RuleML Artifacts**	**Notes**
Domain Model		
Semantic Type	coin:SemanticType	
Attribute	coin:Attribute	
Modifier	coin:Modifier	
Source Sets		
Relation	coin:Relation	
	coin:Column	coin:Column artifact has been introduced as a structured solution to the source set model. A coin:Relation consists of one or more coin:Columns.
Constraints	-	Constraints are used in query optimization, and not implemented in the current release of the work. Instead, the constraints can be coded directly in the constraint handling rules (CHR) file.
Context Axioms		
Context	coin:Context	
Modifier value assignments	coin:ModifierValues, coin:ModifierStaticValue, coin:ModifierDynamicValue	
Conversion functions	RuleML rules	
Elevation Axioms		
Relation elevation	coin:SemanticRelation	
	coin:SemanticAttribute, coin:ComplexSemanticAttribute	

Table 3-1: Summary of COIN-OWL ontology model

As recapitulation, the complete ontology model of eCOIN is presented in Figure 3-6 Complete UML class diagram of COIN-OWL model.

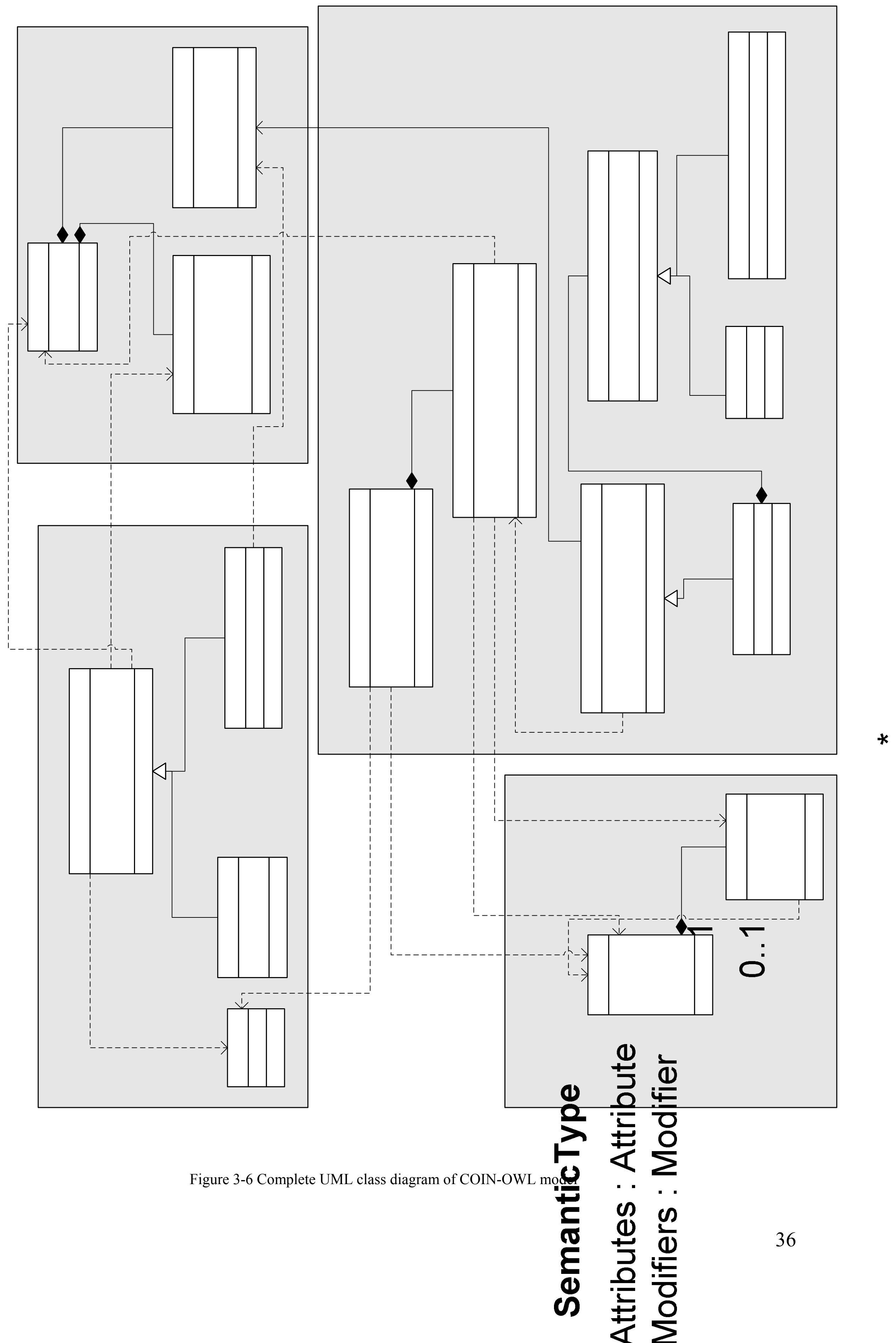


Figure 3-6 Complete UML class diagram of COIN-OWL model

3.5 Design Considerations

The objective of the project is to adopt emerging W3C standards as the data exchange standard in the Context Interchange project while reusing the established context mediation strategy in the project. This means that the proposed model in OWL must be able to translate to FOL/Prolog for actual context mediation and query execution process. This guiding principal is crucial in ensuring the practicality of the proposed model.

In terms of ontology modeling, OWL provides a wider range of language constructs as compared to Prolog. With good understanding of the Context Interchange framework and the language features of OWL, a number of standard OWL constructs are employed in the design to implement some functional requirements that were previously unable to implement in eCOIN.

3.5.1 Choice of OWL Sublanguage

As introduced in the earlier section, OWL is classified into three language family: OWL Lite, OWL DL and OWL Full. The OWL sub-language used in our design is OWL Lite, as this family of language is sufficiently expressive to represent the COIN ontology model.

With our three-tier architecture, the actual reasoning and context mediation is performed at the backend (see Figure 3-1). This essentially means that the computation guarantee of OWL Lite and OWL DL is not required. In other words, we have the liberty to use any of these three classes of OWL sublanguages.

However, OWL Lite contains the language contructs that are rich enough for this purpose. One reason for not pushing to use the upper language family of OWL DL and OWL Full is to preserve the computability of the ontology for future. This allows the reasoning and context mediation, should there be a need in the future, to be performed directly at the OWL level without having to first translate the OWL ontology to the native eCOIN Prolog application.

3.5.2 OWL Ontology and Data

As part of the design direction and the operation model of COIN, we have a slightly different usage adoption of OWL. In the standard usage of OWL for ontology modeling, the ontology and data are both stored in OWL. Depending on the generality of the taxanomy definition, the ontology and data may co-exist on the same OWL document. In other cases, the ontology is defined and stored in a central OWL ontolog library, and referenced in the OWL data document using external namespace reference.

An example the such usage is the OWL Wine ontology (http://www.w3.org/TR/2002/WD-owl-guide-20021104/wine.owl), where both the ontology definition and the individual instantiation (i.e. actual data) are stored in the same OWL document.

On the other hand, COIN utilize the application ontology in a slightly different manner. The COIN-OWL ontology model describes the context semantics of the data sources. Modeled in OWL, this ontology is then used by the context mediation engine to resolve context disparities among the data sources.

While the COIN ontology is modeled in OWL, the actual data may not neccesarily be stored in OWL. This is because by design, COIN is architected to solve the heterogeneous data source interoperability problem. This means that the data dealt by COIN will be from disparate data sources, comprising traditional relational databases or semi-structured data sources such as the World Wide Web or even OWL.

3.5.3 Static Type Checking

One of the biggest differences between modeling the ontology in eCOIN and COIN-OWL is the strongly enforced typing facility in OWL. In OWL, all ObjectProperty and DataProperty requires the formal definition of the range of the property, i.e. the type of object that can be specified in property.

As an example, in eCOIN, we model semantic types and modifiers using the following constructs:

```
rule(semanticType(companyName), (true)).
rule(semanticType(companyFinancials), (true)).
rule(modifiers(companyFinancials, [scaleFactor, currency]), (true)).
```

Here, it is possible for someone to accidentally put companyName as the modifier for companyFinancials:

```
rule(semanticType(companyName), (true)).
rule(semanticType(companyFinancials), (true)).
rule(modifiers(companyFinancials, [companyName]), (true)).
```

However, as all classes are strongly typed in OWL, the following ontology will yield an error when validated against the COIN ontology:

```
<coin:SemanticType rdf:ID="companyName" />
<coin:SemanticType rdf:ID="companyFinancials">
  <coin:Modifiers rdf:resource=”#companyName”>
</coin:SemanticType>
```

3.5.4 Functional Property

In all flavors of OWL (OWL Lite, OWL DL and OWL Full), a property *P* of object *X* can be tagged as functional such that for objects *Y* and *Z*, *X.P*=*Y* and *X.P*=*Z* implies *Y*=*Z*. property *P* of object *X* is denoted as *X.P*.

In other words, object *X* can functionally determine *Y* in `X.P=Y`. Using this language feature, we can enforce a many-to-one relationship between classes. Given the wide array of language features in OWL, this is particularly useful in enforcing syntactically and semantically correct COIN ontology:

```
<owl:Class rdf:ID="Context"/>
<owl:DatatypeProperty rdf:ID="ContextName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Context"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

The use of FunctionalProperty here constraints that a context may have only one name.

As an additional note, such requirements can also be enfored using the owl:cardinality construct. However, it is worth noting that the use of this construct depends on the sublanguage family of OWL. Cardinality expressions with values limited to 0 or 1 are part of OWL Lite. This permits the user to indicate 'at least one', 'no more than one', and 'exactly one'. Positive integer values other than 0 and 1 are permitted in OWL DL. owl:maxCardinality can be used to specify an upper bound. owl:minCardinality can be used to specify a lower bound. In combination, the two can be used to limit the property's cardinality to a numeric interval.

3.5.5 RuleML for Rules Modeling

In the previous work in [20], RDF was used to model the COIN ontology model. However, the work was unable to address the need for an more extensible framework in rules representation. In particulars, conversion rules were encoded as raw string in the RDF document:

```
<coin:Ont_ModifierConversionFunction>
  convfunc|rule(cvt(companyFinancials, O, currency, Ctxt, Mvs, Vs, Mvt, Vt), (attr(O,
  fyEnding, FyDate), value(FyDate, Ctxt, DateValue), olsen_p(Fc, Tc, Rate, TxnDate),
  value(Fc, Ctxt, Mvs), value(Tc, Ctxt, Mvt), value(TxnDate, Ctxt, DateValue),
  value(Rate, Ctxt, Rv), Vt is Vs * Rv)).
  ...
  rule(month("Oct", 10), (true)).
  rule(month("Dec", 12), true)). |companyFinancials|currency
</coin:Ont_ModifierConversionFunction>
```

These rules were then extracted programmatically from the RDF document and used in context mediation. In comparison, the adoption of RuleML for rules modeling provided a cleaner method for this purpose. In COIN-OWL, these rules are stored as RuleML:

```
<rulebase>
  <!-- rule(month("Apr", 04), (true)). -->
  <fact>
    <_head>
      <atom>
        <cterm>
          <_opc><ctor>rule</ctor></_opc>
          <cterm>
            <_opc><ctor>month</ctor></_opc>
            <ind>Apr</ind>
            <ind>04</ind>
          </cterm>
          <ind>true</ind>
        </cterm>
      </atom>
    </_head>
  </fact>
  <!-- rule(cvt(companyFinancials, _O, scaleFactor, Ctxt, Mvs, Vs, Mvt, Vt),
    (Ratio is Mvs / Mvt, Vt is Vs * Ratio)). -->
  <fact>
    <_head>
      <atom>
        <cterm>
          <_opc><ctor>rule</ctor></_opc>
          <cterm>
            <_opc><ctor>cvt</ctor></_opc>
            <ind>companyFinancials</ind>
            <var>_O</var>
            <ind>scaleFactor</ind>
            <var>Ctxt</var>
            <var>Mvs</var>
            <var>Vs</var>
            <var>Mvt</var>
            <var>Vt</var>
          </cterm>
          <cterm>
            <_opc><ctor/></_opc>
            <var>Ratio is Mvs / Mvt</var>
            <var>Vt is Vs * Ratio</var>
          </cterm>
        </cterm>
      </atom>
    </_head>
  </fact>
</rulebase>
```

While this format may look more elaborate, this mode of representation adheres to the publicly accepted RuleML language constructs, and thus allow re-use and interchange of rules easily.

3.5.6 Reasoning in OWL and RuleML

A list of currently available resoning engine is available from [7]. As of now, the reasoning engines for OWL mainly focus on taxanomy classification, consistency checking, and limited query answering based on OWL ontology. This is partially due to the fact that OWL has limited support for rule based expressions, different from the capability provided by RuleML for rules modeling.

On the other hand, we believe that with the stabilization of RuleML, more powerful inference and query answering engine will be available. This may open the door for context mediation and query processing directly at the RuleML level, consolidating the three tier structure of the design (Figure 3-1) into a single implementation platform.

However, this require the entire COIN ontology to be in a single cohesive OWL/RuleML document for the reasoning to take place. A possible technology for this is the Semantic Web Rule Language (SWRL), which is a language proposal based on a combination of OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language (OWL) with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language (RuleML). One of the ongoing work of the Pellet OWL Reasoner [4] is the support for rules reasoning for SWRL.

3.5.7 Semantic Web Rule Language (SWRL)

We noted that in parallel with the development of RuleML, a number of relevant emerging standards have been branched from RuleML, including RuleML Lite and Semantic Web Rule Language (SWRL). RuleML Lite adopts an integrated concrete syntax of XML and RDF, expanding the language construct available in modeling rules. This opens up possibility of a tighter integration between the conversion rules in RuleML and the core ontology in OWL. One possibility is to refer to the entities modeled in the OWL ontology using rdf:resource or href attributes, instead of treating the same entity in both documents as individual and disjoint entities in each of the document.

SWRL has been considered but not implemented in the project as the modeling language is still in its infancy stage. SWRL is the result of an effort to integrate RuleML into OWL, and hence holds a more holistic view of rules and ontology in the Semantic Web, compared to the use of OWL and RuleML separately.

An example of this can be drawn from the SWRL Language Specification Proposal [18]. In section 5, example 5.1-2, which formulate the rule hasUncle, asserting that if x1 hasParent x2, x2 hasSibling x3, and x3 hasSex male, then x1 hasUncle x3:

```
<ruleml:imp>
  <ruleml:_rlab ruleml:href="#example2"/>
  <ruleml:_body>
    <swrlx:individualPropertyAtom  swrlx:property="hasParent">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasSibling">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom  swrlx:property="hasSex">
      <ruleml:var>x3</ruleml:var>
      <owlx:Individual owlx:name="#male" />
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom  swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

From the example, we note that the OWL ontology and RuleML rules are all modeled in one cohesive SWRL document. The highlighted rules fragment that expresses `x3 hasSex male` refers to the OWL class `male` seamlessly using the `owlx:Individual` construct:

```
<swrlx:individualPropertyAtom  swrlx:property="hasSex">
  <ruleml:var>x3</ruleml:var>
  <owlx:Individual owlx:name="#male" />
</swrlx:individualPropertyAtom>
```

In RuleML 0.8, the RuleML version used in the COIN-OWL ontology model, such language facility is not available. To refer to an individual defined in the OWL ontology, there is no other ways but to initialize a new individual in the RuleML rules document, hence creating a slight gap between the OWL ontology and RuleML rules.

CHAPTER 4
COIN-OWL Implementation Strategy

Following from the design in the previous section, the next step is to devise an implementation strategy for context mediation in OWL. In this section, we outline our recommendations for the implementation of context mediation, and conclude with our work on a prototype developed as a reference implementation for the COIN-OWL model.

4.1 Ontology Modularization

COIN-OWL schema

To encourage reusability, the COIN-OWL model schema should be maintained on a centralized repository. Using the import facility in OWL, the model schema can then be referenced from the application ontology using the `owl:imports` construct. Assuming the COIN-OWL model is stored on `http://context2.mit.edu/coin.owl`, the model can be imported as:

```
<owl:Ontology rdf:about="file:/C:/application.owl">
  <owl:imports rdf:resource="http://context2.mit.edu/coin.owl"/>
</owl:Ontology>
```

We recommend using the `xmlns:coin` namespace as the reference namespace for COIN applications.

OWL Ontology

Upon importing the central COIN-OWL model, the application domain ontology can be developed and stored in application specific ontology file. Using the `xmlns:coin` namespace, the classes in COIN-OWL model can be instantiated in the application simply by prefixing the COIN-OWL classes with `coin:`. For example, to create the company attribute, we can simply refer the Attribute class using the following constructs:

```
<coin:Attribute rdf:ID="att_company">
  <coin:AttributeName rdf:datatype="&xsd;string">company</coin:AttributeName>
</coin:Attribute>
```

Rules

Rules such as conversion functions, modifier values assignment and other auxiliary rules are proposed to be modeled using RuleML. One drawback with this option is that RuleML is yet a unified element of Semantic Web/OWL. For this reason, the various RuleML constructs and its concrete syntax are not part of the OWL syntax.

The implementation option is to model and represent these rules in a separate RuleML document, and pass the unification functionality of the OWL ontology and RuleML rules to the application layer. This is addressed in the prototype developed, detailed in section 4.4 COIN-OWL Protégé Plugin.

4.2 Ontology Interoperability

The Context Interchange strategy is designed to solve the age-old problem of data integration. The emergence of standard ontology language such as OWL has however, created a similar problem at the ontology level. In fact, W3C recognizes the existence of such problem – "We want simple assertions about class membership to have broad and useful implications. …It will be challenging to merge a collection of ontologies." [24].

OWL provides a number of standard language constructs that aims at solving a subset of this problem. Ontology mapping constructs such as equivalentClass, equivalentProperty, sameAs, differentFrom and AllDifferent only allows ontology context consolidation at a very limited level. These language constructs are only useful if the consolidation effort requires only disambiguation between ontology. In other words, we can use these facility to tell that a human in ontology A is the same as person in ontology B, but if they are different, we will not be able to tell how different these two classes are, needless to say consolidating these two classes to enable interoperability between the two ontologies.

Although initiated nearly a decade ago, the Context Interchange strategy is still relevant at solving this problem, including the ontology disparity problem with OWL/Semantic Web. eCOIN application can be created in the COIN-OWL model based on the OWL ontology definition of the domain, using the same conventional process of eCOIN. The only requirement of the system is its relational view of the data sources, which require all data source be represented in relational data model. This however, can be solved easily by either using Cameleon or similar web wrapping engine.

4.3 OWL Ontology Development Platform

There is a wide array of ontology editors in the ontology modeling/knowledge representation community, although a fair number of these older generations of ontology editors are yet to support the W3C recommended OWL. For a sufficiently comprehensive list of generic ontology editors used in the industry and academic institutions, we refer the readers to the survey done by Denny [14] and OntoWeb [14]. Among these, editors that supports OWL include Protégé with OWL Plugin [6] and pOWL [5].

One OWL ontology editor that is becoming the de-facto Semantic Web ontology editor is the Protégé Editor. Protégé is an open-source development environment for ontology and knowledge-base system developed by Stanford Medical Informatics at the Stanford University School of

Medicine. Protégé OWL Plugin supports the editing and development of ontology using OWL. Protégé OWL Plugin enables an ontology administrator to:

- Load and save OWL and RDF ontologies
- Edit and visualize OWL classes and their properties
- Define logical class characteristics as OWL expressions
- Execute reasoners such as description logic classifiers
- Edit OWL individuals for Semantic Web markup

The open and extensible architecture of Protégé allows rapid development of Protégé plugins for additional feature sets, such as visual editor for OWL and ontology visual diagram. Figure 4-1 shows a screen capture of the editor's user interface.

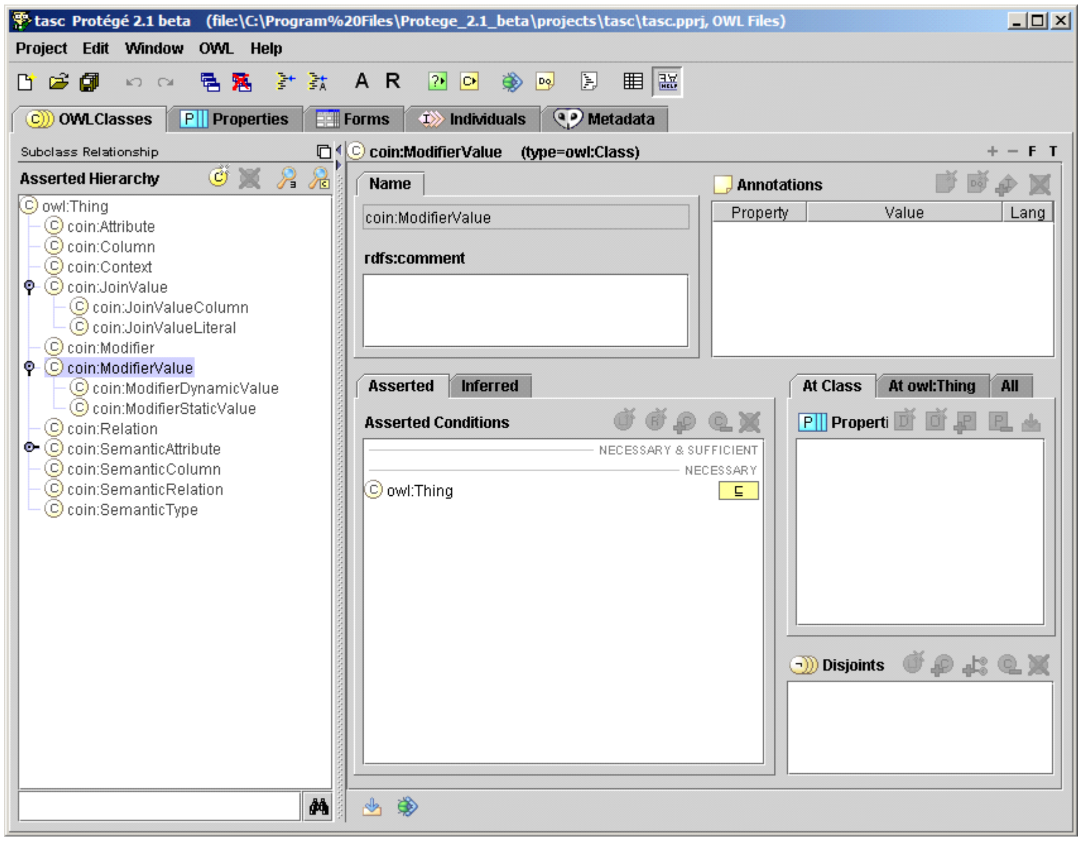


Figure 4-1: Protégé Ontology Editor with OWL Plugin

4.4 COIN-OWL Protégé Plugin

Building on Protégé's open application architecture, we have developed a COIN-OWL protégé plugin, functioning as a reference implementation for the COIN-OWL model. The Protégé plugin take as input the eCOIN application ontology file, written in OWL format, and a RuleML rules file containing application rules, and output a eCOIN Prolog program. This corresponds to step (2) in our approach in Context Interchange in OWL in section 3.1.

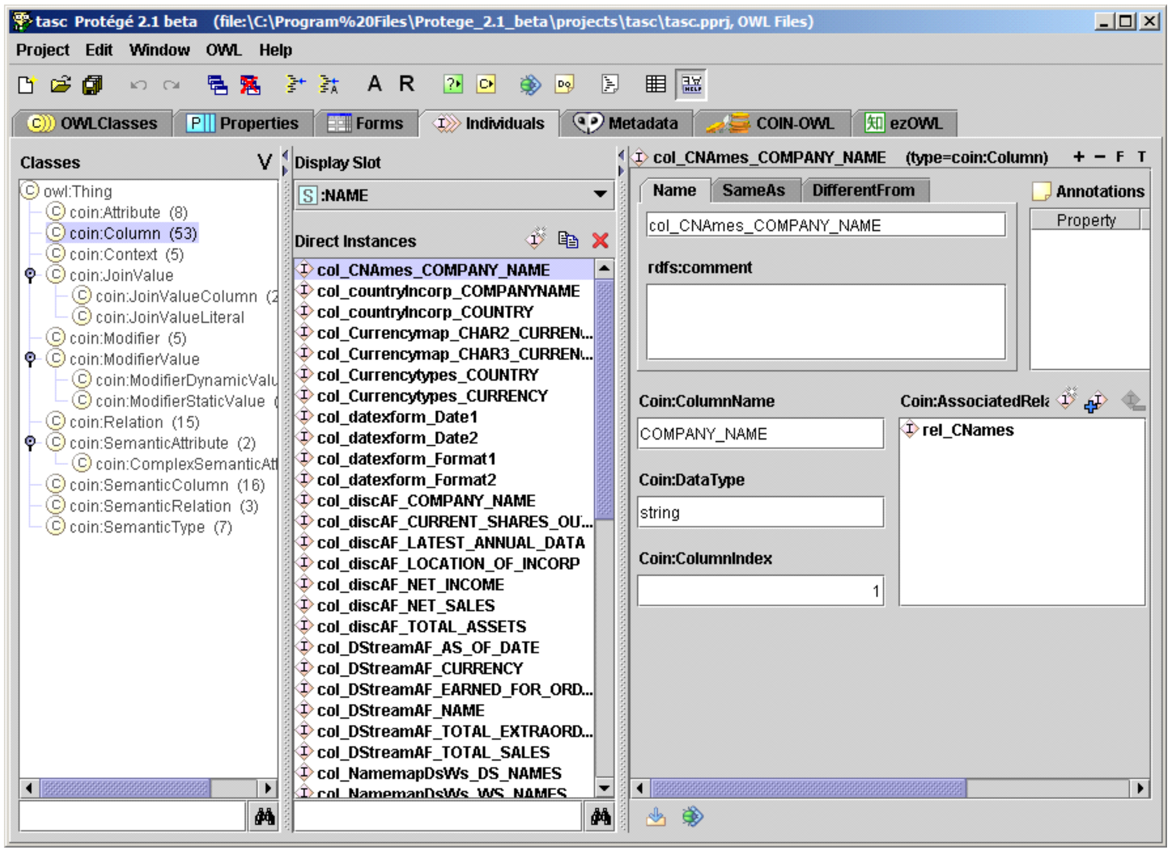
There are several viable approaches to achieving this. As OWL and RuleML are both XML documents, we can employ the eXtensible Stylesheet Language Transformation (XSLT) technology to transform both document into a Prolog program, much like [20]. One important prerequisite for this solution is that the XML documents must adhere to the predetermined format and layout. However, this is not guaranteed if the well-formed XML document (OWL and RuleML) may contain undetermined positional layout, as is the case with using Protégé as the OWL development environment.

Our approach is to leverage on the OWL application programming interface avail to us from Protégé OWL plugin. This allows us to correctly extracts the ontology information from the application OWL ontology, process and translate it to the eCOIN Prolog format in a precise manner.

As for the transformation of RuleML, we currently adopt the first approach presented, using XSLT technology. This is due to the fact that the RuleML XML syntax is realized in XML, not RDF. This imposes a more rigid structure on the rules and ensures that the document adheres to the format/layout required by the XSL stylesheet. Figure 4-3 shows the screen capture of the developed prototype.

It is worth noting that the developed XSL stylesheet for RuleML to Prolog transformation (see Appendix B) can be used independently as an individual component to translate RuleML into Prolog codes.

To use the COIN-OWL Plugin, the application/ontology administrator will use Protégé OWL Plugin to create the application ontology. This is done using the Individuals tab as shown on Figure 4-2.


tasc Protégé 2.1 beta (file:\C:\Program%20Files\Protege_2.1_beta\projects\tasc\tasc.pprj, OWL Files)

Classes:
owl:Thing
coin:Attribute (8)
coin:Column (53)
coin:Context (5)
coin:JoinValue
coin:JoinValueColumn (2
coin:JoinValueLiteral
coin:Modifier (5)
coin:ModifierValue
coin:ModifierDynamicValu
coin:ModifierStaticValue (
coin:Relation (15)
coin:SemanticAttribute (2)
coin:ComplexSemanticAtt
coin:SemanticColumn (16)
coin:SemanticRelation (3)
coin:SemanticType (7)

Display Slot: :NAME

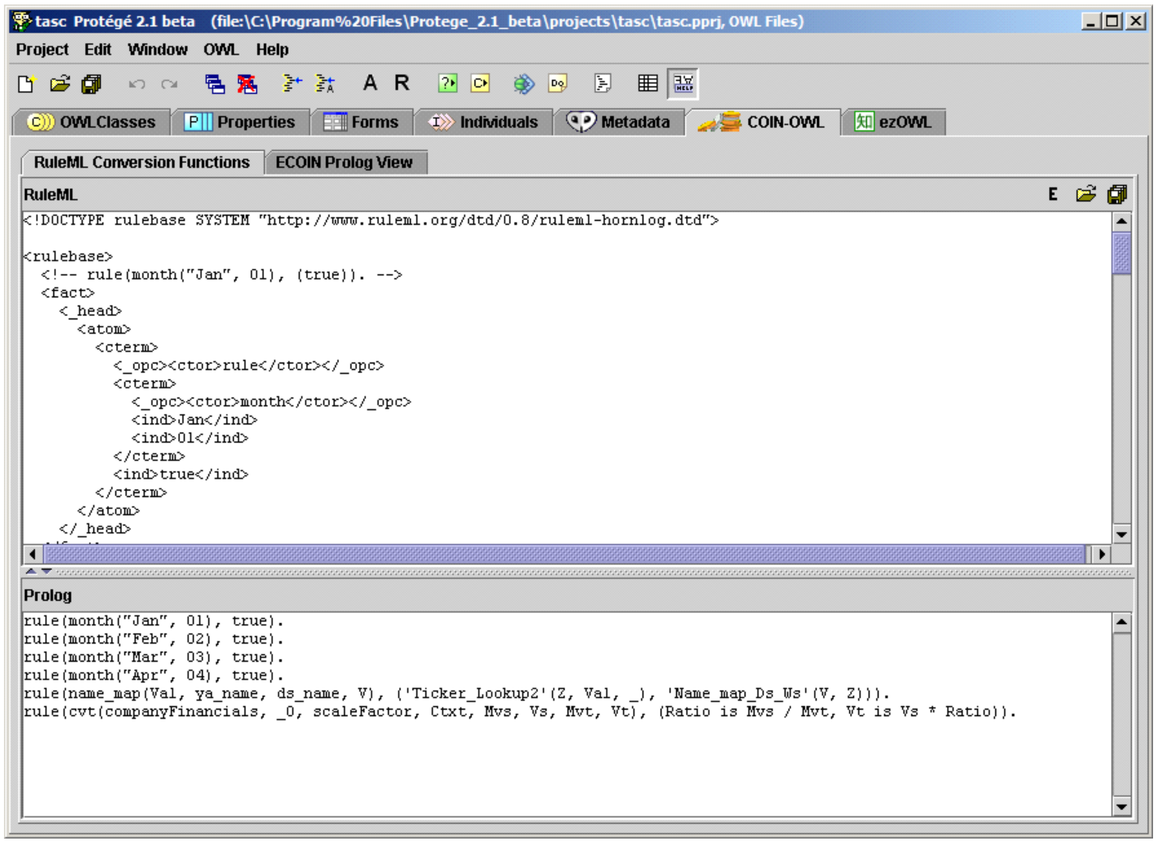
Direct Instances:
col_CNAmes_COMPANY_NAME
col_countryIncorp_COMPANYNAME
col_countryIncorp_COUNTRY
col_Currencymap_CHAR2_CURREN...
col_Currencymap_CHAR3_CURREN...
col_Currencytypes_COUNTRY
col_Currencytypes_CURRENCY
col_datexform_Date1
col_datexform_Date2
col_datexform_Format1
col_datexform_Format2
col_discAF_COMPANY_NAME
col_discAF_CURRENT_SHARES_OU...
col_discAF_LATEST_ANNUAL_DATA
col_discAF_LOCATION_OF_INCORP
col_discAF_NET_INCOME
col_discAF_NET_SALES
col_discAF_TOTAL_ASSETS
col_DStreamAF_AS_OF_DATE
col_DStreamAF_CURRENCY
col_DStreamAF_EARNED_FOR_ORD...
col_DStreamAF_NAME
col_DStreamAF_TOTAL_EXTRAORD...
col_DStreamAF_TOTAL_SALES
col_NamemapDsWs_DS_NAMES
col_NamemapDsWs_WS_NAMES

col_CNAmes_COMPANY_NAME (type=coin:Column)

Name: col_CNAmes_COMPANY_NAME
rdfs:comment:
Coin:ColumnName: COMPANY_NAME
Coin:AssociatedRela: rel_CNames
Coin:DataType: string
Coin:ColumnIndex: 1

Figure 4-2 Creating OWL class instances using the Individuals tab

As Protégé does not currently support the editing of RuleML rules, we have included a simple editor interface for the creation of RuleML conversion rules for the COIN ontology (see Figure 4-3). This feature is accessible from the COIN-OWL tab. The ontology administrator can enter the RuleML conversion functions on the RuleML editor area under the RuleML Conversion Functions tab. Automated conversion function of RuleML rules to Prolog representation can be performed by pressing the “Transform” button.

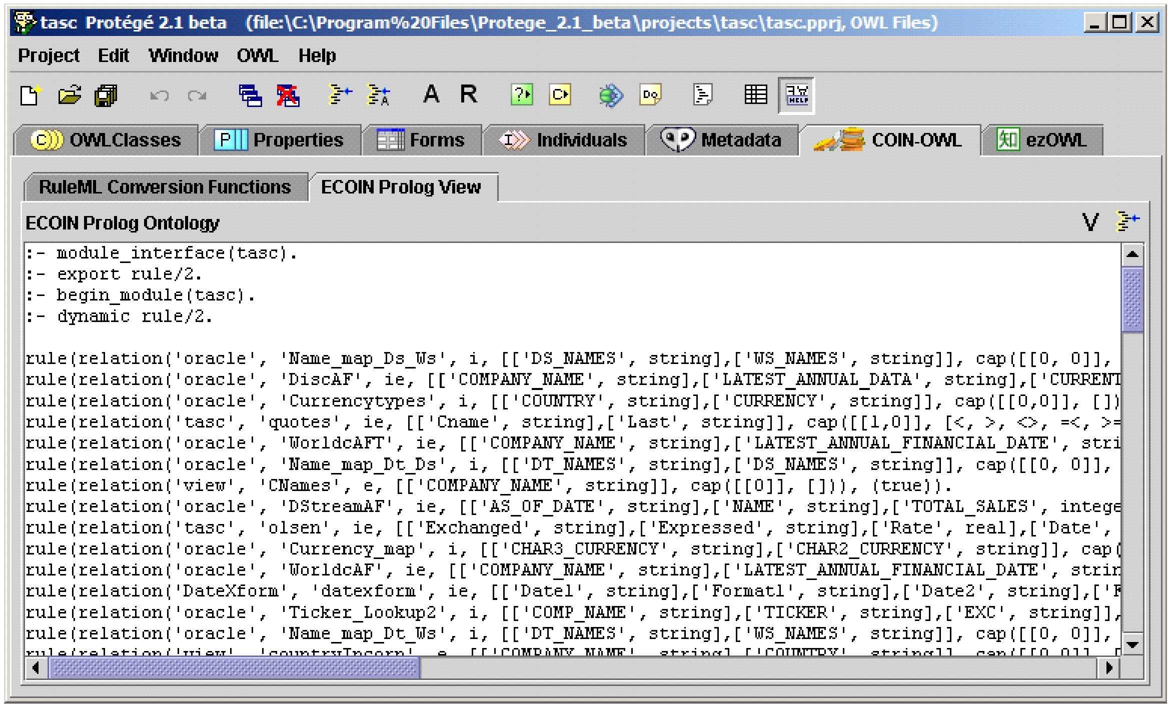

tasc Protégé 2.1 beta (file:\C:\Program%20Files\Protege_2.1_beta\projects\tasc\tasc.pprj, OWL Files)
Project Edit Window OWL Help
OWLClasses Properties Forms Individuals Metadata COIN-OWL ezOWL
RuleML Conversion Functions ECOIN Prolog View
RuleML
Prolog

```
<!DOCTYPE rulebase SYSTEM "http://www.ruleml.org/dtd/0.8/ruleml-hornlog.dtd">

<rulebase>
  <!-- rule(month("Jan", 01), (true)). -->
  <fact>
    <_head>
      <atom>
        <cterm>
          <_opc><ctor>rule</ctor></_opc>
          <cterm>
            <_opc><ctor>month</ctor></_opc>
            <ind>Jan</ind>
            <ind>01</ind>
          </cterm>
          <ind>true</ind>
        </cterm>
      </atom>
    </_head>
```

```
rule(month("Jan", 01), true).
rule(month("Feb", 02), true).
rule(month("Mar", 03), true).
rule(month("Apr", 04), true).
rule(name_map(Val, ya_name, ds_name, V), ('Ticker_Lookup2'(Z, Val, _), 'Name_map_Ds_Ws'(V, Z))).
rule(cvt(companyFinancials, _0, scaleFactor, Ctxt, Mvs, Vs, Mvt, Vt), (Ratio is Mvs / Mvt, Vt is Vs * Ratio)).
```

Figure 4-3: RuleML tab on the COIN-OWL Protégé Plugin


tasc Protégé 2.1 beta (file:\C:\Program%20Files\Protege_2.1_beta\projects\tasc\tasc.pprj, OWL Files)
Project Edit Window OWL Help
OWLClasses
Properties
Forms
Individuals
Metadata
COIN-OWL
ezOWL
RuleML Conversion Functions
ECOIN Prolog View
ECOIN Prolog Ontology

```
:- module_interface(tasc).
:- export rule/2.
:- begin_module(tasc).
:- dynamic rule/2.

rule(relation('oracle', 'Name_map_Ds_Ws', i, [['DS_NAMES', string],['WS_NAMES', string]], cap([[0, 0]],
rule(relation('oracle', 'DiscAF', ie, [['COMPANY_NAME', string],['LATEST_ANNUAL_DATA', string],['CURRENT
rule(relation('oracle', 'Currencytypes', i, [['COUNTRY', string],['CURRENCY', string]], cap([[0,0]], [])
rule(relation('tasc', 'quotes', ie, [['Cname', string],['Last', string]], cap([[1,0]], [<, >, <>, =<, >=
rule(relation('oracle', 'WorldcAFT', ie, [['COMPANY_NAME', string],['LATEST_ANNUAL_FINANCIAL_DATE', stri
rule(relation('oracle', 'Name_map_Dt_Ds', i, [['DT_NAMES', string],['DS_NAMES', string]], cap([[0, 0]],
rule(relation('view', 'CNames', e, [['COMPANY_NAME', string]], cap([[0]], [])), (true)).
rule(relation('oracle', 'DStreamAF', ie, [['AS_OF_DATE', string],['NAME', string],['TOTAL_SALES', intege
rule(relation('tasc', 'olsen', ie, [['Exchanged', string],['Expressed', string],['Rate', real],['Date',
rule(relation('oracle', 'Currency_map', i, [['CHAR3_CURRENCY', string],['CHAR2_CURRENCY', string]], cap(
rule(relation('oracle', 'WorldcAF', ie, [['COMPANY_NAME', string],['LATEST_ANNUAL_FINANCIAL_DATE', strin
rule(relation('DateXform', 'datexform', ie, [['Date1', string],['Format1', string],['Date2', string],['F
rule(relation('oracle', 'Ticker_Lookup2', i, [['COMP_NAME', string],['TICKER', string],['EXC', string]],
rule(relation('oracle', 'Name_map_Dt_Ws', i, [['DT_NAMES', string],['WS_NAMES', string]], cap([[0, 0]],
rule(relation('view', 'countryIncorp', e, [['COMPANY_NAME', string],['COUNTRY', string]], cap([[0,0]], [
```

Figure 4-4 Conversion of OWL ontology and RuleML into eCOIN Prolog ontology

The final eCOIN Prolog ontology can be generated from the COIN-OWL ontology and the RuleML conversion rules from the eCOIN Prolog View tab. This complete Prolog ontology can then be fetched into the eCOIN system for context mediation without the need for any change on the existing system.

CHAPTER 5
Conclusion

In summary, we have presented an ontology model in OWL for the Context Interchange strategy through this project in our effort of adopting the recommended technology by W3C. The COIN-OWL ontology model design is built on the building blocks of the OWL Lite sublanguage family and the Rule Markup Language, which are used to model the core ontology and the rule-based metadata in COIN respectively.

Follow up on that, we put forth the implementation strategy on the use of the COIN-OWL model, and wrapped up our work with a fully working prototype for the COIN-OWL model. The prototype is built as an plugin extension to the Protégé OWL ontology editor, with support to RuleML editing and automated conversion from COIN-OWL ontology to eCOIN Prolog ontology for immediate use in the eCOIN implementation prototype.

With the growing adoption of OWL and the gradual realization of the Semantic Web vision, this work is instrumental in bridging the gap between COIN and Semantic Web. With the COIN-OWL model, it is hopeful that COIN will be able to reach a larger spectrum of audiences, and hence bringing even more contribution to the database/Semantic Web community in the area of heterogeneous data interoperability.

5.1 Future Work

The completion of this project also opens up several other research issues, which we hope to explore in the future. In this section, we highlight some of the interesting and promising research areas.

We noted that in parallel with the development of RuleML, a number of relevant emerging standards have been branched from RuleML, including RuleML Lite and Semantic Web Rule Language (SWRL). As these standards mature, in particular SWRL, which combines OWL and RuleML, such standards promise a more cohesive rule-based ontology model. A simple example illustrating this is given in Section 3.5.7. One reservation on SWRL, however, is that it is based on the RuleML datalog sublanguage, where as the minimum requirement for our current implementation requires the hornlog sublanguage family for total compatibility with Prolog.

Another interesting research area is the use of COIN in ontology interoperability and sharing. With growing adoption of OWL, we can expect disparate ontologies being developed. It is possible that a good number of these ontology will have overlapping domain definition, and the classic problem of data integration will be re-surfaced on the arena of OWL ontology in Semantic Web. As such, we envisage that COIN strategy can be leveraged at the meta-ontology level to solve the ontology interoperability problem.

REFERENCES

[1] "The DARPA Agent Markup Language." http://www.daml.org.

[2] "eCOIN Demo for TASC Financial Example." http://interchange.mit.edu:8080/gcms_v4/Demo.jsp?app_id=2&qindex=0.

[3] "Electronic Business using eXtensible Markup Language (ebXML)." http://www.ebxml.org.

[4] "Pellet OWL Reasoner." http://www.mindswap.org/2003/pellet/index.shtml.

[5] "pOWL - Semantic Web Development Plattform." http://powl.sourceforge.net/.

[6] "Protégé OWL Plugin - Ontology Editor for the Semantic Web."

[7] "OWL Implementations," 2003. http://www.w3.org/2001/sw/WebOnt/impls.

[8] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*, vol. 5, 2001, pp. 34-43.

[9] H. Boley, "The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations," In Proceedings of the 14th International Conference of Applications of Prolog, 2001.

[10] P. Bouquet, F. Giunchiglia, F. v. Harmelen, L. Serafini, and H. Stuckenschmidt, "C-OWL: Contextualizing Ontologies," In Proceedings of the Second International Semantic Web Conference, 2003.

[11] S. Bressan, K. Fynn, C. H. Goh, S. E. Madnick, T. Pena, and M. D. Siegel, "Overview of a Prolog Implementation of the COntext INterchange Mediator," In Proceedings of the 5th International Conference and Exhibition on The Practical Applications of Prolog., 1997.

[12] S. Bressan, C. H. Goh, T. Lee, S. E. Madnick, and M. Siegel, "A Procedure for Mediation of Queries to Sources in Disparate Contexts," In Proceedings of the International Logic Programming Symposium, Port Jefferson, N.Y., 1997.

[13] D. Brickley and R. V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation," 1999. http://www.w3.org/TR/rdf-schema/.

[14] M. Danny, "Ontology Building: A Survey of Editing Tools," 2002. http://www.xml.com/pub/a/2002/11/06/ontologies.html.

[15] A. Firat, "Information Integration Using Contextual Knowledge and Ontology Merging," Ph.D. Thesis, Massachusetts Institute of Technology, Sloan School of Management, 2003.

[16] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel, "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information," *ACM Transactions on Information Systems*, vol. 17, pp. 270-293, 1999.

[17] M. Hori, J. Euzenat, and P. F. Patel-Schneider, "OWL Web Ontology Language XML Presentation Syntax," *W3C Note 11 June 2003*, 2003. http://www.w3.org/TR/owl-xmlsyntax/.

[18] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," 2004. http://www.daml.org/2004/04/swrl/.

[19] H. Kim, "Predicting How Ontologies for the semantic Web Will Evolve," *Communications of the ACM*, vol. 45, pp. 48-54, 2002.

[20] P. W. Lee, "Metadata Representation and Management for Context Mediation," Master Thesis, Massachusetts Institute of Technology, Sloan School of Management, 2003.

[21] D. L. McGuinness and F. v. Harmelen, "OWL Web Ontology Language Overview," *W3C Proposed Recommendation 15 December 2003*, 2003. http://www.w3.org/TR/2003/PR-owl-features-20031215/.

[22] S. Michael and S. E. Madnick, "A Metadata Approach to Resolving Semantic Conflicts," In Proceedings of the 17th Conference on Very Large Data Bases, 1991.

[23] M. Siegel and S. E. Madnick, "A Metadata Approach to Resolving Semantic Conflicts," In Proceedings of the 17th Conference on Very Large Data Bases, 1991.

[24] M. K. Smith, C. Welty, and D. L. McGuinness, "OWL Web Ontology Language Guide," 2003. http://www.w3.org/TR/2003/PR-owl-guide-20031215.

APPENDIX A

COIN-OWL Ontology

```
<?xml version="1.0"?>

<!DOCTYPE rdf:RDF [
     <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
   ]>

<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns=""
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="">
  <owl:Ontology rdf:about="">
    <rdfs:comment>COIN Ontology</rdfs:comment>
  </owl:Ontology>

  <!--
    Context Axioms, consists of:
    1. Context class that models a context,
    2. ModifierValue, ModifierStaticValue and ModifierDynamicValue that represent
valueObject for the modifiers
  -->

  <owl:Class rdf:ID="Context"/>
  <owl:DatatypeProperty rdf:ID="ContextName">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Context"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>

  <owl:Class rdf:ID="Attribute" />
  <owl:DatatypeProperty rdf:ID="AttributeName">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="&xsd;string"/>
  </owl:DatatypeProperty>
  <owl:ObjectProperty rdf:ID="AttributeFrom">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="#SemanticType"/>
    <owl:inverseOf rdf:resource="#Attributes" />
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="AttributeTo">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#Attribute"/>
    <rdfs:range rdf:resource="#SemanticType"/>
```

```
</owl:ObjectProperty>

<owl:Class rdf:ID="Modifier"/>
<owl:DatatypeProperty rdf:ID="ModifierName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Modifier"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="ModifierValues">
  <rdfs:range rdf:resource="#ModifierValue"/>
  <rdfs:domain rdf:resource="#Modifier"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ModifierFrom">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Modifier"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ModifierTo">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Modifier"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SemanticType" />

<owl:ObjectProperty rdf:ID="Modifiers">
  <rdfs:domain rdf:resource="#SemanticType"/>
  <rdfs:range rdf:resource="#Modifier"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="Attributes">
  <rdfs:domain rdf:resource="#SemanticType"/>
  <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="Relation" />
<owl:DatatypeProperty rdf:ID="DataSource">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="RelationName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Type">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Bindings">
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="UnsupportedOperators">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Relation"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="Columns">
  <rdfs:range rdf:resource="#Column"/>
  <rdfs:domain rdf:resource="#Relation"/>
</owl:ObjectProperty>


<owl:Class rdf:ID="Column" />
<owl:DatatypeProperty rdf:ID="ColumnName">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Column"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="DataType">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Column"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ColumnIndex">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Column"/>
  <rdfs:range rdf:resource="&xsd;int"/>
</owl:DatatypeProperty>
<owl:ObjectProperty rdf:ID="AssociatedRelation">
  <rdfs:domain rdf:resource="#Column"/>
  <owl:inverseOf rdf:resource="#Columns" />
</owl:ObjectProperty>


<owl:Class rdf:ID="ModifierValue" />
<owl:Class rdf:ID="ModifierStaticValue">
  <rdfs:subClassOf rdf:resource="#ModifierValue" />
</owl:Class>
<owl:Class rdf:ID="ModifierDynamicValue">
  <rdfs:subClassOf rdf:resource="#ModifierValue" />
</owl:Class>

<!-- common properties at parent class: ModifierValue-->
<owl:ObjectProperty rdf:ID="ModifierSemanticType">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#ModifierValue"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ModifierContext">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#ModifierValue"/>
  <rdfs:range rdf:resource="#Context"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="ModifierObject">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#ModifierValue"/>
  <rdfs:range rdf:resource="#Modifier"/>
  <owl:inverseOf rdf:resource="#ModifierValues" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="ModifierStringValue">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#ModifierStaticValue"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="ModifierNumericValue">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#ModifierStaticValue"/>
  <rdfs:range rdf:resource="&xsd;float"/>
</owl:DatatypeProperty>

<owl:ObjectProperty rdf:ID="ModifierDynamicValues">
  <rdfs:domain rdf:resource="#ModifierDynamicValue"/>
  <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>

<!--
Elevation Theory
Formal mappings between the sources and the shared ontology
-->

<owl:Class rdf:ID="SemanticRelation"/>
<owl:ObjectProperty rdf:ID="ElevationContext">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticRelation"/>
  <rdfs:range rdf:resource="#Context"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="SourceRelation">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticRelation"/>
  <rdfs:range rdf:resource="#Relation"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="SemanticColumns">
  <rdfs:domain rdf:resource="#SemanticRelation"/>
  <rdfs:range rdf:resource="#SemanticColumn"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="SemanticColumn" />
<owl:ObjectProperty rdf:ID="SourceColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticColumn"/>
  <rdfs:range rdf:resource="#Column"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="TargetSemanticType">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticColumn"/>
  <rdfs:range rdf:resource="#SemanticType"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="AssociatedSemanticRelation">
```

```
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticColumn"/>
  <rdfs:range rdf:resource="#SemanticRelation"/>
  <owl:inverseOf rdf:resource="#SemanticColumns" />
</owl:ObjectProperty>

<!-- Join Value -->
<owl:Class rdf:ID="JoinValue" />
<owl:Class rdf:ID="JoinValueColumn">
  <rdfs:subClassOf rdf:resource="#JoinValue" />
</owl:Class>
<owl:Class rdf:ID="JoinValueLiteral">
  <rdfs:subClassOf rdf:resource="#JoinValue" />
</owl:Class>

<owl:ObjectProperty rdf:ID="JoinSourceSemanticColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#JoinValue"/>
  <rdfs:range rdf:resource="#SemanticColumn"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="JoinValueContext">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#JoinValue"/>
  <rdfs:range rdf:resource="#Context"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="JoinTargetSemanticColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#JoinValueColumn"/>
  <rdfs:range rdf:resource="#SemanticColumn"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="JoinTargetString">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#JoinValueLiteral"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<!-- Semantic Attribute -->
<owl:Class rdf:ID="SemanticAttribute" />
<owl:Class rdf:ID="ComplexSemanticAttribute">
  <rdfs:subClassOf rdf:resource="#SemanticAttribute" />
</owl:Class>

<owl:ObjectProperty rdf:ID="SourceAttribute">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticAttribute"/>
  <rdfs:range rdf:resource="#Attribute"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="SourceSemanticColumn">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#SemanticAttribute"/>
  <rdfs:range rdf:resource="#SemanticColumn"/>
</owl:ObjectProperty>
```

```
  <owl:ObjectProperty rdf:ID="TargetSemanticColumn">
    <rdf:type rdf:resource="&owl;FunctionalProperty" />
    <rdfs:domain rdf:resource="#SemanticAttribute"/>
    <rdfs:range rdf:resource="#SemanticColumn"/>
  </owl:ObjectProperty>

  <owl:ObjectProperty rdf:ID="JoinValues">
    <rdfs:domain rdf:resource="#ComplexSemanticAttribute"/>
    <rdfs:range rdf:resource="#JoinValue"/>
  </owl:ObjectProperty>

</rdf:RDF>
```

APPENDIX B

Extensible Stylesheet (XSL) for RuleML to Prolog Transformation

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" exclude-result-prefixes="rdf
coin" version="1.0">

<xsl:output method="text" indent="no" media-type="string"/>

<xsl:variable name="LOWERCASE" select="'abcdefghijklmnopqrstuvwxyz0123456789'" />
<xsl:variable name="UPPERCASE" select="'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'" />

<xsl:template match="rulebase">
  <xsl:apply-templates select="fact" />
  <xsl:apply-templates select="imp" />
  <xsl:text>&#10;</xsl:text>
</xsl:template>

<xsl:template match="imp">
  <xsl:apply-templates select="_head" />.
</xsl:template>

<xsl:template match="fact">
  <xsl:apply-templates select="_head" />.
</xsl:template>

<xsl:template match="_head">
  <xsl:apply-templates select="atom" />
</xsl:template>

<xsl:template match="atom">
  <xsl:apply-templates select="cterm" />
</xsl:template>

<xsl:template match="cterm">
    <xsl:for-each select="child::*">
      <xsl:variable name="curEl" select="name()"/>
      <xsl:apply-templates select="." />
      <!--<xsl:value-of select="$curEl" />-->
      <xsl:if test="$curEl != '_opc'">
        <xsl:if test="position() &lt; last()-1">, </xsl:if><xsl:if
test="position()=last()-1">, </xsl:if>
      </xsl:if>
    </xsl:for-each>)</xsl:template>

<xsl:template match="_opc">
  <xsl:variable name="data" select="ctor" />
  <xsl:choose>
```

```
    <xsl:when test="string-length($data)=0">(</xsl:when>
    <xsl:when test="starts-with($data, translate(substring(.,1,1), $LOWERCASE,
$UPPERCASE))">'<xsl:value-of select="$data"/>'(</xsl:when>
    <xsl:otherwise><xsl:value-of select="$data"/>(</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="var"><xsl:value-of select="." /></xsl:template>

<xsl:template match="ind">
  <!-- if starts with upper case, we need to enclose within quotes -->
  <xsl:variable name="data" select="." />
  <xsl:choose>
    <xsl:when test="starts-with($data, translate(substring(.,1,1), $LOWERCASE,
$UPPERCASE))">"<xsl:value-of select="$data"/>"</xsl:when>
    <xsl:otherwise><xsl:value-of select="$data"/></xsl:otherwise>
  </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

APPENDIX C

RuleML XML Schema (DTD)

Horn-Logic RuleML Sublanguage

```
<!-- An XML DTD for a Datalog RuleML Sublanguage -->
<!-- Last Modification: 2002-04-01 -->
<!-- ENTITY Declarations -->
<!-- in this ruleml-datalog.dtd, parameter entities set two *.module switches to
INCLUDE -->

<!ENTITY % datalog.module "INCLUDE">
<!ENTITY % datalog-and-hornlog.module "INCLUDE">


<!-- hence all conditional sections "<![%*.module;[" . . . "]]>" activate their content;
-->
<!-- in a stand-alone use of the current DTD "<![%*.module;[" and "]]>" are thus no-ops
-->

<!-- ENTITY and ATTLIST Declarations only for upward compatibility with XML Schema -->
<!-- URI-valued (CDATA) attributes optionally specify an XML Schema on 'rulebase' root
-->

<!ENTITY % URI "CDATA">
<!ATTLIST rulebase xsi:noNamespaceSchemaLocation %URI; #IMPLIED>
<!ATTLIST rulebase xmlns:xsi %URI; #IMPLIED>

<!-- ELEMENT and ATTLIST Declarations -->
<!-- 'rulebase' root element uses 'imp' rules and 'fact' assertions along with 'query'
tests as top-level elements -->
<!-- It has an optional label.  -->

<!-- direction attribute indicates the intended direction of imp inferencing;  -->
<!-- it is a preliminary design choice and has a 'neutral' default value -->

<!ELEMENT rulebase (( (_rbaselab, (imp | fact | query)* ) | ((imp | fact | query)+,
_rbaselab?) )?) >
<!ATTLIST rulebase direction (forward | backward | bidirectional) "bidirectional">

<!-- 'imp' is short for "implication"; it is a kind of rule -->
<!-- 'imp' rules are usable on the rulebase top-level -->
<!-- 'imp' element uses a conclusion role _head followed by a premise role _body, or
equivalently -->
<!-- (since roles constitute unordered elements), uses a premise role _body followed by
a conclusion role _head -->
<!-- "<imp>_head _body</imp>" stands for "_head is implied by _body", i.e., "_head is
true is implied by _body is true", or equivalently, -->
<!-- "<imp>_body _head</imp>" stands for "_body implies _head", i.e., "_body is true
implies _head is true" -->
```

```
<!-- rule label is a handle for the imp: for various uses, including prioritization -->

<!ELEMENT imp ((_rlab, ((_head, _body) | (_body, _head))) |
               (_head, ((_rlab, _body) | (_body, _rlab?))) |
               (_body, ((_rlab, _head) | (_head, _rlab?)))) >

<!-- 'fact' assertions are usable as degenerate rules on the rulebase top-level -->
<!-- 'fact' element uses just a conclusion role _head -->
<!-- "<fact>_head</fact>" stands for "_head is implied by true", i.e., "_head is true"
-->

<!-- "_rlab" is a handle for the fact: for various uses, including editing -->

<!-- NOTE:  for now, fact is not required to be ground -->
<!-- FUTURE DESIGN:  perhaps require fact to be ground;
     note that any requirement of groundedness of fact's must be enforced beyond the
DTD validation -->
<!ELEMENT fact ((_rlab, _head) | (_head, _rlab?)) >

<!-- 'query' elements are usable as degenerate rules on the rulebase top-level -->
<!-- 'query' element uses just a premise role _body -->
<!-- "<query>_body</query>" stands for "false is implied by _body", i.e., "_body cannot
be proved", which is to be refuted by generating the bindings for free variables in
_body -->

<!-- "_rlab" is a handle for the query: for various uses, including editing -->

<!ELEMENT query ((_rlab, _body) | (_body, _rlab?)) >
<![%datalog-and-hornlog.module;[

<!-- _head role is usable within 'imp' rules and 'fact' assertions -->
<!-- _body role is usable within 'imp' rules and 'query' tests -->
<!-- _head uses an atomic formula -->
<!-- _body uses an atomic formula or an 'and' -->
<!ELEMENT _head (atom)>
<!ELEMENT _body (atom | and)>

<!-- an 'and' is usable within _body's -->
<!-- 'and' uses zero or more atomic formulas -->
<!-- "<and>atom</and>" is equivalent to "atom"-->
<!-- "<and></and>" is equivalent to "true"-->
<!ELEMENT and (atom*)>
]]>

<![%datalog.module;[
<!-- "_rbaselab" is is short for "rulebase label"; must be ind(ividual);
     this allows naming of an entire individual rulebase in a fashion that is
accessible
     within the knowledge representation; -->
<!-- e.g., this can help for representing prioritization between rulebases, or perhaps
     to enable forward inferencing of selected rulebase(s)  -->

<!ELEMENT _rbaselab (ind)>
<!-- "_rlab" is short for "rule label"; must be ind(ividual);
```

```
     this allows naming of a rule (either imp or fact) in a fashion that is accessible
     within the knowledge representation; -->
<!-- e.g., this can help for representing prioritization between rules -->
<!-- NOTE:  rule labels are not required to be unique within a rulebase -->
<!ELEMENT _rlab (ind)>

<!-- atomic formulas are usable within _head's, _body's, and 'and's -->
<!-- atom element uses an: -->
<!-- _opr ("operator of relations") role followed by a sequence of zero or more
arguments, or similarly -->
<!-- (since roles constitute unordered elements, and the zero-argument case must not
cause ambiguity), -->
<!-- a sequence of one or more arguments followed by an _opr role -->
<!-- the arguments may be ind(ividual)s or var(iable)s -->

<!ELEMENT atom ((_opr, (ind | var)*) | ((ind | var)+, _opr))>
]]>

<!-- _opr is usable within atoms -->
<!-- _opr uses rel(ation) symbol -->
<!ELEMENT _opr (rel)>

<!-- there is one kind of fixed argument -->
<!-- individual constant, as in predicate logic -->
<!ELEMENT ind  (#PCDATA)>

<!-- there is one kind of variable argument -->
<!-- logical variable, as in logic programming -->
<!ELEMENT var  (#PCDATA)>


<!-- there are only fixed (first-order) relations -->
<!-- relation or predicate symbol -->
<!ELEMENT rel  (#PCDATA)>
```

Horn-Logic RuleML Sublanguage

```
<!-- An XML DTD for a Horn-Logic RuleML Sublanguage -->
<!-- Last Modification: 2001-07-10 -->

<!-- ENTITY Declarations -->
<!ENTITY % datalog-and-hornlog.module "INCLUDE">
<!ENTITY % datalog.module "IGNORE">
<!ENTITY % datalog SYSTEM "ruleml-datalog.dtd">
%datalog;

<!-- ELEMENT Declarations -->
<!-- complex, compound, or constructor terms are usable within other cterms, tups,
rolis, and atoms -->
<!-- cterm element uses _opc ("operator of constructors") role followed by sequence of
five kinds of arguments, -->
<!-- or vice versa, much like atoms (explained below) -->
```

```
<!ELEMENT cterm ((_opc, (ind | var | cterm | tup | roli)*) | ((ind | var | cterm | tup
| roli)+, _opc))>

<!-- _opc is usable within c(onstructor )terms -->
<!-- _opc uses c(onstruc)tor symbol -->
<!ELEMENT _opc (ctor)>

<!-- constructors -->
<!ELEMENT ctor (#PCDATA)>


<!-- NOTICE: tups and rolis are still very preliminary -->
<!-- n-tuples are usable within other tups, rolis, cterms, and atoms -->
<!-- tup element uses sequence of five kinds of arguments -->
<!ELEMENT tup   ((ind | var | cterm | tup | roli)*)>

<!-- "roli" is short for "role list" -->
<!-- sequence is not (syntactically) significant among its children, i.e., it is
sequence-free -->

<!ELEMENT roli ((_arv)*)>
<!ELEMENT _arv ((arole, (ind | var | cterm | tup | roli)) | ((ind | var | cterm | tup |
roli), arole)) >
<!ELEMENT arole (#PCDATA)>


<!-- "_rbaselab" is is short for "rulebase label"; may be ind(ividual) or
c(onstructor )term;
     this allows naming of an entire individual rulebase in a fashion that is
accessible
     within the knowledge representation; -->
<!-- e.g., this can help for representing prioritization between rulebases, or perhaps
     to enable forward inferencing of selected rulebase(s)  -->
<!-- SYNTACTIC REQUIREMENT BEYOND DTD:  for now, must be GROUND (e.g., if cterm) -->
<!-- FUTURE DESIGN:  might permit to be non-ground;
     e.g., to instantiate a personal messaging agent to the particular user;
     but that would require that coincidence of variable names be significant ACROSS
rules,
     and there are expressively simpler ways to achieve the same effect -->
<!ELEMENT _rbaselab (ind | cterm)>

<!-- "_rlab" is short for "rule label"; may be ind(ividual) or c(onstructor )term;
     this allows naming of a rule (either imp or fact) in a fashion that is accessible
     within the knowledge representation; -->
<!-- e.g., this can help for representing prioritization between rules -->
<!-- NOTE:  rule labels are not required to be unique within a rulebase -->
<!-- SYNTACTIC REQUIREMENT BEYOND DTD:  any logical variables (var elements)
appearing within the rule label (i.e., within _rlab's cterm child) must also appear
within
the rule body and/or head -->
<!-- FUTURE DESIGN:  probably will want to permit even stronger restrictions on the
appearance of variables, e.g.,
"must appear within both the rule head and body" or
"must appear within the rule body but not in any literal which has negation-as-failure,
nor in any literal which can sensed" in OLP or CLP or SLP or SCLP -->

<!-- NOTE:  rule label is not required to be ground; semantically, instantiating
the rule label's logical variables corresponds to instantiating the (rest of the)
rule's variables.  For example, if the rule says "Mortal(?x) if Man(?x)", and
the rule label is "SocraticSyllogism(?x)", then "SocraticSyllogism(Joe)"
corresponds semantically to the rule label for "Mortal(Joe) if Man(Joe)".
 -->

<!ELEMENT _rlab (ind | cterm) >
<!-- atomic formulas are usable within _head's, _body's, and 'and's -->
<!-- atom element uses an: -->
<!-- _opr ("operator of relations") role followed by a sequence of zero or more
arguments, or similarly -->
<!-- (since roles constitute unordered elements, and the zero-argument case must not
cause ambiguity), -->
<!-- a sequence of one or more arguments followed by an _opr role -->
<!-- the arguments may be ind(ividual)s, var(iable)s, c(onstructor )terms, (n-)tup(le)s,
or ro(le )li(st)s -->

<!ELEMENT atom ((_opr, (ind | var | cterm | tup | roli)*) | ((ind | var | cterm | tup |
roli)+, _opr))>
```