Metadata Representation and Management for Context Mediation

by

Philip W. Lee

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirement for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 21, 2003
(June 2003)

Author _____
Department of Electrical Engineering and Computer Science
May 21, 2003

Certified By _____
Stuart E. Madnick
John Norris Maguire Professor of Information Technologies
Thesis Supervisor

Accepted By _____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Metadata Representation and Management for Context Mediation
by
Philip W. Lee

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 2003

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

The Context Interchange (COIN) framework is a platform of concepts that bridges the contextual divide among heterogeneous data sources. In this thesis, we present a metadata representation and management layer that augments the Context Interchange framework. On metadata representation, a suite of XML-based representation of COIN metadata is fashioned. These XML-based representations include RDF, RuleML, and RFML. The transformation between the different representations is done through XSLT. An intuitive user interface is also created to aid the creation of metadata for context mediation. The user interface features textual and graphical components that conceptualize ontology, context, and source information. On metadata management, a registry is set up to provide centralized access of metadata files. The designed system provides facility for knowledge sharing and reuse, a concept that is missing in the previously Prolog dominated COIN system.

Thesis Supervisor: Stuart E. Madnick
Title: John Norris Maguire Professor of Information Technologies,
Sloan School of Management

Acknowledgement

I would like to take this opportunity to thank the individuals of Context Interchange team whose valuable help led to the successful completion of my thesis. My thesis advisor, Prof. Stuart Madnick, has generously contributed his time and incredible talent in guiding me every step of the way. His enthusiastic and industrious drive for higher standards motivates me to take on harder challenges academically and personally. I would also like to thank Dr. Michael Siegel for his guidance and patience in keeping me at pace in my progress. Special thanks go to Frank Manola of MITRE, whose bottomless insight of the Semantic Web has inspired the design of this thesis. Last but not least, I would like to express gratitude to Aykut Firat for contributing his technical expertise.

I have also received tremendous support and blessing from within my family. Mom, thank you. Your selfless sacrifice is catalyst to my success. The timeless wisdom that personal excellence stems from academic excellence, which you taught me early on in my life, has benefited me wonderfully. To my grandmother, whose story of teaching herself how to read and write in war-torn China has encouraged me to never give up in hard times, I owe my special gratitude. Finally, I would like to thank my brother, whose exemplary diligence and curiosity in knowledge has fueled my academic journey since childhood.

Table of Contents

Table of Figures

1 Introduction

There have been many barriers which prevented meaningful exchange of information since humans started to use symbols and sounds to represent things in this world. In the early ages when modern transportation was lacking, human interaction was limited by geographical barriers. Fortunately, geographical barriers are virtually non-existent in today's electronic age. Email and the World Wide Web enable people to communicate at the speed of light. Information can travel on these electronic vehicles of communication even though the sources of information are physically apart. A second communication barrier is the language disparity among people. The very symbols and sounds that enable one group of people to communicate are barring another group from joining the conversation. Therefore, having a common language is a prerequisite to meaningful exchange of information. The language problem among humans is less severe than among information systems because the number of languages in this world has stabilized and there are only a limited set of spoken languages. The proliferation of standards among information systems has improved but confused the communication channels. Huge expenses are often incurred in reconciliation of information from companies wishing to share their business logic and knowledge. A third communication barrier is contextual difference. Even after two parties agree on a chosen language to communicate, the semantics, or meanings in the language, of things are still context dependent. A concept that exists in one place may not be so common in another. For example, a military personnel may refer to six o'clock in the evening as 18:00, while a non-military person may interpret it as 6 PM. Given the barriers of communication mentioned, it is no surprise that meaningful exchange of information rarely occurs naturally. Fortunately, a research effort to effectively resolve those communication issues is already underway at the Context Interchange (COIN) Group in the Sloan School of Management at MIT.

The framework developed here at COIN for targeting contextual communication issues is a modeling technique. The COIN Model encapsulates and communicates the technique through its constituent elements, which are ontological, relational, and contextual in nature. A COIN application is an encapsulation of a domain of interest for the purpose of context mediation. Every COIN application is an instance of the COIN model. The extended-COIN, or eCOIN, is an implementation of the COIN framework. eCOIN takes as inputs a COIN application and performs context mediation on user submitted queries. eCOIN consists of two major components, the Planner/Optimizer/Executioner (POE), and the abduction engine. The abduction engine is written in Prolog, a logic programming language, and provides the logic behind context mediation work. The POE utilizes the abduction engine to convert user submitted queries into context mediated queries, and connects to data sources in an optimized fashion for the purpose of executing the mediated queries. The information presented in a COIN application is used directly by the abduction engine, and therefore must be written in Prolog language. While Prolog is a common language among practitioners of logic programming, it is not easily understandable by someone with business background, who simply wants to model their domain for context mediation. The work of this thesis targets precisely at this limitation of eCOIN.

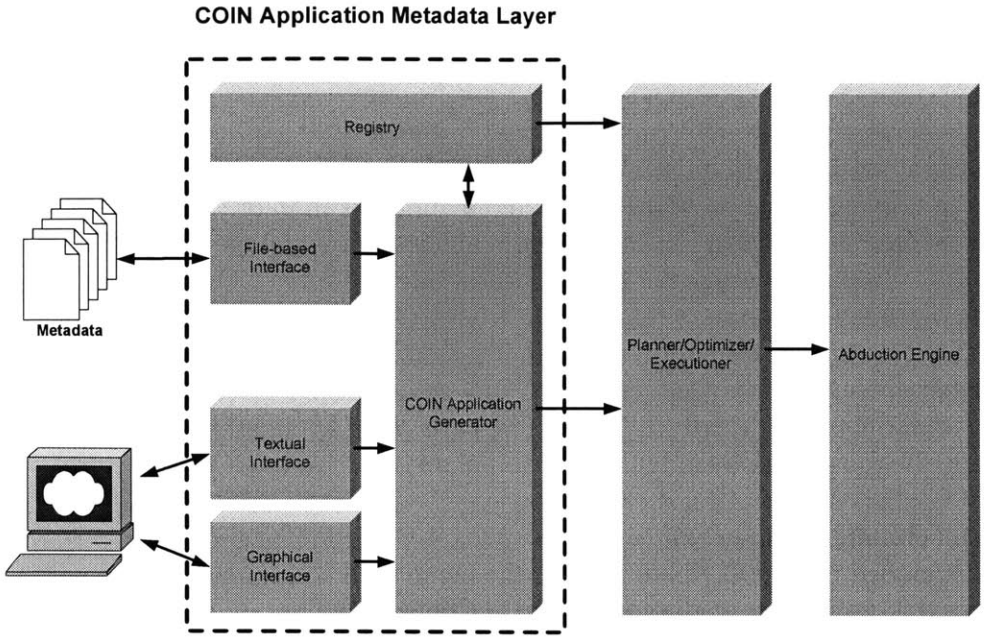


Figure 1.1 – Overall picture of COIN Application Metadata Layer

The work of this thesis constitutes the COIN Application Metadata Layer, as shown in Figure 1.1. This metadata layer shields the users of eCOIN from the need to understand Prolog in order to create COIN application. There are three channels of interacting with the metadata layer, and they are the Textual Interface, the Graphical Interface, and the File-based Interface. The textual interface is designed for human users and is a form-based interface organized to guide the input of every aspects of application metadata. The graphical interface couples with the textual interface to provide the users with a graphical view of the application metadata.

The file-based interface is designed for machine users that already have the application metadata in one of the supported XML-based formats (RDF, RuleML, RFML), and want to make use of the Prolog generator function in the metadata layer to interact with eCOIN. The metadata layer stores application metadata in files under the Resource Description Framework (RDF) format. By representing application metadata in RDF, a channel to the RDF modeling community is established for eCOIN. This will facilitate the import of RDF ontology into COIN application in the future. Since RDF is a XML-based language, it is possible to transform the metadata in RDF into other XML-based languages via Extensible Stylesheet Language Transformation (XSLT). XSLT files are written and used by the metadata layer for transforming RDF metadata into Rule Markeup Language (RuleML) and Relational Functional Markup Language (RFML). This means that application metadata expressed in RuleML or RFML is acceptable by the metadata layer. Furthermore, application metadata is no longer limited to a single representation standard. Just as RuleML and RFML are added to list of supported representation format through XSLT, one can readily write additional XSLT files for

transforming the application metadata between other markup languages. The benefit of separation of knowledge and representation is apparent in this design. COIN application developers no longer need to be proficient in Prolog and any XML-based language can be used to encode the application metadata.

Since XML documents are inherently file-based web resources, the application metadata are stored in files rather than in traditional databases. This means that anyone with access to the internet can download and view the COIN application via an internet browser. Similarly, any machine process can locate the COIN application via the internet. The added accessibility should encourage the reuse of knowledge within a COIN application. Not only are the ontology and context information from COIN a great resource to other information systems, similar information from other systems are great resources to COIN as well. To further encourage knowledge reuse, COIN application metadata is componentized to allow greater flexibility in sharing parts of an application. This design should encourage the reuse of knowledge both external and internal to COIN.

1.1 Context Interchange Framework

The approach developed here at COIN to address the mentioned communication issues is commonly referred to as context mediation [1, 2, 3]. Context mediation is an automated process that, once set in motion, will seamlessly integrates information from different contexts. Setting context mediation in motion is a two-fold procedure. First, the participants must agree on a single language of communication on the semantic level. This means that the context mediation system must somehow know the different representations of objects in different contexts. For example, the language for price in one country is "price", while in another country it is "prix" (French for price). Irrespective of how an object is referred to in different countries, the mediation system must know that "price" and "prix" both refer to the same thing. This is achieved by having a common *ontology* on the domain of communication for all participants. Upon the common ontology, each participant can tell the system how each object in the domain is represented in his own context. The process of mapping vocabulary from one's domain onto the common ontology is known as *elevation* in the COIN framework. In our example, the price attribute from both data sources are elevated to "price" in the common ontology.

However, having a common semantic of the domain is just half the work. The data values that any object may hold could potentially have intrinsic contextual differences. For example, the "price" may have value "100" in division 1. Without additional information, the mediation system would not know the equivalent value of 100 in another context. In one scenario, the user querying for price from division 1 may have the same concept of price as division 1; and so the value "100" is valid in both contexts and no mediation is required. In another scenario, the user may not have the same concept of price as division 1; and so the value "100" from division 1 is something else, i.e. "90", in the user's context. The mediation system must somehow know how to convert values from one context into another. This is achieved by the concept of *modifier*, introduced by the COIN framework. Modifiers are values that indicate to the system if conversions of the data between contexts are necessary. For modifiers to work, every context must

assign a modifier value to each kind of data the system may run across. Possible modifier values for "price" are "EUR" and "USD", which are currency symbols. In the scenario where the modifier values of "price" for two contexts are the same (i.e. both are "USD"), no conversion is necessary and the system will simply pass the data unchanged between the two contexts. In the case where the modifier values are different (i.e. "EUR" and "USD"), then the system will know to convert data between the two contexts. Once contextual differences are detected, the actual conversion of data is governed by the concept of *conversion function*. Conversion functions are mathematical equations or logical expressions which can be evaluated programmatically. Conversion functions need only be defined once per modifier. Once defined, all participants of context mediation can share the same set of functions. Presented so far about the COIN framework is just a high level overview. For a full theoretical explanation of the COIN framework, the reader is referred to the literatures by Goh, et al. [1, 2, 3].

1.2 Related Work and Motivation

A previous effort related to this thesis is the Graphical Metadata Management (GMM) system built by Usman [5]. In a nutshell, GMM is both an user interface and a metadata management system. As an user interface, GMM functions to collect and

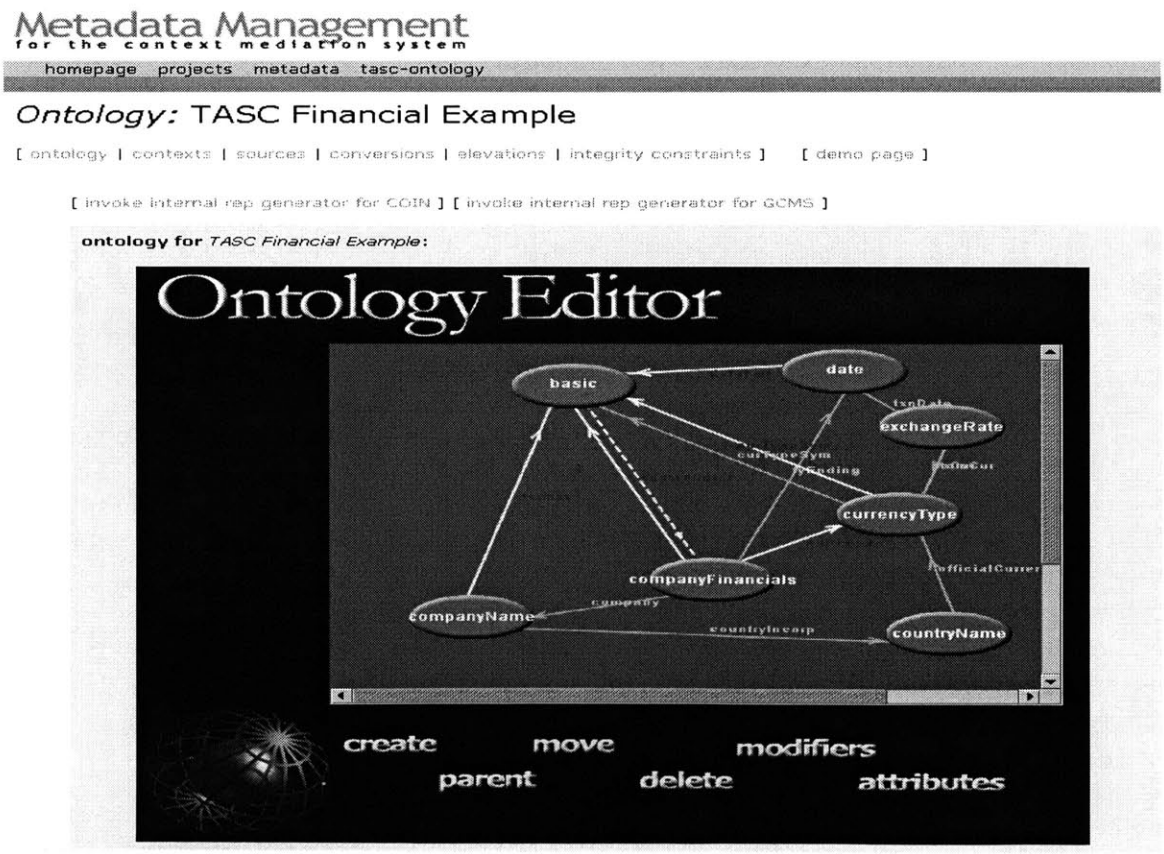


Figure 1.2 – Graphical Metadata Management User Interface

display COIN application metadata via a graphical presentation. A screenshot of the GMM user interface is shown in Figure 1.2. There is a screen for each major components of the COIN application. For example, the ontology editor screen, shown in Figure 1.2,

is where the user can view and edit the ontology of a COIN application. As a metadata management system, GMM stores the information collected from the user interface into a database. The GMM database is a collection of tables that are organized according to the elements of the COIN model. Besides application metadata, graphical information that renders the ontological graph is stored in the database as well. Through the user interface, the user can invoke the Prolog generator of GMM to produce the Prolog application files necessary for context mediation by eCOIN.

While the GMM performs well for its intended usage, there are several considerations of improvement that are not addressed by that system. A desirable area of improvement is the ability to easily share and incorporate metadata between COIN and other modeling communities. Research in area such as ontological modeling is a collaborative effort that spans many research communities. The ability to tap into these communities hinges on the ability to share information with them. To encourage information sharing, COIN metadata must be presentable in a format that is more commonly understandable than Prolog. The work of this thesis picks up in the area of metadata representation. Related to choosing a suitable representation for sharing is the consideration for accessibility, which is an area addressed by this thesis as well.

Another important area of consideration on improving GMM is the completeness of the graphical representation of metadata. The graphical content in GMM doesn't extend beyond the screen shown in Figure 1.2. Although the ontology of a COIN application already tells the viewer a lot about the application, there are other aspects of the application that could be shown graphically as well. For instance, the elevation of data objects onto the ontology is a desirable association that could be represented graphically. By enriching the graphical content, more information can be transmitted to the viewer per screen. The work of this thesis has identified several instances of improvement in this area.

1.3 Objectives

The work of this thesis augments eCOIN in the areas of metadata representation and management. The focus of metadata representation concerns how information is represented for processing and storage, as well as how information is presented to user through the user interface. On the metadata management side, the focus is on how the information is accessed and shared by various processes within eCOIN, as well as by external processes. There are three objectives that drive the design of the system presented in this thesis. The objectives are: (1) separation of knowledge and representation, (2) human aspect, and (3) knowledge sharing and reuse. In Chapter 11, the concluding chapter, we will discuss how well this thesis addresses each of these objectives.

1.3.1 Separation of Knowledge and Representation

In an ideal world, all individuals and machines would communicate to each other under one common language, and there would be just one global representation of knowledge. However, there is no universal representation of knowledge in the real world. Every society, every group, every individual has his or her own preference on the format of

representation. A system that needs to communicate with multiple processes can easily be worn down by the need to maintain multiple representation of the same information. In the system built by this thesis, such problem is avoided by committing to separation of knowledge and representation. In another words, the system would only worry about maintaining its own version of the knowledge, but provides a mean for other processes to easily access the same information in their preferred representation.

1.3.2 Human Aspect

The essential knowledge needed by eCOIN is the metadata that define COIN applications. Human users of eCOIN are the primary source of such metadata. Therefore, the human aspect of the system built by this thesis is an issue that cannot be overlooked. By human aspect, we are referring to the ease of use of the user interfaces of the system, as well as its effectiveness in communicating information to users. Keeping human aspect in high priority avoids any potential bottleneck that may develop between the system and its users on the flow of information.

1.3.3 Knowledge Sharing and Reuse

The proverb "avoid reinventing the wheel" is the principle behind this objective. Within a COIN application, there are significant amount of information that could be potentially shared or reused by another application. For instance, ontologies from different COIN applications tend to build upon each other. This is not a surprising observation because ontology is a way of modeling concepts or things. An ontology that is modeled correctly should almost always have an universal quality to it. Provided that the potential reward of knowledge sharing and reuse is high, the burden of realizing that potential falls heavily on the knowledge management system. The system proposed by this thesis attacks this objective by making knowledge easily accessible, and widely acceptable. Easy access encourages more users to obtain the information, while wide acceptability means more users can use the information obtained. Building COIN application can become a much easier task if this objective is executed successfully.

1.4 Thesis Road Map

This thesis is organized as follows. The various markup language specifications mentioned, as well as existing eCOIN technologies, are described in sufficient details in Chapter 2. The work of this thesis constitutes the COIN application metadata layer. An overview of the metadata layer is shown in Chapter 3. The internal COIN model, which is a data structure that supports the metadata layer, is presented in Chapter 4. The design and user interaction of the textual and graphical interface are detailed in Chapter 5 and 6 respectively. Chapter 7 describes how the COIN application generator produces COIN application in various markup language technologies. The management structure of the COIN registry is presented in Chapter 8. Chapter 9 shows how the technology of this thesis fits into the eCOIN framework. A reasonably sized COIN application, known as Disaster Relief, is presented in Chapter 10, and observations are on the experience with creating the application from the user interfaces. Finally, Chapter 11 concludes the work of this thesis and motivates the reader on future related work.

2 Underlying Technology

The work of this thesis augments the ongoing research effort on context mediation at the Context Interchange (COIN) Group at MIT Sloan School of Management. This section reviews the current mediation technology developed at COIN, as well as some technology central to metadata representation. The technology presented in this section will pave the way for later discussion on the core concepts of this thesis.

2.1 Overview of eCOIN

The goal of COIN system is to achieve semantic integration by mediation of contexts associated with data sources and users. There are four ingredients that make this possible. Figure 2.1 depicts the relationship between the four ingredients, which are COIN Model, extended-COIN (eCOIN), data sources, and the users. Central to the system is eCOIN, which draws upon the COIN model, the data sources, and the users for input necessary to perform context mediation. The COIN model and data sources are *a priori* information about the integration domain that must be supplied to eCOIN, where as user supplied information are gathered at the time of mediation. The COIN model is central to this thesis and a full discussion of it is presented in a later section.

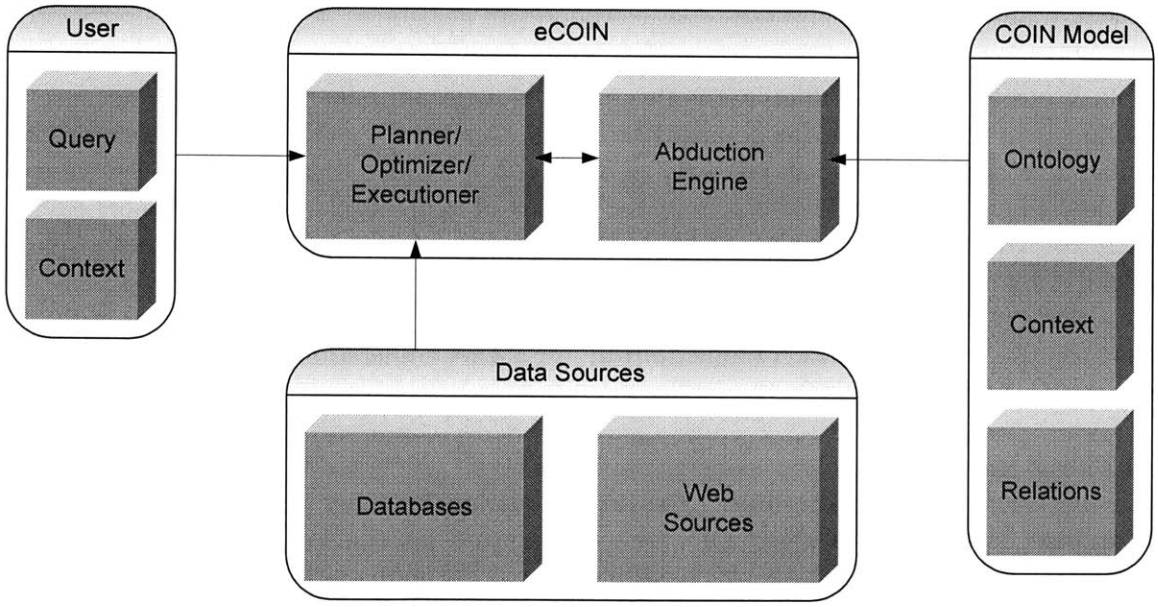


Figure 2.1 – Overview of eCOIN

Semantic integration is not useful if there is no actual data to be integrated on. Data sources provide the data to be integrated on, as well as intermediate data necessary for converting data between sources. Data sources can be database tables of financial data from two companies which wish to have an integrated view of their company financials. The supporting data sources in this case may include currency rates which are needed for converting financial data from different countries. Data sources are not limited to traditional physical databases. The role of World Wide Web as an effective platform of data publishing provides another category of data sources known as web sources. Web sources can range from web pages displayable on a web browser, to web services which are queryable by machines. The Cameleon Web Wrapper [6] is a technology developed

at COIN for the purpose of querying web sources through structured relational queries. eCOIN accepts data from different types of sources, including both databases and web sources.

Another ingredient of context integration is the information supplied by participating users. Users can be an actual person interacting with eCOIN through the user interface, or it can be a software process interacting with eCOIN via procedure calls. Every user must identify the context (the receiver context) he is in. Information that defines a context is predefined in the COIN model. Once the receiver context is known, the system can start to perform an user specified query. The query is in SQL format and encompasses the data sources he wishes to query. The system will know the source context and apply the appropriate mediation rules.

The two major components within eCOIN are the abduction engine and the planner/optimizer/executioner (POE). The abduction engine provides the mediation facility to eCOIN, while the POE carries out the role of obtaining data from data sources and fulfilling the user queries from those data. The first stage of context mediation begins when the POE converts a user-submitted query from SQL format into Datalog format. The conversion is necessary because the context mediation is carried out in the Prolog environment of the abduction engine. During the next stage, a Datalog query and a COIN application in Prolog are sent to the abduction engine where context mediation takes place. The result of context mediation is a mediated query in Datalog, which is sent back to POE. POE must then convert the mediated query back to SQL because the data from data sources are relational in nature. At the final stage of context mediation, the POE issues the mediated query on the relational database and returns the result of execution to the user. The mediation process begins again when another query is submitted by the user.

2.2 Prolog

Prolog stands for Programming in logic. It is a simple, but powerful language that deals with objects and relationships between objects, using only three types of statements: facts, rules, and queries [17]. Control structures are provided by the Prolog interpreter, rather than the programmer. A Prolog programmer only has to describe the solution to a problem, rather than having to tell the computer how to compute the solution. This reduces the programming burden. A COIN application written in Prolog consists solely of rule statements in the following form.

rule(head_statement, body_statement).

A rule statement is a compound statement having a conclusion statement and a premise statement. A conclusion statement, also know as the head of the rule, is an atomic formula with zero or more arguments. The premise, or body of the rule, may be an atomic formula, or a combination of statements. The head of the rule describes what fact the rule is intended to define. The body describes the set of goals that must be satisfied for the head to be true. Notice also that rules are always ended with a dot. Consider the following example.

rule(is_a(X, male), is_a(X, son)).

This simple rule reads like this: *X is a male if X is a son.* The variable in this rule is *X*. Variables do not need to be declared before use, but they are usually denoted by a capitalized first letter, or encased by single quote. When *X* is instantiated to an object in one part of the rule, the same instantiation applies for all parts of the rule. All rule statements take the same form as the example above. In the case where the body statement consists of multiple goals, each goal is evaluated one after the other.

2.3 COIN Model

The COIN model encapsulates the organization of knowledge necessary for semantic integration among heterogeneous data sources. Metadata contained in the COIN model is utilized by COIN system for the purpose of context mediation. The design of COIN model is described in precise details in the works of Cheng Goh, et al. [1, 2, 3]. A primer of COIN model is presented here to aid readers in understanding the work of this thesis.

COIN model is comprised of three components, which are the ontology, context, and source. Each component is described in the subsections that follow. Prolog is the native language that is used to encode COIN application that is processed by abduction engine. A Prolog representation of each concept in the COIN model will be illustrated in the subsections. The readers should not conclude that Prolog is the preferred way to describe COIN model. COIN model is a concept that can be expressed in many programming or modeling languages.

2.3.1 Ontology

Ontology is the hierarchical structuring of knowledge which formally specifies the objects or entities that exist in some area of interest and the relationships that hold among them. The objects in an ontology is labeled semantic type in the COIN model. There are two typical relationships that exist in an ontology, namely inheritance and attribute relationship. COIN model takes ontological relationship a step further by introducing the contextual relationship, called modifier.

Semantic Type

Objects or entities in ontology are identified as *semantic types* in the COIN model. Semantic type, as its name suggests, reflects the type of an object. If semantic type is viewed as a place holder of data, then the data type is the semantic type. There is a special semantic type in the COIN model known as *basic*. It is the most primitive object type that every other object inherits from. Semantic type is always declared in relation to its base type. Therefore, the declaration of semantic type is strongly tied to inheritance relationship, which is described in the next section.

Inheritance

Inheritance represents a parent-child relationship in the object orientation sense between two semantic types. A semantic type must inherit from either *basic* or another semantic

type. In another words, every semantic type, beside *basic*, has an inheritance relationship with another semantic type. Inheritance is declared as follows in Prolog.

rule(is_a(semtypeB, semtypeA), (true)).

This rule statement is an inheritance relationship, which says "*semtypeB is a child of semtypeA*", or "*semtypeA is the parent of semtypeB*". Both semtypeA and semtypeB are the names of semantic type. The above rule statement is also a declaration for the semantic type named *semtypeB*.

There is the concept of a *basic* semantic type in the COIN model which stands for the root of all semantic types. When the ontology is viewed strictly as a COIN ontology, it is understood that a semantic type that has no indicated parent in the ontology is the child of *basic*, since *basic* is the root of every semantic type. All the attributes of the parent semantic type is inherited by its children. Attribute relationship is explained in the next section.

Attribute

Attribute relationship between semantic types in a COIN model describes property of a semantic type. The semantic type that the attribute extends from is known as the domain; and the semantic type that the attribute extends to is known as the range. Attribute is declared as follows in prolog.

rule(attributes(semtypeA, [attribute1, attribute2]), (true)).

This rule statement is an attribute relationship, which says "*semtypeA has the attributes attribute1 and attribute2*". Attributes are declared per semantic type. The domain of these attribute relationships is *semtypeA*. *semtypeA* is declared to have two attributes, namely *attribute1* and *attribute2*. Notice the range semantic type in an attribute relationship is not found in the attribute declaration. The range is found on the attribute axiom, which is described in later section.

Modifier

Modifier modifies the values of semantic types and gives definition to a context (see section on modifier axiom). The semantic type that the modifier extends from is known as the domain; and the semantic type that the modifier extends to is known as the range. Modifier is declared as follows in prolog.

rule(modifiers(semtypeA, [modifier1, modifier2]), (true)).

This rule statement is a modifier relationship, which says "*semtypeA has the modifiers modifier1 and modifier2*". Just like attributes, modifiers are declared per semantic type. The domain of these modifier relationships is *semtypeA*. *semtypeA* is declared to have two modifiers, namely, *modifier1* and *modifier2*. The range in a modifier relationship is not found in the modifier declaration, but in the modifier axiom, which is described in later section.

2.3.2 Context

Context is declared in the COIN model as one of three types: a root context, a context which shares the same definition as another context, or a context which overrides another context. Context is defined by the values of its modifiers. Associated with context as well are the elevation axioms.

Root Context

Context is declared as root as follows in prolog.

rule(is_a(contextA, basic), (true)).

Context declaration syntax is very much like semantic type declaration. A root context is indicated by the keyword *basic*.

Context Sharing

rule(is_a(contextB, contextA), (true)).

This rule statement is a context declaration, which says "*contextB shares the same contextual definition as contextA*". By declaring that *contextB* is a *contextA*, the modifiers for *contextB* will automatically take on the same definitions as *contextA*, without the need to have explicit definition.

Context Overriding

A context which overrides another context is declared the same way as context sharing. The difference is that context overriding occurs when one or more of the modifiers are explicitly declared for the new overriding context.

Modifier Axiom

Modifiers give definition to a context and they are defined per context. Each modifier essentially is a list of values indexed by the context. Each value, known as the modifier value, can be either static or dynamic. A static value is either a string or number value that stays constant. A dynamic value describes how to get to a value without specifically specifying the value at design time. The actual value of a dynamic modifier is not known until run time, and can change as data in the sources change. Associated with each modifier is a conversion function. Conversion function, as its name suggests, is a function which converts a semantic type value between two contexts, based on the modifier defined for the semantic type.

rule(modifier(companyFinancials, O, scaleFactor, disclosure, M), (cste(basic, M, disclosure, 1))).

This is an example of a static modifier axiom, which assigns a constant value to the modifier for the specified context. *companyFinancials* is the semantic type being modified by the modifier *scaleFactor*. The modifier value under the *disclosure* context is the integer *1*.

rule(modifier(companyFinancials, O, currency, disclosure, M), (attr(O, company, Company), attr(Company, countryIncorp, Country), attr(Country, officialCurrency, M))).

This is an example of a dynamic modifier axiom, which dynamically draws the modifier value to an elevated source via attribute axioms. This says that the *currency* modifier in the *disclosure* context has the value held in the semantic type *officialCurrency*. In addition, the path that links the domain *companyFinancials* to the range *officialCurrency* is specified by a list of attributes. In this example, *companyFinancials* is first linked to the company semantic type, then to the *countryIncorp* semantic type, and finally to *officialCurrency*. The attributes used in a dynamic modifier is declared elsewhere as an attribute axioms (see section below).

Attribute Axiom

Like relation, attribute declared between semantic types in the ontology needs to be elevated. The elevation of attributes can span one or more relations. Attribute axioms are used for defining dynamic modifiers, as well as conversion functions.

rule(attr(X, officialCurrency, Y2), ('Currencytypes_p'(X, Y1), 'DStreamAF_p'(_, _, _, _, _, Y2), value(Y1, datastream, Y), value(Y2, datastream, Y))).

The above rule statement is an attribute axiom which maps the attribute *officialCurerncy* between two elevated relations. The domain of the attribute is indicated by the variable *X*, whose data value is obtained from the elevated relation *Currencytypes_p*. The range of the attribute is indicated by the variable *Y2*, whose data value is obtained from the elevated relation *DStreamAF_p*. When attribute axiom involves multiple relations, as in the above example, we need to specify constraint conditions of the join. This is done via a series of *value* statements that trail the rule statement. In the above example, the value of *Y1* from *Currencytypes_p* is constrained to the value of *Y2* from *DStreamAF_p*.

Conversion Function

Conversion function is the function that converts value from one context into an equivalent value in another context. Sometimes conversion functions are straight forward arithmetic operation, and other times they could involve attribute axioms and elevation axioms.

*rule(cvt(companyFinancials, _O, scaleFactor, Ctxt, Mvs, Vs, Mvt, Vt), (Ratio is Mvs / Mvt, Vt is Vs * Ratio)).*

The rule statement above is a conversion function for scaling a number referenced by the semantic type *companyFinancials*. The function calculates the target value *Vt* by multiplying the source value *Vs* by a *Ratio*. The *Ratio* is in turn derived from the dividing the *scaleFactor* value in the source context by the *scaleFactor* value in the target context, where *scaleFactor* is a modifier defined for both contexts.

*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)).*

The rule statement above is a more complicated conversion function involving attribute and elevation axioms. The underlying equation that relates the target value *Vt* to the source value *Vs* is a simple multiplication by a currency factor *Rv*. However, the complexity is introduced by the statements used in establishing the value of the currency factor. The attribute axiom *fyEnding* retrieves the date needed in the currency lookup from the *olsen_p* elevated relation, while the currency names are provided as inputs to the conversion function via the variables *Mvs* and *Mvt*.

Elevation Axiom
A relation is linked to the ontology by a set of elevation axioms. An elevation axiom maps a relation attribute to a semantic type in the ontology. This mapping allows the ontology to be queried.

rule(olsen_p(
skolem(currencyType, Exch, olsen_context, 1, olsen(Exch, Express, Rate, Date)),
skolem(currencyType, Express, olsen_context, 2, olsen(Exch, Express, Rate, Date)),
skolem(exchangeRate, Rate, olsen_context, 3, olsen(Exch, Express, Rate, Date)),
skolem(date, Date, olsen_context, 4, olsen(Exch, Express, Rate, Date))),
(olsen(Exch, Express, Rate, Date))).

The rule statement above defines the elevated relation *olsen_p* from the underlying relation *olsen*. Elevation axiom consists of a name for the elevated relation, a set of skolem functions, and a source signature. The source signature consists of the relation name, and the attributes in the desired order. Each skolem function maps a semantic type to an attribute of the relation. There are as many skolem functions as there are number of attributes in the relation. Multiple attributes can be mapped to the same semantic type.

2.3.3 Source
Sources are the storage of data. Sources can be a database, web wrapper engine, or even web service server, as long as they can be queried. Sources are better known as relation in a COIN model. The distinction between source and relation is subtle. Source is a physical location where queries are sent to. A relation is a relational object inside a source and it serves as the subject of a query. The physical attributes that describe a source is not part of the COIN model, but it is needed by the POE of eCOIN for retrieving the actual data.

Relation
Relation in the COIN model is the metadata equivalent of a physical data source. Multiple relations can be defined for each physical source. A relation consists of a list of attributes which indicates the queryable objects of the relation. In a relation that corresponds to a database table, the attributes corresponds to the columns of the table.

rule(relation(cameleon, olsen, ie, [['Exchanged', string], ['Expressed', string], ['Rate', real], ['Date', string]], cap([[1, 1, 0, 1]], [<, >, <>, =<, >=])), (true)).

The rule statement above is a relation declaration, which consists of a source name, a set of attribute names and their types, and a capability record. The name of this relation is *olsen*. The source is *cameleon*. The relation has four attributes named *Exchanged*, *Expressed*, *Rate*, and *Date*. Their types are indicated alongside attribute names. The value "*ie*" indicates that the relation can be queried both internally and externally. This value is primarily used by the POE of eCOIN in executing a query. A capability record tells which attributes are bounded, and what kinds of constraint operations are supported. For a database table, specifying the primary keys would be sufficient for establishing the bounding condition. A source would also require a connection string, which specifies the location of the source and how to connect to it. The connection information is currently stored separately in a registry, rather than in the COIN model.

Integrity Constraint

Integrity constraint defines the attributes in a relation that must be bounded in order to retrieve a unique record. The integrity constraint is analogous to the capability record of a relation, but integrity constraint is used exclusively by the abduction engine. There could be multiple integrity constraints per relation.

olsen_ic1 @ '?'olsen(A1, B1, C1, D1), '?'olsen(A1, B1, C2, D1) ==> C1=C2.

The above is an integrity constraint for the Olsen currency conversion source. It says that if the first, second, and fourth attributes (*from-currency, to-currency,* and *date*) were bounded, then there is a unique value for the third attribute (*conversion rate*). In other words, the *from-currency*, *to-currency*, and *date* must be specified when querying the Olsen relation for conversion rate.

2.4 Markup Languages

Markup languages are all the tag based data description languages under the family of Extensible Markup Language (XML) [7]. XML is a way to describe structure data through tags similar to those used in HTML. XML tags define elements of the data and the data types of the data. Each element can be as simple as a piece of string, or as complex as a set of other elements. There is no limit to the number of XML tags one can define. There are fundamental differences between XML and HTML. While HTML is designed to display data, XML is designed to describe what the data is.

2.4.1 XML and Its Advantages

XML is a very flexible way to pass data around. It can be used to persist data, or used as document locator to locate web resources. These are just a few of the uses of XML, but important uses that are pertinent to the work of this thesis. XML is text based, which makes them more readable. XML documents can use existing infrastructure already built for HTML. For example, the HTTP protocol used to transport HTML documents can serve to transport XML documents equally well.

XML parsing is well defined and widely implemented, so one can retrieve XML document in a variety of environment. Most web browsers today support XML document viewing. There are at least one XML parser that exists for every common programming platform; and if a platform doesn't yet support XML, one can easily write a parser for it because XML is very well defined. Applications can also rely on XML parsers to do some data validation, such as type checking. XML schema is used to validate XML documents. Schema eliminates much of the need to write validation code. Also, a schema can be shared among organizations to allow data sharing.

There have been many markup languages that emerged under the XML family. Each emerged markup language serves a specialized domain of usage. The Resource Description Framework (RDF), for example, is a XML based language for representing information about web resources. The markup languages pertinent to this thesis are RDF, RuleML, RFML, and HTML. Each of these languages is introduced in subsequent sections, with the exception of HTML. It would be too redundant to prime the reader on HTML because most reader would already have enough working knowledge to render an introduction unnecessary.

2.4.2 Resource Description Framework (RDF)

The Resource Description Framework (RDF) [10], developed by the World Wide Web Consortium (W3C) [8], is an infrastructure for capturing the semantics of metadata. The RDF infrastructure is a simple yet powerful one. It assumes no semantics of the resources it describes. This allows for a wide variety of metadata from different resource communities to be encoded and shared. RDF uses XML as the common language for exchange and processing of metadata. Structural constraints are imposed on XML by RDF to provide a framework for expressing semantics. In addition to being made for machine processing, RDF can be made readable by human, through graph representation. Elements of RDF pertinent to the work of this thesis are presented below. The roles of these elements in the work of this thesis are discussed in later chapters. Readers interested in knowing the full capabilities are encouraged to read the W3C RDF Primer [10].

RDF describes resources. A resource is any object identifiable by an Uniform Resource Identifier (URI) [11]. The most common form of URI is the Uniform Resource Locator (URL), which is a subset of URI. URI describes the mechanism used to access the resource, the computer address of the resource, and the name of the resource on the computer. For example, the URI http://web.mit.edu/index.html identifies the file *index.html* on the computer *web.mit.edu*, and specifies the Hypertext Transfer Protocol (http) as the method of accessing the file. Since an RDF document can compose of multiple resources, an optional fragment identifier is sometimes used in conjunction with URI to locate a specific resource in an RDF document. For example, if within the file *index.html*, there is a resource named *section2*, then this particular resource is located by http://web.mit.edu/index.html#section2. The "#" separator combines an URI and a fragment identifier to produce a URI Reference, or URIref. In this thesis, the term URI and URIref will be used interchangeably. If an URI contains the "#" separator, then it is an URIref; otherwise, it is a plain URI.

Let us look at a short example to illustrate the concepts of RDF. Below are two statements that we will encode in RDF shortly.

(1) The *name* of *Customer123* is *John Smith.*

(2) The *referral* of *Customer123* is *Customer234.*

Each statement is called a *triple*, which consists of three elements, a *subject*, a *predicate*, and an *object*. A *subject* is the resource being described by the statement. *Predicate* is a property of the subject. *Object* describes the value of a particular property. In triple notation, the two statements above are expressed as:

(1) <Customer123> <name> "John Smith"

(2) <Customer123> <referral> <Customer234>

Elements in angle brackets are resources, whereas elements in quotes are strings. Triple notation follows the *(subject, predicate, object)* format. From triple notation, RDF graph can easily be generated by modeling *subjects* and *objects* as nodes, and *predicates* as edges. The Figure 2.2 below is a RDF graph of the customer example. The option to visually inspect a RDF document is one of the many advantages of RDF.

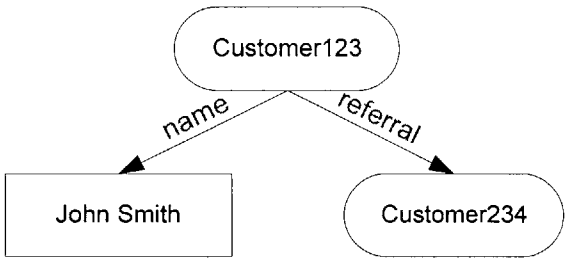


Figure 2.2 – RDF graph of the customer example

A RDF document consists of simple statements about resources. Each of the statements is a triple. Below is a representation of the customer example in RDF. Let's consider the short example line by line.

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.          xmlns:exterms="http://www.example.com/terms/">
4.   <rdf:Description rdf:ID="Customer123 ">
5.       <exterms:name>John Smith</exterms:name >
6.       <exterms:referral rdf:resource="#Customer234"/>
7.   </rdf:Description>
...
8. </rdf:RDF>
```

The first line is a standard XML declaration, which specifies the version of XML the document is encoded in. This helps the application that interacts with the document to pick a correct XML parser. In addition to stating the start of the RDF content in the document, the second and third lines declare the namespaces used in the document. A namespace declaration consists of a name and an URI. Once a namespace is declared, it can be referenced anywhere in the document by its name. Namespace can be thought of

as the prefix that helps to reduce the length of commonly used URI. On line 4, the *rdf* namespace is used on the element *Description*. If the *rdf* namespace were expanded, the *Description* element would read http://www.w3.org/1999/02/22-rdf-syntax-ns#Description. The subject of the *rdf:Description* element, in this case, is "Customer123". Line 5 assigns the string object "John Smith" to the *name* property of the current resource. A property can also take on an URI of a resource as its value. This is demonstrated on line 6, where the *referral* property is an URI of another resource named *Customer234* in the same document. Notice that the lack of prefix in an URI indicates a resource relative to the current document. Line 7 closes the *rdf:Description* element. Line 8 closes the *rdf:RDF* element, and the document ends.

2.4.2.1 – RDF Schema

Different organizations have different views and definitions of even the most common things. RDF schema provides a way for organizations to define their own semantics of resources, and communicates those resource definitions to other interested parties. In addition, RDF schemas can be reused by communities other than the origination. This helps to reduce the number of duplicated schemas on resources and results in a tighter semantic web.

RDF schema is just a RDF document, with the distinguishing quality that it borrows resource definitions from the W3C's RDF schema, *rdf-schema*. The most basic types of resource defined in a RDF schema are classes and properties. The definitions of these resources are obtained from W3C. A RDF class corresponds to a class in object-oriented languages. Any kind of things can be represented by RDF classes. RDF properties are attributes of RDF classes. Both domain and range of a RDF property are RDF class.

So far, we have demonstrated our customer example using elements from some predefined RDF schema. But if we assume that the Customer class from the example above was never defined, then we would need to define our own definition for the class in a RDF schema. Let's define a RDF schema for our customer example presented earlier.

```
1. <?xml version="1.0"?>
2. <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
4.   <rdfs:Class rdf:ID="Customer"/>
5.   <rdfs:Property rdf:ID="name">
6.     <rdfs:domain rdf:resource="#Customer"/>
7.     <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
8.   </rdfs:Property>
9.  <rdfs:Property rdf:ID="referral">
10.    <rdfs:domain rdf:resource="#Customer"/>
11.    <rdfs:range rdf:resource="#Customer"/>
12.  </rdfs:Property>
13. </rdf:RDF>
```

XML declaration is stated as usual on line 1. Lines 2 and 3 specifies the namespaces used in the document. The *rdfs* namespace points to the W3C's RDF schema that defines RDF class and property. Line 4 declares the *Customer* class. RDF classes defined here are instances of the RDF-defined class *rdfs:Class*. Lines 5 to 8 define the *name* property, whose domain is of type *Customer*, and range is of type string. RDF properties defined here are instances of the RDF-defined class *rdfs:Property*. The domain of a property is the class that this property applies to. The range of a property specifies which class that the value of this property can take on. Thus, the *name* property belongs to *Customer* and its value must be of type string. Lines 9 to 12 define the *referral* property of *Customer*. This property differs from the first in that its range is not of type string, but of type *Customer*. This illustrates that we can readily use any classes defined in the same RDF document as the range. Line 13 ends the RDF document. The RDF elements presented in this brief introduction will be useful for later discussion of the COIN metadata in RDF representation.

2.4.3 Rule Markup Language (RuleML)

RuleML is a XML-based language for representation of rules and facts [12]. Different classes of rules are represented by different set of XML tags in RuleML. We are particularly interested in RuleML treatment of derivation rules, as the Prolog rules used in abduction engine belong under this class of rules. Derivation rules are rules that assert conclusions when certain preliminary conditions are recognized. When applied backward, we can prove conclusions from the premises. Similarities between RuleML and Prolog are pointed out at various point in the discussion of the coming example. The basic syntax for declaring a derivation rule in RuleML is as follows:

<imp><_head>*conclusion***</_head><_body><and>***premise1*...*premiseN***</and></body></imp>**

The rule enclosing tag is **<imp>**, which is similar to the *rule()* declaration in Prolog. Every rule has a head and a body, enclosed by the **<_head>** and **<_body>** tags respectively. Since Prolog is a positional language, no special keyword is necessary for declaring the head and body of rule in Prolog. Instead, Prolog relies on the convention about the positions of rule elements. The first clause of a rule in Prolog is always the head of the rule. The subsequent clauses form the body of the rule. Within the head of a rule is the conclusion of that rule. The body of a rule is the set of preliminary conditions, or premises, that must be satisfied in order for the conclusion to be true.

Let us expand on the customer example from the introduction on RDF by turning it into the simple rule below.

A customer's referral is Customer234 if that customer's name is John Smith.

This rule in RuleML is as follows:

```
<imp>
 <_head>
  <atom>
   <_opr><rel>referral</rel></_opr>
```

```
    <var>customer</var>
    <ind>Customer234</ind>
   </atom>
  </_head>
  <_body>
   <atom>
    <_opr><rel>name</rel></_opr>
    <var>customer</var>
    <ind>John Smith</ind>
   </atom>
  </_body>
</imp>
```

The same rule in Prolog is as follows:

rule(referral(Customer, "Customer234"), name(Customer, "John Smith")).

The head of the rule contains the conclusion "a customer's referral is Customer234". Customer is a variable, as indicated by the <var> tag surrounding it. In Prolog, a variable is simply indicated by a string with the first letter is capitalized. The constant "Customer234", as indicated by the <ind> tag, references a specific customer. The relation "referral", which has the operator role, is indicated by the <rel> and <_opr> tags. Each clause in the rule is labeled as an atom. Following the head of a rule is the body, which reads "a customer's name is John Smith". Notice that in Prolog, the necessity for tags is gone because the elements are positioned. The predicate of a clause, such as "name", always appears in front of the parenthesis. The first element within the parenthesis in a Prolog clause is always the subject, such as customer. The rest of the elements in a clause are objects of the predicate. By using tags and not relying on positional declaratives, RuleML makes no assumption about the recipients of the information. This is indeed a major advantage of XML-based representation. For a more in depth look at RuleML, the reader is encouraged to read the concise version of the design document on RuleML [13].

2.4.4 Relational Functional Markup Language (RFML)

RFML is a XML-based language for representing relational-functional information. This information can be either relational or functional. Examples of relations include the tables of relational databases, and clauses of Horn logic. Examples of functions include mathematical equations, and algorithms that can operate on data structures. Since Prolog, or Horn logic, is relational in nature, we will be focusing primarily on the relational aspect of RFML. Both RuleML and RFML are languages for describing rules, but RFML differs from RuleML in that RFML is designed for describing a specific subset of rules, or relational-functional rules. Another important difference between RuleML and RFML is that RFML relies on some positional declaration. For instance, the predicate of a clause in RFML is always the first element in the clause. This is exactly the same positional convention used in Prolog. As will be apparent in the coming example, RFML statements are less cumbersome than RuleML because fewer tags are required due to

positioning of elements. We are interested in RFML representation because it is a closer relative to Prolog, and makes for a good transition representation to RuleML. The basic syntax for declaring a Horn clause in RFML is as follows:

<hn><pattop>*conclusion***</pattop><callop>***premise1***</callop>**...**<callop>***premiseN***</callop>**<**/hn>**

The tag enclosing a Horn clause statement is <hn>. Every Horn clause has a head and a body, enclosed by the <pattop> and <callop> tags respectively. Similar to RuleML, the head of a clause is the conclusion, where as the body is a set of preliminary conditions, or premises, that must be satisfied in order for the conclusion to be true. Let us examine the same simple rule we have examined for RuleML, which is repeated below.

A customer's referral is Customer234 if that customer's name is John Smith.

This rule in RFML is as follows:

```
<hn>
  <pattop><con>referral</con><var>customer</var>
    <con>Customer234</con></pattop>
  <callop><con>name</con><var>customer</var>
    <con>John Smith</con></callop>
</hn>
```

The head of the Horn clause contains the conclusion "a customer's referral is Customer234". Customer is a variable, as indicated by the <var> tag surrounding it. The predicate "referral", following the Lisp-like Relfun convention, is always the first element in a <pattop>. The same positioning convention applies for predicates in <callop> as well. Predicates are denoted as constant by the <con> tag. For a more in depth look at RFML, the reader is encouraged to read the specification document on RFML [14].

2.5 eXtensible Stylesheet Language Transformation: The Encompassing Link

So far we have introduced a couple of XML-based languages, such as RDF, RuleML, and RFML. XML document, on its own, is just a mean for storing and structuring of information, such that the information can be communicated from one piece of software to another. However, with the introduction of eXtensible Stylesheet Language transformation (XSLT), a lot more could be done with XML formatted data. There are two main usages for XSLT. One usage is the use of XSLT to query and manipulate the information stored within XML documents. On this regard, XSLT can be compared to SQL for relational databases. XSLT helps to turn an ordinary XML document into query-able source. Another usage of XSLT is on the transformation of XML documents between different XML-based formats. It is this capability that allows XML documents to achieve the separation of knowledge and representation. Let us demonstrate these usages through the customer example we have already seen.

The RDF statements below will be the input to a XSLT conversion.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.com/terms/">
  <rdf:Description rdf:ID="Customer123">
    <exterms:name>John Smith</exterms:name >
    <exterms:referral rdf:resource="#Customer234"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Customer234">
    <exterms:name>Mark Peter</exterms:name >
  </rdf:Description>
</rdf:RDF>
```

The RDF above encodes information, such as names and referrals, about customers. Next, consider the XSLT below, which will be used to process the RDF.

```
<xsl:template match="rdf:RDF">
  <html>
    <head><title>List of Customers</title></head>
    <body>
      <xsl:apply-templates select="rdf:Description"/>
    </body>
  </html>
</xsl:template>
<xsl:template match="rdf:Description">
  <p><xsl:value-of select="exterms:name"/> </p>
</xsl:template>
```

The XSLT above consists of two templates. A template statement in XSLT is similar to a select statement in SQL. The first template statement *<xsl:template match="rdf:RDF">* tells the XSLT processor to find all nodes labeled "rdf:RDF". The second template *<xsl:template match="rdf:Description">* asks for all nodes labeled "rdf:Description". Since each of our *Description* nodes is a customer, we are really telling the processor to look at all customers. Within the first template, there are HTML code which will be outputted by the processor, as well as a call to the template on customers. Inside the template on customer is a select statement, <xsl:value-of select="exterms:name"/>, which returns the name of a customer. The output of processing the RDF source with the XSLT above is shown below.

```
<html>
  <head><title>List of Customers</title></head>
  <body>
    <p>John Smith</p>
    <p>Mark Peter</p>
  </body>
</html>
```

This example illustrates the querying capability of XSLT, as well as its transformation capability. The original RDF source is a data source on customers. Customer by customer, the XSLT extracts the name information and displays them in HTML fashion. Notice that the information from the RDF input source is carried over to the HTML output. This transformation highlights the separation of knowledge and representation. Through XSLT, information that is normally existed in machine readable format can now be displayed for humans to understand. A very rich site to find out the full capabilities of XSLT is at the W3C site [16].

3 Design Overview

3.1 Overview of the System

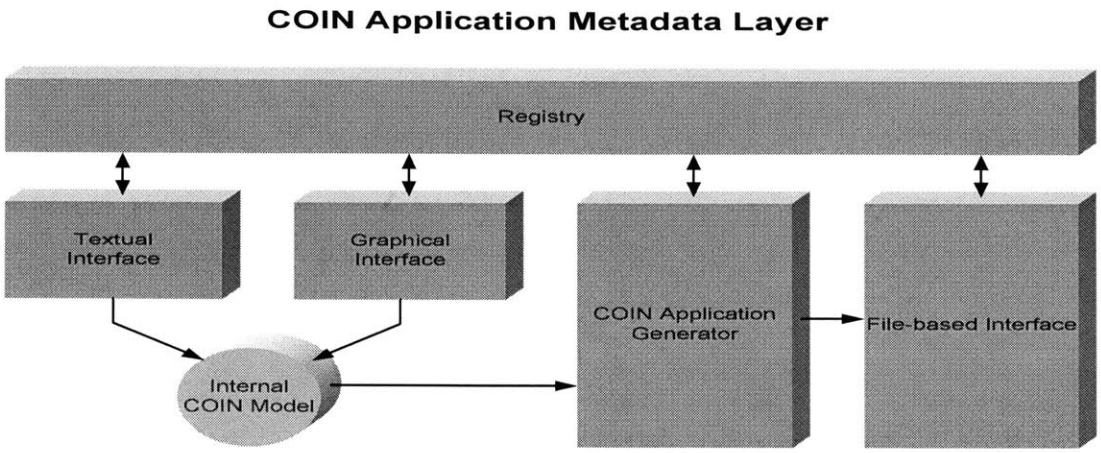


Figure 3.1 – Architecture of COIN Application Metadata Layer

At W3C, *metadata* is defined as machine understandable information for the web [9]. Metadata at COIN is machine understandable information for context mediation. The organization of metadata at COIN is defined by the COIN Model. At the conceptual level, the COIN model consists of ontological information, contextual information, and source information that together enables context mediation. From the information gathering stage to the storage stage, these metadata must take on the appropriate representation at each stage in order to be effective. On the user side, the layout of the user interface is a kind of representation designed to facilitate the collection of information. The processing stage can benefit from an efficient storage format. Finally, the ability to share and reuse metadata is an important factor in the design of the metadata representation.

The work of this thesis compliments the mediation engine (eCOIN) in three ways: (1) it enables the reuse of metadata through XML based representation; (2) it provides an user interface for developing application that runs on eCOIN; (3) it establishes a registry for managing metadata. The design of this work is summarized in the Figure 3.1. At the heart of the design is the Internal COIN Model (ICM). This is the transient data structure that store application metadata at run time. The ICM connects each component of the COIN Application Metadata Layer. The Text Interface and Graphical Interface populate the ICM with user inputs, and convert information in the ICM into screen displays. Application metadata in the ICM is persisted into files in RDF format. Metadata in RDF

format is translated into RuleML, RFML, and Prolog through a series of XSLT transformation produced by the Application Generator. The Prolog version of application metadata is consumed by eCOIN for context mediation. The location of the application files are registered in the Registry, which is needed both for the retrieval of application metadata by the ICM and the eCOIN. Each component of the design is described in length in subsequent sections. The motivational example presented in the introduction section will serve to illustrate the representation of metadata at each stage.

3.2 Motivational Example

A simple COIN application is presented in this section to facilitate the demonstration of various concepts introduced in this thesis. Since the work of this thesis is heavily involved in the representation of metadata, a single consistent example can serve to elucidate the differences between the various representations. The COIN application that is our example is a simple application on tents. This is part of a larger application for disaster relief. Often times in disaster relief situations, tents are deployed to house the disaster victims. Some of the factors that determine which types of tents are deployed may include the size and weight of the tent, the number of persons it fits, the seasons it is suitable for, and the price. Since our example is primarily for highlighting the elements of a COIN application, we will limit the scope by considering just the weights of the tents. Let us examine this application in details.

The ontology of this application is shown in Figure 3.2. There are four semantic types in the ontology, namely *basic*, *product*, *tent* and *weight*. Inheritance relationship is represented by solid line labeled "is_a". Unless noted otherwise, every semantic type shown in the ontology without inheritance is assumed to inherit from *basic*. The only non-basic inheritance in this example is the inheritance of *tent* from *product*. Attribute relationship is represented by solid line with label other than "is_a". The single attribute in this example is the *weight* attribute of *product*. The third and last kind of relationship, modifier relationship, is represented by dashed line. The single modifier in this example is the *weightUnit* modifier of *weight*.

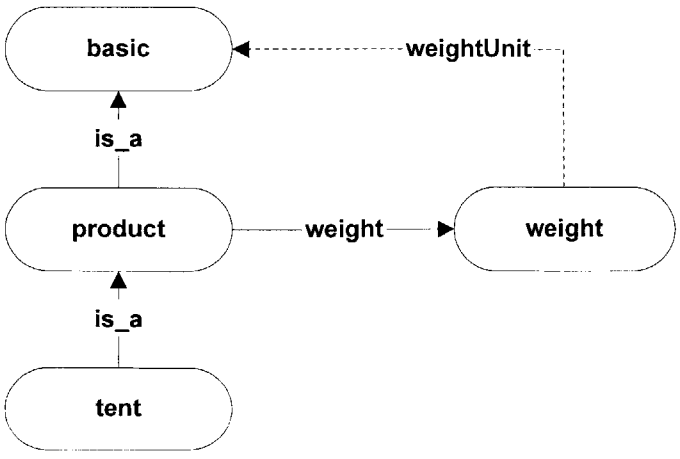


Figure 3.2 – Sample COIN Application on Tents

There is one relation in this tent application, which is summarized in the table below. The name of the relation is *dsr_tent_terranova*. The column names are shown in the left column and their types are in the right column.

dsr_tent_terranova	
Maker	String
Model	String
Name	String
Seasons	String
Sleeps	String
Minimum_weight	Number
Floor_area	String
Price	Number

Figure 3.3 – Relation *dsr_tent_terranova* of tent application

The company Terranova is an outdoor supplies company that sells tents. The relation above describes the attributes of tents. These attributes are maker, model, name, seasons, sleeps, minimum weight, floor area and price. Seasons attribute is a code that indicates which seasons of the year a tent is suitable for. There is usually a temperature range associated with each season code. Sleeps attribute indicates the number of persons the tent manufacturer thinks can comfortably fit in a tent. Since this number is often open for interpretation, it makes for an interesting point for context mediation. However, we will only be primarily interested in the minimum weight attribute in our simple tent example.

The contexts in this application are *c_us* for United States context, and *c_uk* for United Kingdom context. The tent supplier is a company located in the UK, so it is under the context *c_uk*. Users who query the application can be from either US or UK. The values of modifier *weightUnit* are lb under *c_us* context, and kg under *c_uk* context. These modifier values are used in the detection phase of the mediation process to detect for potential contextual conflict. The conversion function related to weight conversion is a simple unit of measurement conversion, which multiplies a quantity in one unit by a unit factor to get the same quantity in another unit. The unit factor is provided by a web source and is part of the standard function library in COIN. The relation *dsr_tent_terranova* is elevated to *dsr_tent_terranova_p* under the *c_uk* context. Since we are primarily interested in the minimum weight attribute of tent in this example, all columns are elevated to *basic* with the exception of *Minimum_weight*, which is elevated to the semantic type *weight*.

This COIN application on tent will be used in subsequent chapters to demonstrate (1) input of metadata through the textual interface, (2) the graphical interface, and (3) metadata representation in different formats. In the textual interface chapter, we will demonstrate the entering and editing features using the metadata presented here on tent application. In the graphical interface, we will demonstrate how the same metadata can be displayed graphically. Finally, we will illustrate how separation of knowledge and representation is achieved by casting this set of metadata into different formats.

4 Internal COIN Model (A Data Structure)

The Internal COIN Model (ICM) is a transient data structure that stores the application metadata at run time. As data storage, the ICM has the dual purpose of accepting data and providing data. User interface calls upon the ICM to provide the application metadata that populates the screen. When user modifies the COIN model via the interface, the changes are stored in the ICM. Similarly, application metadata stored on files are loaded into the ICM, which can then be used by the user interface or other translation tasks. When the user is finished with editing a COIN application, the data inside the ICM is persisted in file formats for immediate access by eCOIN, or for future retrieval.

In designing the ICM, there are two requirements that must be satisfied: (1) the ICM must be able to completely define the COIN Model as described in the introduction section; (2) the ICM should be tolerant to future changes to the COIN Model. Each element of the COIN Model has a counterpart in the ICM. To illustrate that the ICM does indeed completely define the COIN Model, the motivational example, which is presented in prolog in the introduction section, will be expressed in the ICM step by step. Once the readers see that the metadata in a COIN model can be contained in the ICM, they should be convinced that the ICM satisfies the first requirement. The last subsection describes an application programming interface (API) of ICM. The API is the answer to the second requirement. As long as programs interact with the ICM through the API, the programs will be well shielded from future changes to the COIN Model.

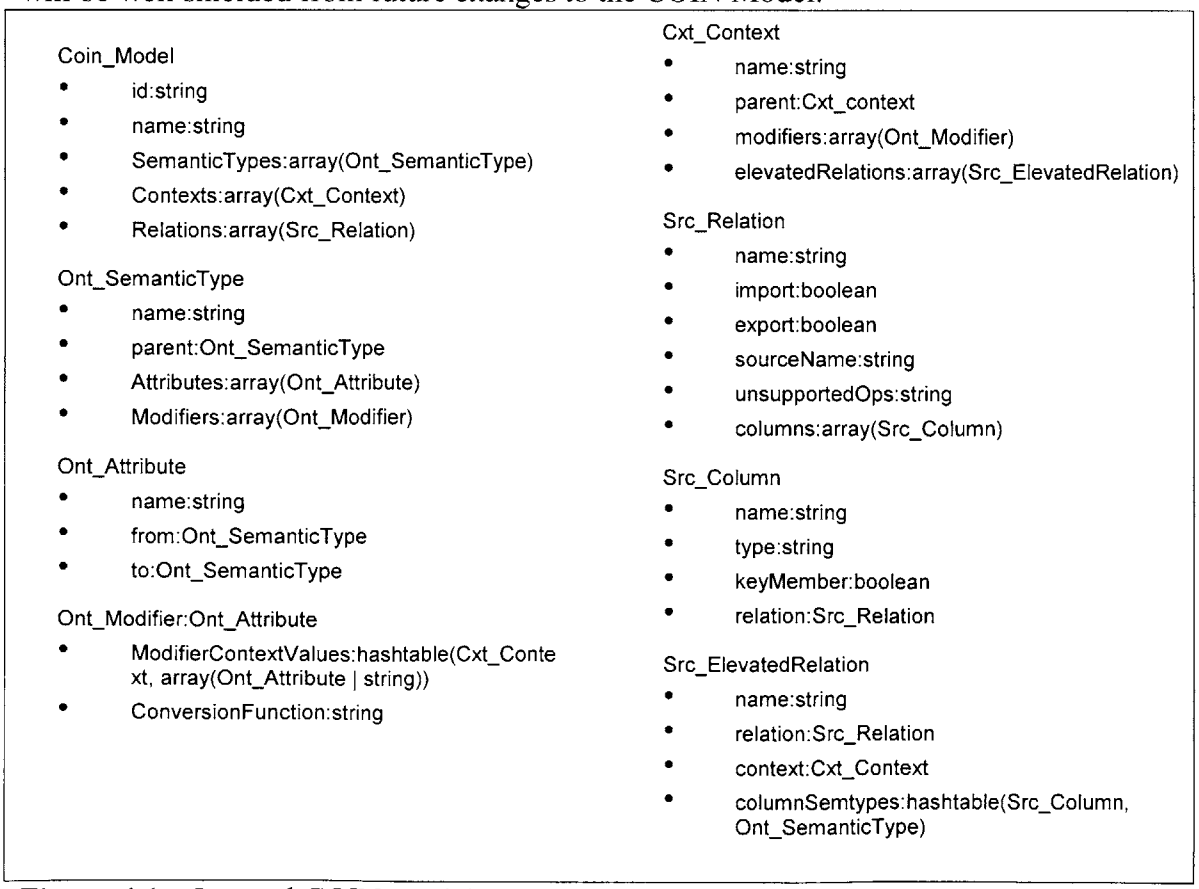


Figure 4.1 – Internal COIN Model at a glance

ICM is a data structure in object-oriented programming language. In object-oriented programming language, data structures are called *classes* and the functions which operate on these classes are *methods*. Classes have *properties* that define the data that makes up the characteristics of these classes. Each property stores a single data type, whether it is a text, a number, or a whole other class. Object orientation is a design methodology, and thus it is possible to write the ICM using programming languages such as C++, C#, and Java. Since the user interfaces are written in C#, the ICM is written in C# as well for compatibility. Figure 4.1 presents a summary of the ICM. Coin_Model, Ont_SemanticType, Ont_Attribute, Ont_Modifier, Cxt_Context, Src_Relation, Src_Column, and Src_ElevatedRelation are class objects of the ICM. The bullet points under each of these classes are their properties. The subsections that follow explore each class in ICM in details.

4.1 ICM – Coin_Model

The Coin_Model class in ICM is the top level data structure from which all the components of a COIN model can be reached. The properties of Coin_Model are:

Id : string
Name : string
SemanticTypes : array(Ont_SemanticType)
Contexts : array(Cxt_Context)
Relations : array(Src_Relation)

The *id* property corresponds to the application ID of a COIN model. The id property uniquely identifies a *Coin_Model*, just as the application ID is the unique identifier for a COIN application. The *name* property is the name of the COIN application and it does not have to be unique. The *SemanticTypes* property is an array of type *Ont_SemanticType*, and it holds the semantic types of the ontology of a COIN model. More over, *SemanticTypes* is the linkage to the metadata that describes the entire ontology. This will be clear in the Ont_SemanticType section. The *Contexts* property is an array of type *Cxt_Context*, and it holds the contexts defined in the model. All context dependent elements of the COIN model, such as modifiers and elevation axioms, are accessed through *Contexts*. The section below on *Cxt_Context* will make this clear. Finally, the *Relations* property is an array of type *Src_Relation*, and it holds the relations used in the model. Below is an instance of *Coin_Model* populated with metadata from the tent example.

Id : "392"
Name : "Simple Disaster Relief"
SemanticTypes : [basic, product, tent, weight]
Contexts : [c_us, c_uk]
Relations : [dsr_tent_terranova]

As seen in the introduction section where the tent example is described, the *id* of the application is `392`, and the *name* is `Simple Disaster Relief`. The *SemanticTypes* of this model are `basic`, `product`, `tent`, and `weight`. Each of these is an *Ont_SemanticType* object, which has its own properties that describe it. *Contexts*

defined in this model are `c_us` and `c_uk`, both are *Cxt_Context* objects. There is only one relation, `dsr_tent_terranova`, defined in this model and it is contained in the *Relations* array. It should be apparent that the *Coin_Model* class can be viewed as a container of objects that link each aspect of the COIN model together. To explore an aspect of the model, one can dive into the corresponding object to retrieve the information. Fortunately, surrounding the *Coin_Model* is an API that makes it unnecessary to expose the structures of its inner properties. The API is described in details in later section.

4.2 ICM – Ont_SemanticType

The Ont_SemanticType class in ICM is the data structure which stores the semantic type of a COIN model. The properties of Ont_SemanticType are:

Name : string
Parent : Ont_SemanticType
Attributes : array(Ont_Attribute)
Modifiers : array(Ont_Modifier)

The *name* property holds the name of a semantic type. The name of a semantic type is globally unique in the COIN model. The *Parent* property points to another Ont_SemanticType object that is the parent of the current semantic type. Thus, an inheritance relationship is established between two semantic types. The *Attributes* property is an array of Ont_Attribute objects. This array stores all the attributes that has the current semantic type as their domain. The Ont_Attribute class is described in details in the next section. The *Modifiers* property is a array of Ont_Modifier objects. This array stores all the modifiers that has the current semantic as their domain. The Ont_Modifier class is described in a coming section. If the Ont_SemanticType class is viewed graphically in an ontology, the class covers the node which represents the semantic type, and all the edges (attributes and modifiers) that start from the semantic type. As you will recall, an ontology consists of semantic types, inheritances, attributes and modifiers. Since each Ont_SemanticType object holds its associated inheritance, attributes and modifiers, a collection of all Ont_SemanticType objects is sufficient to describe the whole ontology of a COIN model. This is the reason we say that the an entire ontology can be described by the SemanticTypes array property. Below are instances of Ont_SemanticType populated with metadata from the tent example.

Name : "product"
Parent : basic
Attributes : weight
Modifiers : null

Name : "tent"
Parent : product
Attributes : null
Modifiers : null

Name : "weight"
Parent : basic
Attributes : null
Modifiers : weightUnit

Product, tent, and *weight* are the three semantic types of the tent example. The *basic* semantic type we didn't mention because it has no parent, attributes and modifiers. The attribute *weight* is described under the *product* semantic type. The modifier *weightUnit* is found in the *weight* semantic type. The inheritance relationship between *product* and *tent* is found in the *tent* semantic type. This is because the *is_a* relationship originates from *tent*. Every other semantic type inherits from *basic*. This indeed does fully describe the entire ontology of the tent example.

4.3 ICM – Ont_Attribute

The Ont_Attribute class in ICM represents the attribute of COIN model. The properties of Ont_Attribute are:

Name : String
From : Ont_SemanticType
To : Ont_SemanticType

The *name* property holds the name of an attribute. An attribute name is the unique only local to the semantic type that it originates from. This means that a semantic type cannot have two attributes, with the same name, originating from it. However, attributes from different semantic type can have similar name. When referring to an attribute, the semantic type name and attribute name together establish an unique identifier. The *from* property holds the Ont_SemanticType object that is the originating semantic type of the attribute. The *to* property holds the destination semantic type object. Below is an instance of attribute from the tent example.

Name : "weight"
From : Product
To : Weight

The above instance describes an attribute named *weight*, which connects from the semantic type *product* to the semantic type *weight*. The fully qualifying identifier for this attribute is the combination of semantic type name *product* and attribute name *weight*. If we adopt the convention of using a dot to connect the two names, we have *product.weight* as the identifier of this attribute. Notice the clarification on the fully qualifying identifier of objects in the ICM is important for representing the COIN model external to ICM. Since ICM is developed in an object-oriented platform, objects within ICM are referenced not by id, but by the actual object. In the textual interface, which is discussed in a later section, the names of attributes are displayed in its fully qualifying form to allow the user to distinguish between attributes.

4.4 ICM – Ont_Modifier

The Ont_Modifier class in ICM represents the modifier of COIN model. It is a subclass of Ont_Attribute, so it inherits the properties *name*, *from*, and *to* from Ont_Attribute. The properties of Ont_Modifier are:

Name	:	String
From	:	Ont_SemanticType
To	:	Ont_SemanticType
ModifierContextValues	:	hashtable(Cxt_Context, array(Ont_Attribute | string))
ConversionFunction	:	String

The *name* property holds the name of a modifier. Similar to an attribute, a modifier name is the unique only local to the semantic type that it originates from. So no two modifiers from the same semantic type can have the same name. When referring to a modifier, the semantic type name and modifier name together establish an unique identifier. The *from* property holds the Ont_SemanticType object whose value is being modified. The *to* property holds the destination semantic type object that the original semantic type is being modified to. The ModifierContextValues property is a hashtable whose keys are Cxt_Context objects, whose values are arrays of Ont_Attribute objects or string object. This hashtable stores the values of the modifier for each given context. Cxt_Context, which is described in the next section, is a class that represents context in the COIN model. For each context key stored in the hashtable, there is a corresponding modifier value for that context. The modifier value is being stored in an array of type Ont_Attribute or type string. The reason there are two possible types of array is because modifier values can be either static or dynamic. Static modifier value is string constant that requires no interpretation before it is used in conversion functions. Dynamic modifier value is a run-time value derived from certain semantic type, whose value is mapped to a relation. In specifying a dynamic modifier value, a series of attribute relationship is used to relay the current semantic type to the target semantic type. Finally, the ConversionFunction property is a string which stores either the name of a conversion function predefined in the function library, or a user defined function written in Prolog. The conversion function library is a feature which is related to the textual interface and the COIN registry; the library will be discussed in details in those sections. Below is an instance of a modifier in the tent example.

Name	:	"weightUnit"
From	:	Weight
To	:	Basic
ModifierContextValues	:	((c_us, "lb"), (c_uk, "kg"))
ConversionFunction	:	"lib_physical_unit"

The above instance describes a modifier named *weightUnit*, which connects from the semantic type *weight* to the semantic type *basic*. The fully qualifying identifier for this modifier is the combination of semantic type name *weight* and modifier name *weightUnit*, or *weight.weightUnit*. Since the *weightUnit* modifier modifies weights from one physical unit to anther, the name of weight units are used as modifier values. There are two modifier values stored in the *ModifierContextValues* hastable; one is the string

"lb", or pound, for *c_us* context, and another is the string "kg", or kilogram, for the *c_uk* context. The Cxt_Context object *c_us* is a context representing the United States, and *c_uk* is the context object representing the United Kingdom. The *ConversionFunciton* property stores the string "lib_physical_unit", which is the name of a predefined library function for converting numeric values between physical units. At the Prolog generation stage of the COIN Application Model Layer, the function name will be substituted by the actual Prolog code of the function. The conversion function library will be discussed in the section on COIN registry.

4.5 ICM – Cxt_Context

The Cxt_Context class in ICM represents the context of COIN model. All context related metadata, such as modifiers and elevations, are accessible from this class. The properties of Cxt_Context are:

Name	: String
parent	: Cxt_Context
modifiers	: array(Ont_Modifier)
elevatedRelations	: array(Src_ElevatedRelation)

The *name* property stores the string that represents the name of the context. Context name is globally unique in a COIN model. The *parent* property points to a context whose characteristics this context shares. Context sharing is the concept where a previously defined context can share its metadata with other contexts. This eliminates the need to repeat the same context information across different context label. An example of context sharing is that of a British company sharing the context defined for the country of Great Britain. The *modifiers* property is an array of Ont_Modifier objects, which represent the list of modifiers that have values defined under the current context. This information duplicates the information found under the ModifierContextValues hashtable property of Ont_Modifier; however, accessing modifier values from a given context is a frequent event that justifies duplicating this information in the Cxt_Context class. This design makes the ICM more efficient in retrieving information at the cost of extra maintenance on synchronizing the information in both classes. Fortunately, the users of ICM would interact with the API, which automatically takes care of this maintenance. The elevatedRelations property is an array of Src_ElevatedRelation. The class Src_ElevatedRelation, which is described in the coming section, stores the elevation metadata for relations of COIN application. The elevatedRelations array is a list of elevation axioms from all relations whose context is the current context. Below are instances of the contexts in the tent example.

Name	: "c_us"
parent	: null
modifiers	: (weightUnit)
elevatedRelations	: null

Name	: "c_uk"
parent	: null

modifiers : (weightUnit)
elevatedRelations : (dsr_tent_terranova_p)

Under the context *c_us*, there is no parent context and the modifier *weightUnit* has a value defined for it. Under the context *c_uk*, there is no parent context and the modifier *weightUnit* has a value defined for it as well. In addition, *c_uk* is the context of the elevated relation *dsr_tent_terranova_p*. Note that relation itself does not belong to any context. It is the elevation information that is context specific.

4.6 ICM – Src_Relation

The Src_Relation class in ICM represents the relation of COIN model. All information regarding a relation, including the column information, is accessible from this class. The properties of Src_Relation are:

Name : string
import : boolean
export : boolean
sourceName : string
unsupportedOps : string
columns : array(Src_Column)

The *name* property stores the string that represents the name of the relation. The *import* and *export* properties are boolean values that indicates whether the relation can be queried by user of eCOIN (export), or by eCOIN alone (import). The *import* and *export* properties are not used by the abduction engine, but by the POE in determining which relations are exposed to users. The *sourceName* property is the name of the source where the current relation is located. If the relation is a database table, then the *sourceName* would be the name of the source database as referenced in the COIN registry. Since source information are often applicable across different COIN application, this information is stored in the COIN registry instead of replicated in every COIN application. Only the name of the source is enough to retrieve the source information from the COIN registry. The *unsupportedOps* property is string that represents which operations are not supported in querying the current relation. There are five possible operations that are unsupported, and they are <, >, =, <=, >=; these operations correspond to "less than", "greater than", "equal to", "less than or equal to", and "greater than or equal to" respectively. For purpose of exporting COIN metadata to XML files down the metadata representation pipeline, the operation symbols are stored after being transformed respectively into "lt", "gt", "et", "le", and "ge". Such transformation prevents the "<" or ">" symbols from being mistook for the start or end of a XML tag by a XML processor. Commas separate the operation in the *unsupportedOps* string; for example, "lt,gt,et,le,ge". Finally, the *columns* property is an array of Src_Column objects. The Src_Column class stores information about individual column of a relation. Below is an instance of a relation in the tent example.

Name : dsr_tent_terranova
import : true

export	:	true
sourceName	:	oracle
unsupportedOps	:	null
columns	:	(Model, Usage, Persons, Weight, Packed_length, Packed_width, Interior_length, Interior_width, Interior_height, Price)

The name of the relation is *dsr_tent_terranova*, and it can be queried both internally and externally. The name of the source, as appeared in the COIN registry, is *oracle*. Notice that while *oracle* may represent a particular brand of database, it is actually referencing a particular database on some server. There is no unsupported operation for this relation. The columns of the relation are as listed above, and each is referencing a Src_Column object, which we will discuss in the following section.

4.7 ICM – Src_Column

The Src_Column class in ICM represents the column of COIN model. Columns are part of a relation. The properties of Src_Column are:

Name	:	string
Type	:	string
keyMember	:	boolean
relation	:	Src_Relation

The *name* property stores the string that represents the name of the column. The name of a column is unique only to the relation it belongs to. Therefore, the fully qualifying identifier for a column is the relation name and the column name. The *type* property is a string indicating the data type of the column. Some common values are "string" and "number", but it could be any type that is understandable by the source. The *keyMember* property indicates whether the column is part of the primary key of the relation. The *relation* property points to the relation that the current column belongs. Below are instances of some of the columns in the dsr_tent_terranova relation.

Name	:	Model
Type	:	string
keyMember	:	True
relation	:	dsr_tent_terranova

Name	:	Usage
Type	:	string
keyMember	:	false
relation	:	dsr_tent_terranova

Name	:	Weight
Type	:	number
keyMember	:	false
relation	:	dsr_tent_terranova

4.8 ICM – Src_ElevatedRelation

The Src_ElevatedRelation class in ICM represents the elevation axioms of COIN model grouped by relation. The properties of Src_ElevatedRelation are:

Name	:	string
relation	:	Src_Relation
context	:	Cxt_Context
columnSemtypes	:	hashtable(Src_Column, Ont_SemanticType)

The *name* property stores the string that represents the name of the elevated relation. The name of an elevated relation is globally unique to the COIN model. The *relation* property points to the relation that is being elevated. The *context* property points to the context which the current elevated relation defines under. Finally, the columnSemtypes is a hashtable that maps columns of a relation to semantic types. The keys of the hashtable are the columns, and the values are the semantic types. Below is an instance of an elevated relation in the tent example.

Name	:	dsr_tent_terranova_p
relation	:	dsr_tent_terranova
context	:	c_uk
columnSemtypes	:	hashtable((Model,basic), (Usage,basic), (Persons,basic), (Weight,weight), (Packed_length,basic), (Packed_width,basic), (Interior_length,basic), (Interior_width,basic), (Interior_height,basic), (Price,basic))

The name of the elevated relation is *dsr_tent_terranova_p*. The relation that it elevates from is *dsr_tent_terranova*. The context *c_uk* is the context under which the elevation applies. All columns in the relation is elevated to basic, with the exception of *Weight*, which is elevated to the semantic type *weight*.

4.9 ICM – Application Programming Interface

There is an Application Programming Interface (API) for interacting with the ICM. The purpose of the API, as with any other APIs, is to allow programmers to work effectively with the ICM, without knowing or relying on the internal representation of the ICM. Internal representation of ICM may change as the design of COIN model changes over time. The API shields such changes from the users. The documentation for the API can be found in appendix A. In the API, there are the usual *Get* and *Set* methods for each user accessible properties of the COIN application. In addition, there are many *Add* and *Remove* methods for adding and removing objects from a list of objects. The API also manages the integrity of the metadata for the users. For example, when an user adds a new semantic type to the COIN application via one of the API methods, the method automatically check if an instance of the semantic type with the specified name already existed. Since semantic type names are unique, the new semantic type cannot be added and an exception is thrown to alert the users. Conversely, when an user deletes a

semantic type from the COIN application via an API method, the method automatically checks for the dependencies other objects in the COIN application have with the semantic type. All objects whose existence depends on the semantic type will be deleted as well. For instance, an attribute can longer exist if one of two semantic types it references is not longer in the application. In a sense, the API is an active enforcer of the COIN modeling rules. By not circumventing the API, programmers can depend on the API to reliably maintain the integrity and consistency of COIN application.

5 Textual Interface

The Textual Interface component of the COIN Application Metadata Layer is a human interface tool that is designed to facilitate the users in development of COIN application. The textual interface is a collection of web forms that accepts and displays application metadata. Mirroring the COIN model, the textual interface is organized into three main forms. They are the Ontology form, Context form, and Source form. Each form handles the relevant section of the application metadata. The Internal COIN Model introduced in the last section is the run-time data structure behind the textual interface. Application metadata retrieved from saved files are first loaded into the ICM. As the user makes changes to an application, the changes are temporarily stored in the ICM until the full content of the ICM is persisted to files. The subsections below explain in details the roles of the various forms in the textual interface.

5.1 Navigating the Textual Interface

The textual interface comprises of a series of forms that maintain different portion of a COIN application. Users can access these forms by interacting with a group of buttons, known as the navigation buttons, which is located at the top of every form. Figure 5.1 shows the navigation buttons.

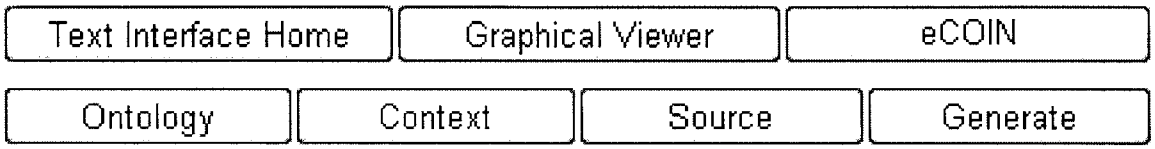


Figure 5.1 – Textual Interface Navigation Buttons

The top row of buttons is for accessing the top level components from the COIN Application Metadata Layer and eCOIN. The button labeled "Text Interface Home" brings the user to the Application Select Form, which is described in section 5.2. The button labeled "Graphical Viewer" brings the user to the Graphical Interface, which is described in Chapter 6. The button labeled "eCOIN" transfers the user to the eCOIN interface and automatically loads the current COIN application for mediation.

The second row of buttons is for accessing components of the textual interface. The button labeled "Ontology" brings the user to the ontology editing page, described in section 5.3. The button labeled "Context" brings the user to the context editing page, described in section 5.5. The button labeled "Source" brings the user to the source editing page, described in section 5.4. The button labeled "Generate" persists the ICM to files and brings the user to the COIN application files viewer, which is described in section 5.6. When creating a new COIN application, the order of traversing the pages of the textual interface is ontology, source, and then context. In the rest of this chapter, the discussion on each page of the interface follows the same ordering.

5.2 Application Selection Form

The Application Selection Form is a web page that allows the user to either select an existing application for editing, or create a new application. A screenshot of the form is

shown in Figure 5.2. There are a couple of places on this form that require user inputs. The first place where an input is needed is in the URL of this page. Lets take the sample

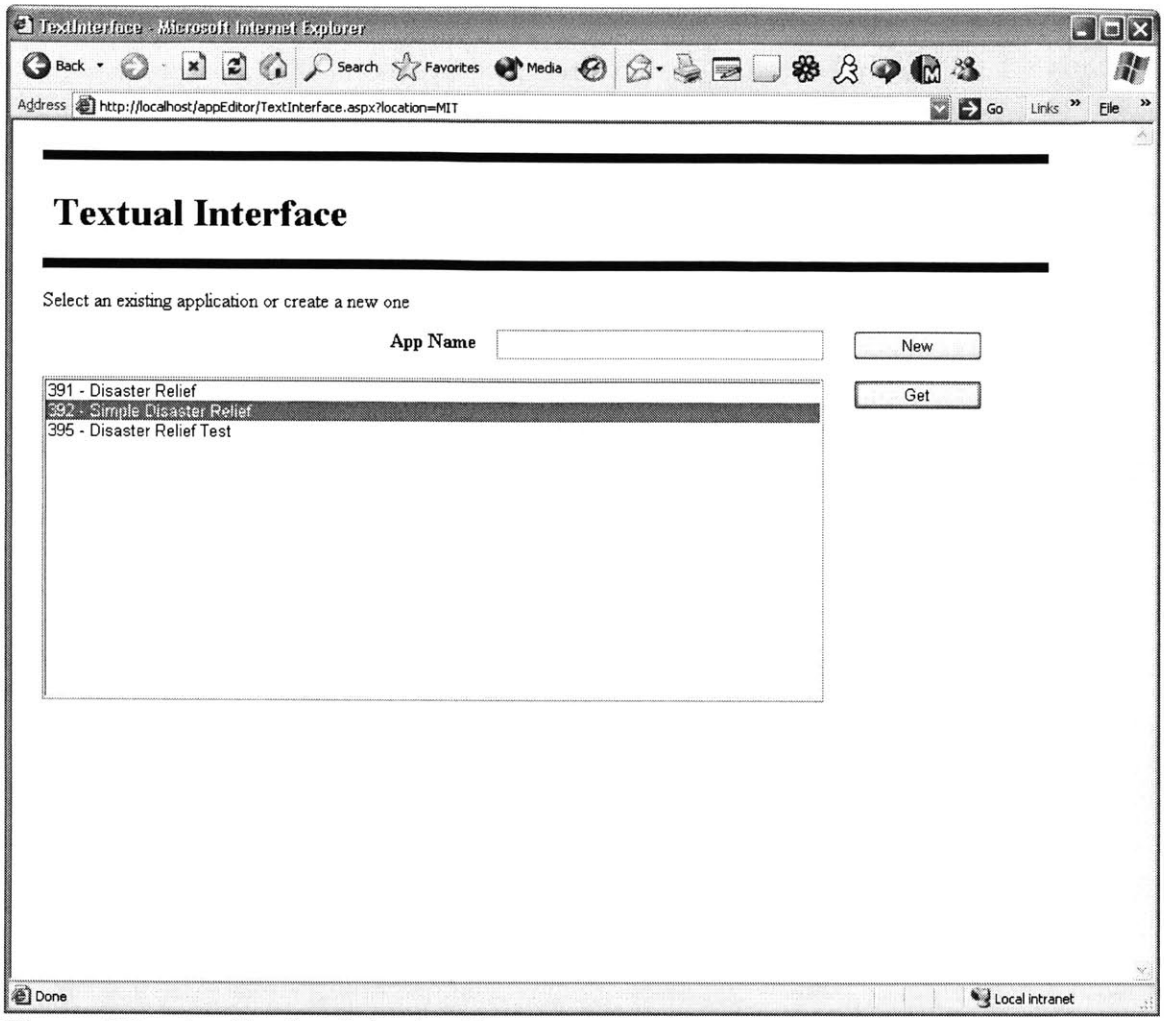


Figure 5.2 – Application Selection Form

URL *"http://localhost/appEditor/TextInterface.aspx?location=MIT"* for illustration purposes. The hostname *localhost* in the URL indicates the address of the server hosting the textual interface. The next item in the URL path is *appEditor*, which indicates the name of the project as it is known in the code. The next item is *TextInterface.aspx*, which is the name of the actual web resource. The extension *.aspx* indicates that it is a web form created using the Microsoft .Net Framework [20]. The *?* symbol that follows indicates the start of request parameters. The only parameter the user needs to specify in the URL is *location*. The *location* parameter tells the textual interface to look up the application registry belonging to the specified location, *MIT* in this case. The COIN registry is described in a later section, but for the present it is sufficient to say that the registry is grouped by location. In effect the sample URL would cause the textual interface to find all available applications from the COIN registry defined under the *MIT* location. The available applications are listed in the list box on the form. The user can then select an application and press the *Get* button to start editing. To create a new

application, the user can provide a name for the application in the text box labeled "App Name" and press the *New* button to start editing the application. An application ID is automatically generated for the new application. The ID generated is usually the next highest number in the assigned sequence. Spaces are allowed in the name of an application and there is currently no limitation on the length of the name. It is up to the user to supply concise and meaningful name for the application. The new application is also automatically registered in the COIN registry under the location specified in the URL. Whether the user is editing an existing or new application, the user will be taken to the ontology form, which is described in the next section. Besides the URL, the interface is also accessible from within eCOIN. This is described in details in Chapter 9 on technology merging.

5.3 Ontology Form

The Ontology Form is a web page for editing the ontology of a COIN application. An ontology consists of semantic types, and the relationships between them, such as inheritance, attribute, and modifier. The form is separated into three sections: Semantic Types, Inheritances, and Attributes. Since the information needed to declare modifiers on the ontology level is similar enough to that of attributes, the mechanism for declaring attributes and modifiers is lumped together in the Attributes section. Lets explore each section in details.

Figure 5.3 is a screen shot of the ontology form. At the top of the form are the navigation buttons for navigating to different sections of the textual interface, or to other interfaces. Also at the top are the ID and name of the application being edited. The oversized numbers on the screenshot are painted on for illustration purposes. They are not displayed on the actual web page. The area labeled **1** is the Semantic Types section. There is a text box for entering the name of new semantic types, and a list box for displaying the names of the semantic types already in the ontology. There are two buttons, one labeled "Add" and the other "Delete". The Add button adds to the ontology the semantic type whose name appears in the text box. The Delete button removes from the ontology the semantic type whose name is selected in the list box. Removing a semantic type from the COIN application will trigger all metadata related to the semantic type to be removed. This is a feature in the ICM that ensures the integrity of the metadata in the COIN application.

Through out the textual interface there are similar Add and Delete buttons that require the user to either type in some text, or select some text, prior to clicking on them. The text required or displayed depends on the section of the textual interface the user is on. In the semantic type section, the text displayed are semantic type names. In the context section, the text could be context names. Due to the analogous nature of the buttons appearing on different places in the textual interface, their functions will not be repeated to the readers in subsequent description of the textual interface. The readers can conclude that the Add button, wherever it appears, is for adding information to the COIN application; and the Delete button is for removing information from the COIN application. Once the user has click on one of these buttons, a HTTP request is sent back to the server, where by the appropriate changes are made to the COIN application loaded in the ICM, and the

updated information is sent back to the client browser via the same web page that initiated the changes.

The area labeled **2** in Figure 5.3 is the Inheritances section. This is the section where inheritances among semantic types are declared. In every inheritance relationship is a

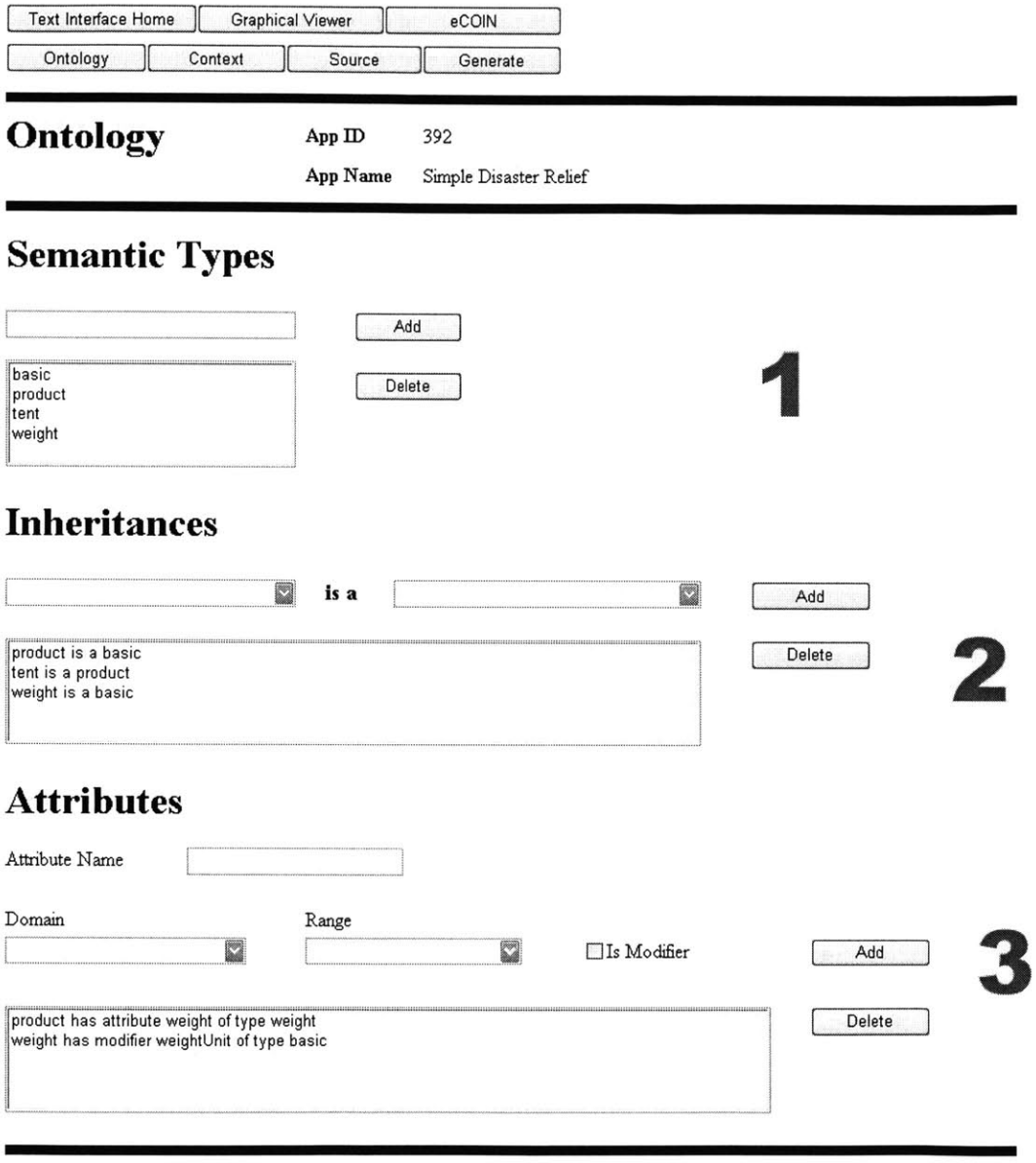


Figure 5.3 – Ontology Form

child and a parent. The phrase "*child* is a *parent*" describes this relationship exactly. The dropdown box located to the left of the "is a" label contains a list of possible child semantic types. The dropdown to the right is the list of possible parent semantic types. Once both child and parent semantic types are selected, the user can add the new inheritance relationship to the COIN application. The new inheritance will appear in the list box as the phrase "child is a parent", whereby the child is the name of the child

semantic type and the parent is the name of the parent semantic type. Each phrase, or inheritance relationship, in the list box can be selected for removal.

The area labeled **3** in Figure 5.3 is the Attributes section. This is the section where attribute and modifier relationships are declared. Recalling from the section on COIN model, in every attribute and modifier relationship is a domain and a range. The domain is the semantic type that owns the relationship; the relationship is a property of the domain. For an attribute, the range is the semantic type that represents the type of the attribute value. For a modifier, the range is the semantic type that the domain gets modified to. The phrase "*domain* has attribute/modifier *name* of type *range*" captures the essence of such relationship. Every attribute and modifier relationship in a COIN application is displayed in the list box as such a phrase. Once an attribute/modifier name, a domain and a range are specified, the user can add the relationship to the COIN model. The checkbox labeled "Is Modifier" denotes whether the relationship is an attribute or a modifier. The user can also select a phrase, or relationship, from the list box for removal.

The user can leave that ontology section at any time by using one of the navigation buttons at the top. The ontological information from the tent example is shown in the figure. We will discuss the source form next.

5.4 Source Form

The Source Form is a web page for editing the relations and their columns. The form consists of a section for declaring relation, and a section for declaring each column in a relation. Figure 5.4 shows a screenshot of the source form. Lets discuss the form section by section.

The area labeled **1** is the relation section where relations are declared. The properties that define a relation are its name, the name of its source, its type, and its unsupported operations. The name of the relation is entered through the text box provided in the form. The type of the relation, which is one of *import*, *export*, or *both*, is selected from a radio button group. The unsupported operations, which are <, >, =, <=, >=, are selected from a group of checkboxes that supports multiple selections. The name of the source is selected from a dropdown box that contains a list of all available sources. The list of available sources is obtained from the COIN registry. Multiple COIN application can share the same sources. Once these properties of a relation are specified, the user can add the relation to the COIN application. The phrase "*relation* of *source* is of *type* and has *unsupported operations*" expresses a relation and its properties. Every relation in a COIN application is expressed as such a phrase and displayed in the list box on the form. The user can select on a phrase, or relation, from the list box to remove it from the application.

The area labeled **2** is the column section where columns of any relation are declared. The properties that define a column are its name, the name of the relation it belongs to, its type, and its key property. The name of a column can be entered through the text box in the form. The name of the relation must be selected from a dropdown box that contains a list of the name of all existing relations. The type of the column, which is one of string,

integer, or real, is selected from the dropdown box provided. The user can toggle the checkbox labeled "Member of Primary Key" on or off to denote whether a column is a member of the primary key for the relation it belongs to. Once these properties of a column are supplied, the user can add the column onto the specified relation. The phrase "*relation* has *column* of *type* and is/is-not part of key" describes a column. Every column of a relation is expressed as such a phrase and displayed in the list box on the form. Upon arriving at this web page initially, the columns from all relations are displayed in the list box. There is a *Get* button on the form for displaying just the columns from a relation the user has selected.

The relation and column metadata is shown in Figure 5.4. Once the source information is defined, the user can proceed to define the elevations for these relations in the Context form that is described next.

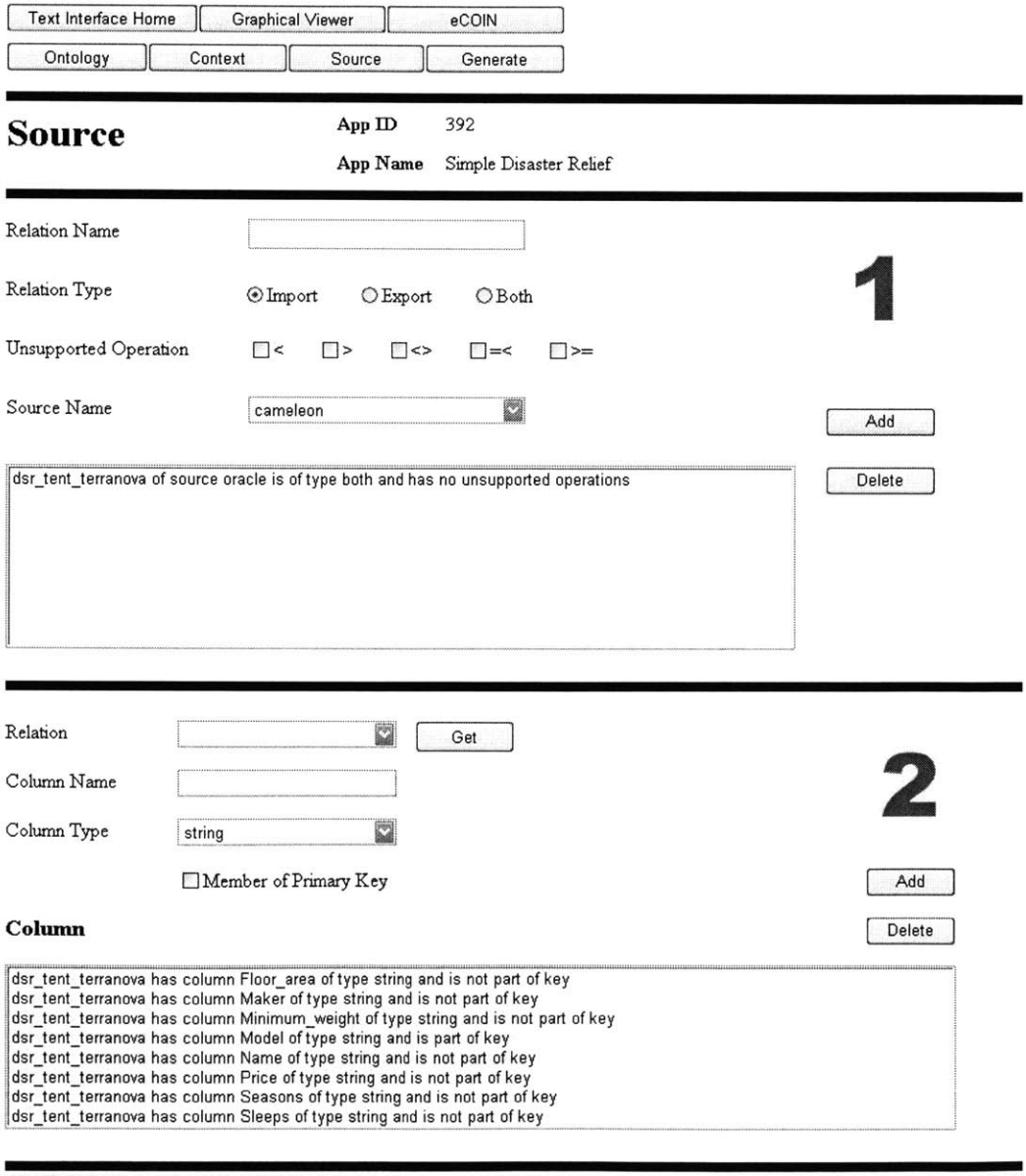


Figure 5.4 – Source Form

5.5 Context Form

The Context Form is a web page for editing the context information of a COIN application. The context form consists of three sections. There is a section for declaring contexts, a section for declaring the elevation of relations, and a section for defining modifier values. The context form is shown in Figure 5.5. Lets describe this form section by section.

The area labeled **1** in the form is the context section. This is the section where contexts are declared. The name of the contexts is entered through the text box on the form. There is a dropdown box, which lists all existing contexts, for declaring context sharing. When declaring a new context, the user can select an existing context from the dropdown box for context sharing. The phrase "*context* is a *parent context*" describes context sharing, where *context* is the name of the new context and the *parent context* is the name of the context that is being shared by the new context. Every context that has context sharing defined is displayed as such a phrase in the list box on the form. If no context sharing is defined, just the name of the context is displayed in the list box. The user can select a phrase, or context, from the list box for removal from the COIN application.

The area labeled **2** is the relation elevation section. This is the section where elevation information is declared and displayed. Initially, the form shows the elevation for every relation in the COIN application in the list box labeled "Relation Elevation". The user can restrict which relation elevations are displayed by using the two *Get* buttons on the form. The *Get* button next to the context dropdown box retrieves the elevations for all relations that have elevations defined under the selected context. Once a context is selected from the dropdown box and the user has clicked on the *Get* button, a list of relations that have elevations defined under the selected context is displayed in the list box labeled "Relation". At that point, the user can further restrict which relation elevation is displayed by selecting one of the relation and clicking on the *Get* button next to the relation list box.

The process of declaring new elevation is a two steps process. First, a relation and a context must be specified; then the elevation of each column can be added. Once the relation is selected on the relation dropdown box and the context is selected on the context dropdown box, the user can click on the "Add Relation Context" button to start elevating the columns of the relation. A list of all columns from the selected relation are listed in the list box labeled "Column", and a list of all available semantic types are listed in the list box labeled "Semantic Type". To elevate a column, the user simply selects from the list boxes a column to be elevated, a semantic type to elevate the column to, and click on the *Add Elevation* button. The phrase "*relation* under *context* has *column* and is elevated to *semantic type*" describes an elevation axiom for a column. Each elevated column appears as such a phrase in the list box labeled "Relation Elevation" on the form. To remove an elevation, the user can select a phrase, or elevation axiom, from the list box and click on the *Delete Elevation* button.

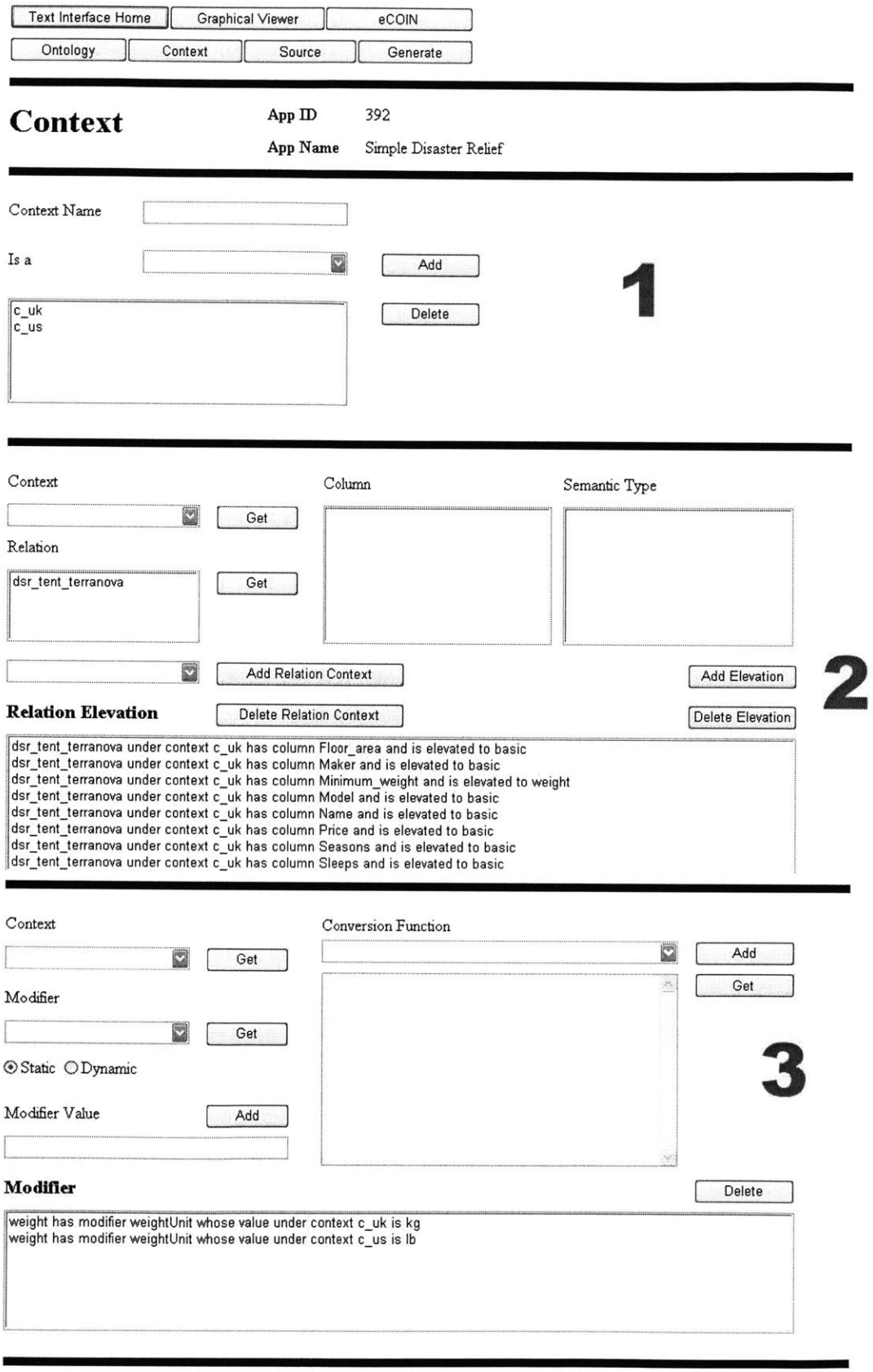


Figure 5.5 – Context Form

The area labeled **3** is the modifier section. This is the section where modifier values are defined per context, and conversion functions are defined per modifier. Initially, the form shows the modifier values for every modifier under every context in the lists box labeled "Modifier". The user can restrict which modifier values are displayed by using the two *Get* buttons on the form. The *Get* button next to the context dropdown box retrieves modifier values declared under the selected context across all modifiers. The user can further restrict the modifier values displayed to just those belonging to a single modifier by selecting a modifier from the list box labeled "Modifier", click on its *Get* button. If no context were selected from the context dropdown box when the Get button was pressed, the modifier values for the selected modifier across all contexts will be displayed.

To define a modifier value for a modifier, a context name and a modifier name must first be selected from the dropdown boxes. Then the user must select either *Static* or *Dynamic* from the radio button group. By default, the *Static* button is selected. Recalling from the section on COIN model, a static modifier value is a string or number value that stays constant; whereas a dynamic value is a list of attribute names that describes how to get to a value at run time. The modifier value is entered through the text box labeled "Modifier Value". To enter a static value, simply select the *Static* button and type in the value in the text box. To enter a dynamic value, select the Dynamic button and type in the names of the attributes, using the separator symbol "->" between the names. The names of the attributes should be in the order that links them from the modified semantic type to the destination type. This would avoid potential ordering error in situations where multiple paths exist between semantic types. Entering them in order, starting from the attribute closest to the modified type, would also visually help other users to discern the dynamics of the modifier value. The phrase "*semantic type* has *modifier* whose value under *context* is *modifier value*" describes a modifier value for a modifier under one context. Every modifier value defined in a COIN application is expressed as such a phrase and displayed in the list box labeled "Modifier". To remove a modifier value from a COIN application, simply select a phrase, or modifier value, from the list box and click the *Delete* button.

Conversion functions for modifiers are declared in this section as well. Once a modifier is selected from the *Get* button for modifier, the conversion function for the modifier, if defined, is displayed in the multi-line text box labeled "Conversion Function". There are two ways to declare conversion functions for modifiers. One way is through the dropdown box, and the other way is through the multi-line text box. The conversion function dropdown box contains the names of all pre-defined functions taken from the function library from the COIN registry. The function library contains some of the most common conversion function, in generic form, that is non-specific to any COIN application. To find out the content of a conversion function, the user can select the name of a function from the dropdown box and click on the *Get* button for conversion function. The content of the pre-defined conversion function will be displayed in the multi-line text box. The other way to define a conversion function is to write out the conversion function in Prolog in the multi-line text box provided. Notice that often times conversion functions invoke other helper functions that must be defined as well. Recalling from the section on Prolog that each Prolog statement in a COIN application is

a rule, the user can enter in the text box as many rules as necessary to fully support a conversion function. There must be at least one rule labeled that uniquely defines the conversion function. Once a conversion function is defined using either the dropdown box or the text box, the user can click the *Add* button to add it to the COIN application. To remove a conversion function from a modifier, the user can select a modifier from the modifier dropdown box and click on the *Delete* button for conversion function.

5.6 Application File Viewer

The Application File Viewer is a web page for displaying the metadata of COIN application. A number of XML-based representations of COIN application are introduced by this thesis. Each representation is file-based, and this file viewer provides a way to display the content of the files in a consolidated fashion for easy comparison. This page is accessible from the navigation button labeled "Generate". Prior to loading

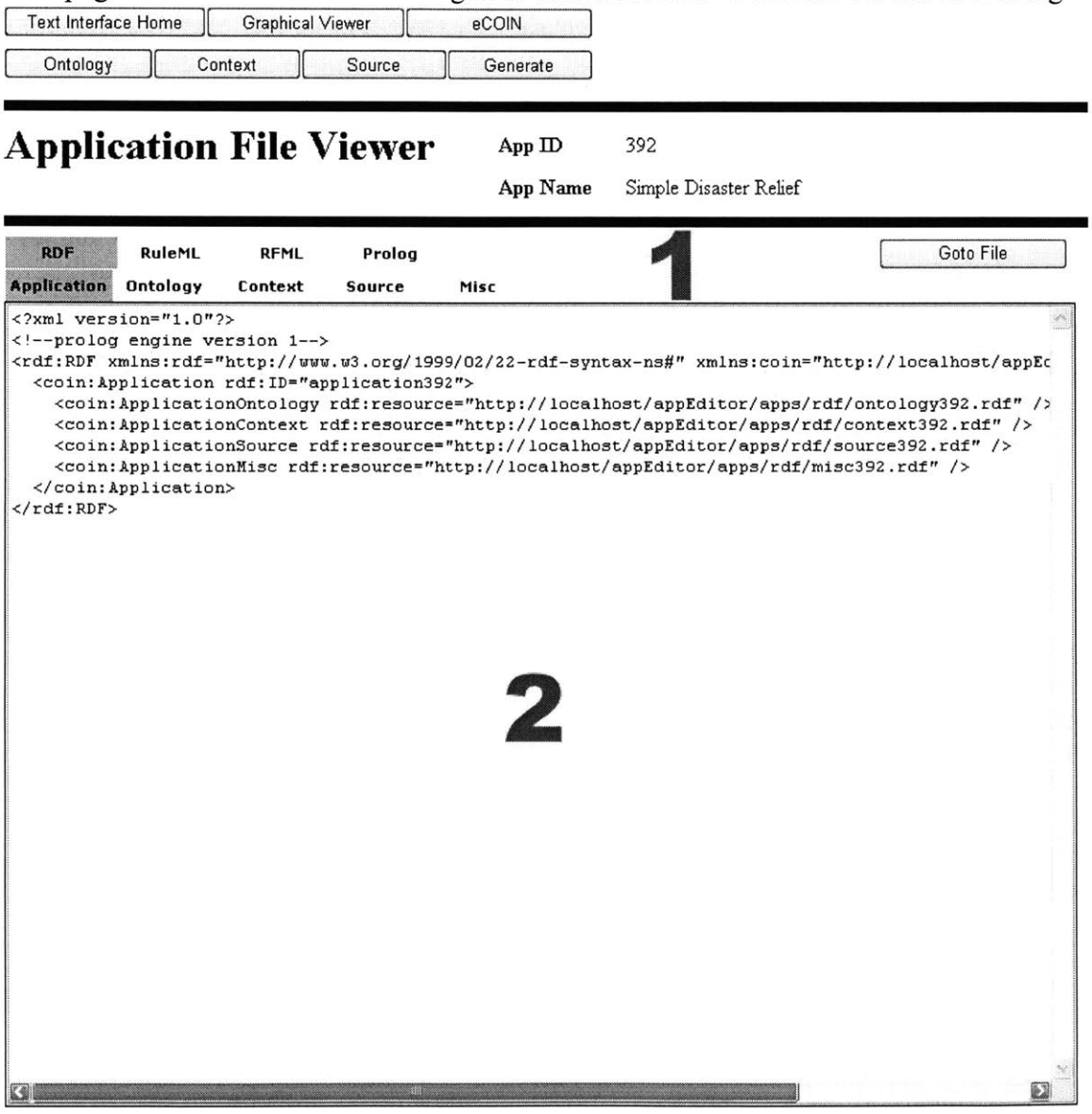


Figure 5.6 – Application File Viewer

the page, the Generator of the COIN Application Metadata Layer is invoked to persist the metadata in ICM into files. The Generator is discussed in details in a later section. Lets describe each section of the file viewer, which is shown in Figure 5.6.

The area labeled **1** on the page is the file selection panel. There are a number of navigation tabs for locating the different files. The four tabs on the top row are for accessing the four different representations (RDF, RuleML, RFML, and Prolog) of a COIN application. By clicking on the RDF tab, the bottom row of tabs expands into five tabs, which are labeled "Application", "Ontology", "Context", "Source", and "Misc". Since the RDF representation spans multiple files, the bottom row of tabs is for navigating each of the five files. The text area labeled **2** shows the file currently selected from the navigation tabs. Clicking on the RuleML tab brings to view the file that contains the COIN application in RuleML representation. There is no tab on the bottom row for the RuleML tab because the application is contained in a single file in this format. Similarly, clicking on the RFML tab brings to view the RFML file and the bottom tabs disappear. Finally, clicking on the Prolog tab reveals two tabs on the bottom row. The two tabs are labeled "HTML" and "Non-HTML". The tab labeled "HTML" brings to view the HTML version of the application Prolog file. The tab labeled "Non-HTML" brings to view the actual Prolog application file that will be used by the abduction engine in eCOIN. Additionally, there is a button labeled "Goto File" in the area labeled **1** on the page. This button brings the web browser to the actual location of the selected file. If the user is using Microsoft Internet Explorer, the XML-based application file being displayed can benefit from the browser's XML tree navigating capability, which allows the user to collapse and expand on each element tag.

6 Graphical Interface

The Graphical Interface of the COIN Application Metadata Layer is a tool designed to facilitate the human users in visualizing a COIN application. Since most of the information in a COIN application is ontologically oriented, it presents well in a graphical format. The graphical interface makes use of this feature by allowing the users to incrementally display information around an ontology. There are two instruments of display in the graphical interface. One is a tree structure that organizes the application metadata into a hierarchy of tree nodes. The other is an image panel where metadata are automatically drawn into images and generated on the screen. Since auto-generated images of the ontology often have the problem of overlapping, or unnecessary crossing of lines and shapes, the graphical interface also provides a mean for user to specify the positions and sizes of objects drawn on the screen. A snapshot of the graphical interface, displaying the tent application, is shown in Figure 6.1. The following sections examine each part of the graphical interface in details.

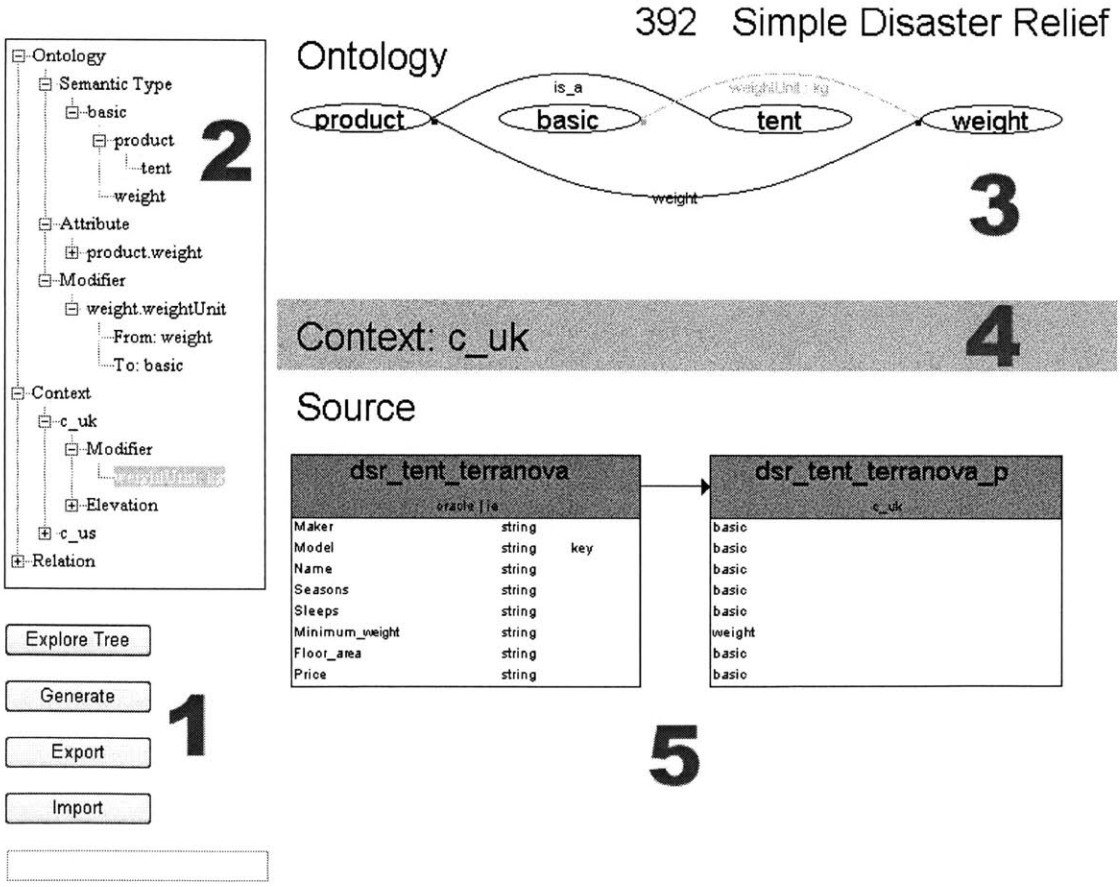


Figure 6.1 – Graphical interface displaying the tent example

6.1 Tree Navigation Structure

The area labeled **2** in Figure 6.2 is the tree control structure that allows the user to navigate an entire COIN application. By navigation, we mean the ability to focus on particular section of the application by expanding certain nodes and collapsing others.

The navigation tree serves the dual purposes of displaying application metadata, and providing control over the content in the image panel. The displaying part of the navigation will be described first, followed by the control part.

In designing the interface, the layout of the screen components is optimized for information density. Given a limited screen space, it is best to give the most space to the component that carries the most information across to the user. Naturally, the most amount of screen space is dedicated for the image panel where the ontology is graphically displayed. This leaves the navigation tree with a relatively small amount of space. This means the user can access from the tree structure only a narrow window of information at a time. To compensate for this, a separate page is dedicated entirely for displaying the tree structure, where more tree nodes can be explored at a time. The separate page is accessed from the button labeled "Explore Tree", located in the area labeled **1**. Below is a snapshot of the entire application tree found under the tree exploration page, with all nodes expanded.

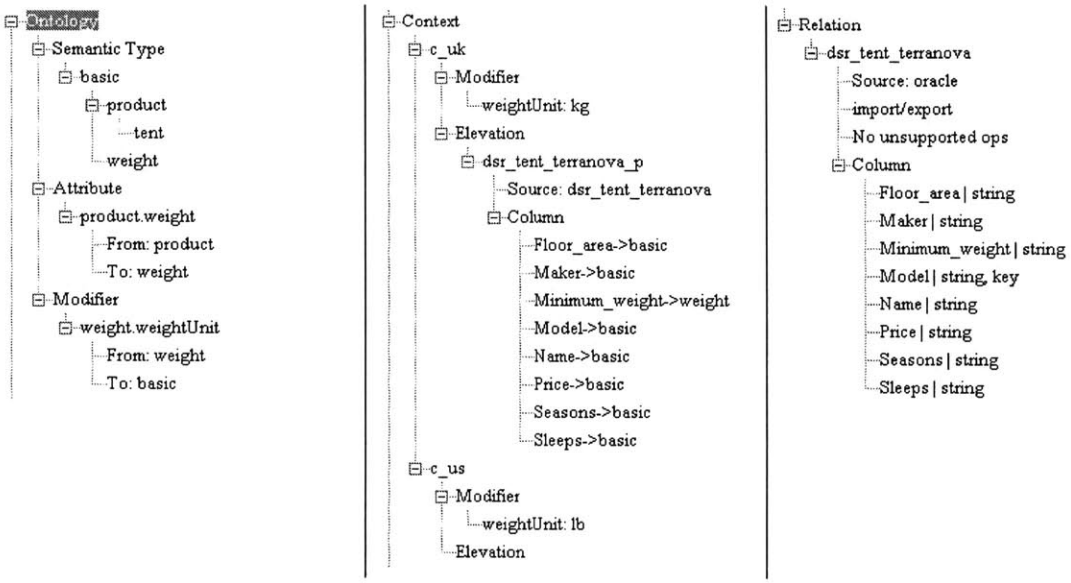


Figure 6.2 – Navigation Tree Expanded (Tree is coiled around from left to right for illustration purpose)

The left most section of Figure 6.2 shows the ontology of our tent application. The node labeled "Ontology" contains three child nodes, labeled "Semantic Type", "Attribute", and "Modifier". The *Semantic Type* node contains a group of nodes whose labels are the names of the semantic types in the ontology. Moreover, the nodes are arranged such that the depths of the nodes represent the depth of the inheritance relationships found in the ontology. The node representing the semantic type *basic* is, as expected, the root node of the semantic type tree. The sibling nodes, *product* and *weight*, are both immediate children of the *basic* node. This means that both *product* and *weight* are one inheritance relationship from the semantic type *basic*. The *tent* node has a node depth of two, which means it is two inheritance relationships from the semantic type *basic*. Since *tent* is the immediate child node of *product*, the semantic type *tent* inherits directly from the semantic type *product*.

The second child node of the *Ontology* node is the *Attribute* node. The *Attribute* node contains a node for each attribute relationship in the ontology. Each attribute node is labeled under the format "*semanticType_name.attribute_name*", where *semanticType_name* is the name of the originating semantic type, and *attribute_name* is the name of the attribute. Notice the dot between semantic type name and the attribute name. It is necessary to specify the fully identifying name of each attribute because multiple attributes could use the same name, as long as they originate from different semantic type. Within each attribute node is a node describing the originating semantic type, and a node describing the destination semantic type. The format of the labels of these two nodes respectively are "From: *sematnicType_name*" and "To: *semanticType_name*", where *semanticType_name* is the name of the corresponding semantic type.

The third child node of the *Ontology* node is the *Modifier* node. The *Modifier* node contains a node for each modifier relationship in the ontology. The modifier nodes are very similar to the attribute nodes. Each modifier node is labeled under the format "*semanticType_name.modifier_name*", where *semanticType_name* is the name of the originating semantic type, and *modifier_name* is the name of the modifier. Notice the dot between semantic type name and the modifier name. It is necessary to specify the fully identifying name of each modifier because multiple modifiers could use the same name, as long as they originate from different semantic type. Within each attribute node is a node describing the originating semantic type, and a node describing the destination semantic type. The format of the labels of these two nodes respectively are "From: *sematnicType_name*" and "To: *semanticType_name*", where *semanticType_name* is the name of the corresponding semantic type.

The middle section of the tree in Figure 6.2 shows the contexts of the tent application. The node labeled "Context" contains as many child nodes as there are contexts in a COIN application. There are two contexts in the tent application, so two child nodes are found under the *Context* node. The two context nodes are labeled *c_uk* and *c_us*, which corresponds to the names of the contexts as defined in the metadata. Within each context node is a *Modifier* node and an *Elevation* node. The *Modifier* node contains a node for each modifier whose value is defined under the current context. The label of each modifier node is "*modifier_name*: *modifier_value*", where *modifier_name* represents the name of the modifier, and *modifier_value* represents its value. If the modifier value is static, then the value is simply a string value. For dynamic modifier, the value is a list of attribute names separated by the symbol "->". In the case of the tent example, the name of the modifier is *weightUnit*; its value is *kg* under the *c_uk* context, and *lb* under the c_us context.

The other node within a context node besides the *Modifier* node is the *Elevation* node. The *Elevation* node contains a node for each relation elevated under the current context. The label of each elevation node is the name of the elevated relation. Since *dsr_tent_terranova_p* is only one elevated relation in the *c_uk* context, there is only one elevation node found under this context. Within each elevation node is a source node

whose label is "Source: *source_name*", where *source_name* is the name of the underlying relation of the elevated relation. *Column* node is the other node within the elevation node. The *Column* node contains a node for each column of the relation and the semantic type the column is elevated to. The label of each column node is "*column_name->semanticType_name*". For instance, the column *Minimum_weight* in the relation *dsr_tent_terranova* is elevated to the semantic type *weight*.

The right most section of the tree in Figure 6.2 shows the relations of the tent application. The node labeled "Relation" contains as many child nodes as there are relations in the application. There is one relation, named *dsr_tent_terranova*, in the application. The label of each relation node is the name of the relation. The first node under the relation node is the source name of the relation. The label of this node is "Source: *source_name*". the second node under the relation node is the import/export property of the relation. The label of this node reflects the value of this property. The third node under the relation node denotes the unsupported operations of the relation. There is no unsupported operation for the relation in the example. If there were, the label of the node would have the format "<,>,<>,<=,=>", where the unsupported operations are separated by commas. Finally, the last node under the relation node is the *Column* node. The *Column* node contains as many nodes as there are columns in the relation. Each column node label has the format "*column_name* | *column_type*, key", where *column_name* is the name of the column, and *column_type* is the type of the column. The keyword "key" is present in the label if the column is part of the primary key of the relation. For instance, the column node representing the *Model* column of the *dsr_tent_terranova* relation is labeled "Model | string, key".

The reader should be convinced by now that the navigation tree maintains a complete description of a COIN application in a structured and easy to navigate format. Moreover, since the navigation tree is browser-rendered object, the tree can be printed via the print function of the internet browser. As mentioned earlier, the navigation tree also serves the purpose of providing user the control over what metadata to display in the image panel. Before this function of the navigation tree can be discussed, lets examine the image panel first in the next section.

6.2 Graphical Image Panel

The Graphical Image Panel displays an image of the graphical representation of the application metadata. The areas labeled 3, 4 and 5 in Figure 6.1 belong to the image panel. Users can press the *Generate* button to refresh the content of the image panel. The metadata is shown incrementally around the ontology of the application. At the ontology layer, the semantic types, attributes, and modifiers are drawn out in an ontology graph with appropriately labeled nodes and edges. At the source layer, the relations and their corresponding columns are displayed in a tabular format. Finally, at the context layer, contextual information such as modifier values are added to the ontology, and elevations of relations are revealed. Each layer of the image panel is described in details in the following sections.

6.2.1 Ontology layer

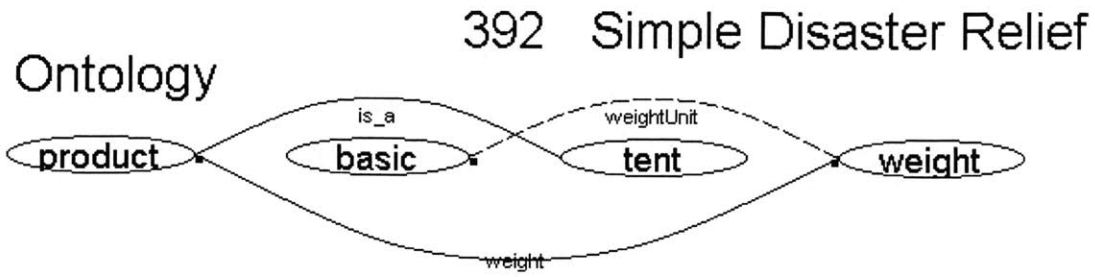


Figure 6.3 – Ontology Layer of Graphical Image Panel

The ontology layer of the graphical image panel is the ontology of a COIN application, drawn out as an ontological graph. The nodes of the graph are semantic types, and the edges are one of inheritances, attributes or modifiers. Semantic types are represented as oval-shaped objects in Figure 6.3. The names of the semantic types are shown in the center of the ovals. The attributes are represented as solid, curved lines, connecting two semantic types. The curve originates from a semantic type that is the domain of the attribute, and terminates at a semantic type that is the range of the attribute. The terminating end of the attribute curve is denoted by a closed circle. The labels of the attributes are the names of the attributes. Modifiers are drawn similarly to attributes, except that modifiers are represented by dashed curves. The inheritance relationships are also drawn similarly to the attributes, except that the label on the solid curve is always "is_a" to remind the viewer of the phrase "child semantic type *is a* parent semantic type". Figure 6.3 illustrates the ontology layer. Notice that two edges, an inheritance edge and a modifier edge, in figure have their lines crossed. This is the direct result of automatically generated graph, and it serves to emphasize the importance of the layout adjustment feature of the graphical interface. This feature, which will be discussed in details in a later section, allows objects on the graph to be re-positioned by the user.

6.2.2 Source Layer

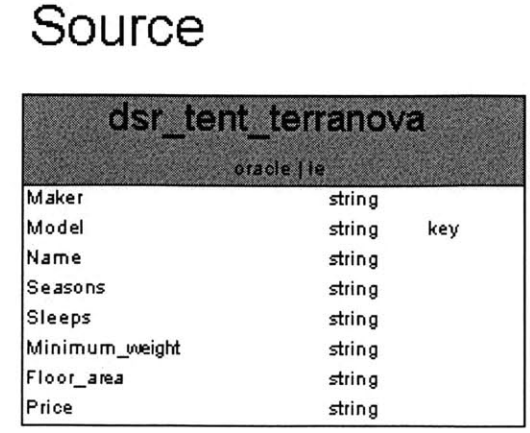


Figure 6.4 – Source Layer of Graphical Image Panel

The source layer of the graphical image panel displays the relations of a COIN application. Figure 6.4 shows a relation from our tent example. The relations are drawn out in tabular format to emphasize their relational nature. Each relation is represented by a box with a title and a body. The title area is shaded in blue. It contains the name of the relation in bold fonts, and the properties, namely source and import/export, in regular fonts. The body area displays the columns of a relation in rows. Each row starts with the name of the column, followed by the column type, and sometimes the keyword *key*, when the column is part of the primary key.

6.2.3 Context Layer

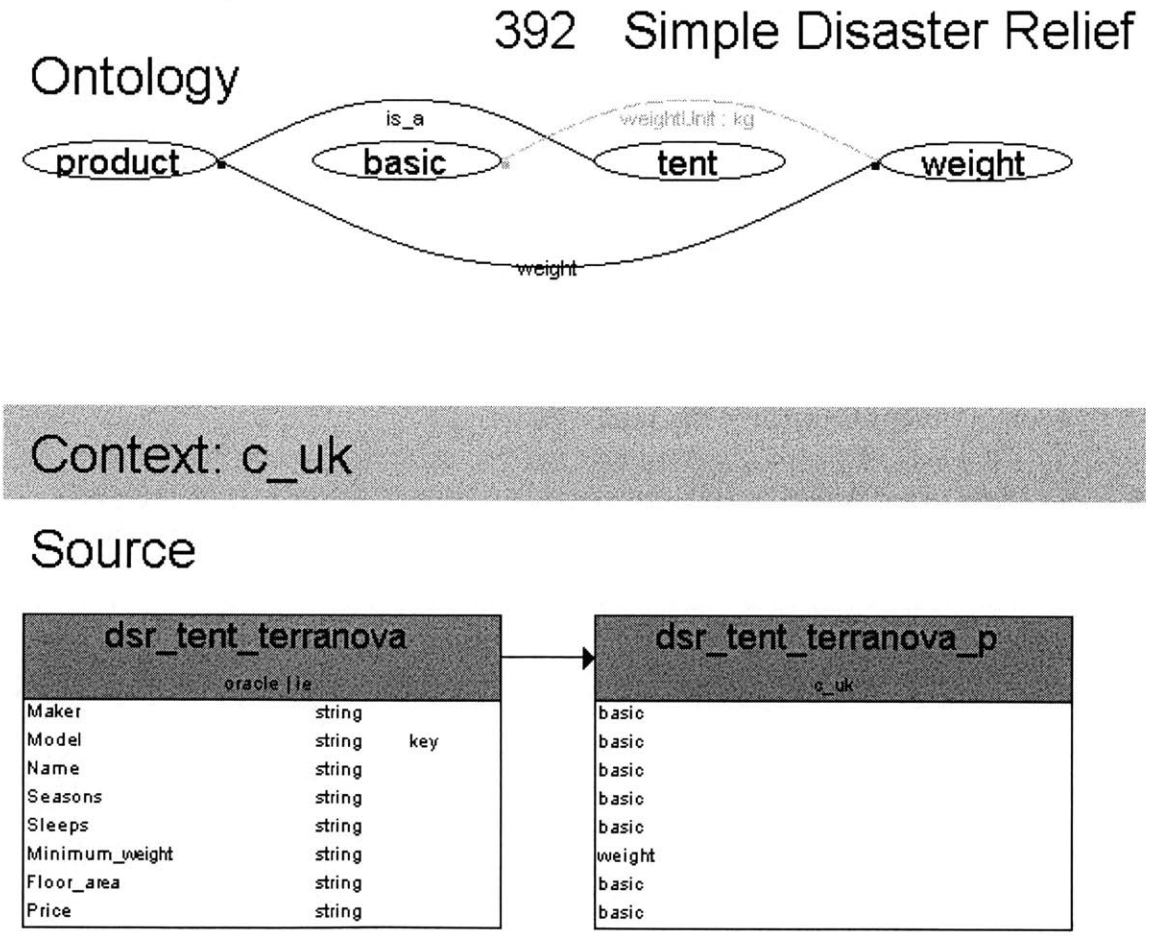


Figure 6.5 – Context Layer of Graphical Image Panel

The context layer of the graphical image panel applies contextual information to the ontology layer and source layer, thereby dynamically altering the image for each context in a COIN application. Figure 6.5 shows a complete image panel with basically the same ontology and source layers introduced in earlier sections. New to the image is the contextual information for the *c_uk* context. The band between the ontology and modifier layers displays the name of the context the image is currently drawn for. The ontology layer is where the modifier values, which are context based, are introduced. Notice the modifier *weightUnit*, being highlighted by the orange dashed line, in the ontology from the figure. In addition to the modifier name, the modifier value *kg* is

shown next to the name of the modifier. This information added by the context layer helps to elucidate the concept that the value held by the semantic type *weight* is associated with the physical unit kg under the *c_uk* context. Another way of interpreting this information is that the value held by the semantic type *weight* will be modified when presented to a context where the weight unit is not kg.

The source layer is where the elevations of relations, which are context based, are introduced. The elevation is graphically displayed as a tabular box that is placed to the right of the relation being elevated. The box is similar to the box used to represent the original relation. An arrow is drawn from the original relation to the elevated relation to denote the relationship. The title area of the elevation box contains the name of the elevated relation in bold, as well as the context of elevation. The body area of the elevation box has rows showing the elevated semantic types of the columns from the relation. Notice that each column from the relation in the left is lined up to the right with the semantic type it is elevated to. This tabular format chosen allows the viewer to examine the information on a relation, as well as its elevation, at a glance.

It is important for the reader to keep in mind that the context layer drawn on the image panel changes according to the context selected. The selection of context happens in the navigation tree. This selection process will be described in the next section.

6.3 Dynamic Content Control of the Image Panel

The graphical image panel combines with the navigation tree to provide the user control over the display of dynamic contents. There are two types of dynamic contents in the graphical representation of a COIN application. One type of dynamic content is the information that defines a context. Context can be thought of as a filter that alters which modifier values and elevations are shown in the generated image. The user decides which context filter is applied by selecting a context from the navigation tree. The other type of dynamic content is the highlighting that is painted on the image. The highlighting is a visual aid for the user to discern an object of interest among other objects in a COIN application. The user controls which object is highlighted by selecting the corresponding object in the navigation tree. The different dynamically generated contents are discussed in the following subsections. It is important to point out to the reader that the benefit of highlighting certain objects in the image may not be apparent in the relatively simple tent example. However, in a larger application, the ability to find a semantic type among numerous other types becomes crucial for the user.

6.3.1 Highlighting of Selection

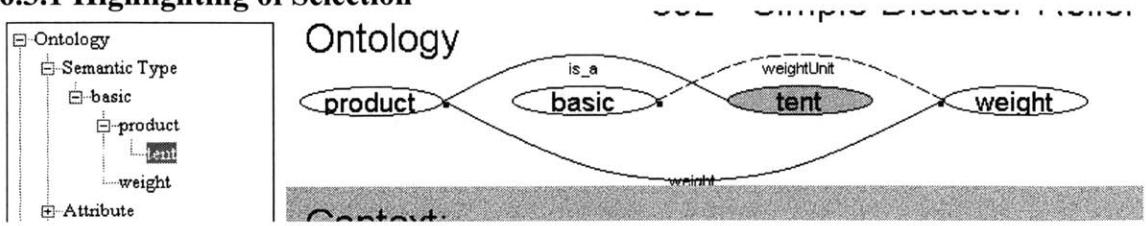


Figure 6.6 – Semantic Type Selection

Figure 6.6 illustrates the selection of the *tent* semantic type. The selection is made in the navigation tree by clicking on the name of a semantic type. Once clicked, the oval corresponding to the selected semantic type is highlighted with the color orange.

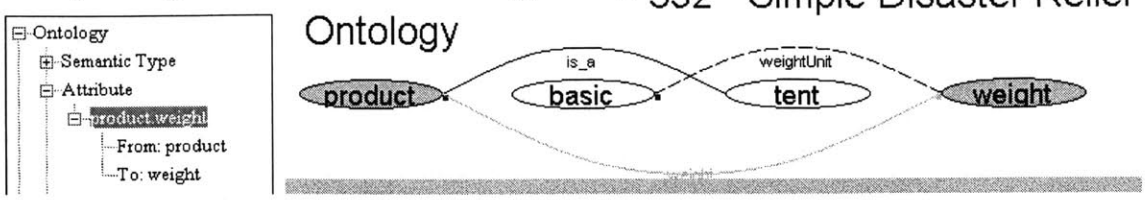


Figure 6.7 – Attribute Selection

Figure 6.7 illustrates the selection of the *weight* attribute. The selection is made in the navigation tree by clicking on the name of an attribute. Once clicked, the line representing the selected attribute is highlighted, as well as the two ovals corresponding to the *From* and *To* semantic types of the attribute.

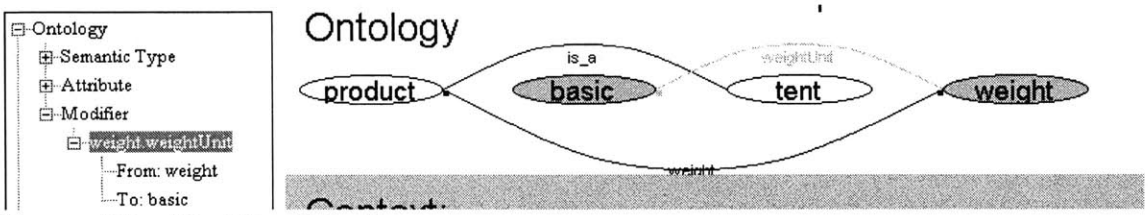


Figure 6.8 – Modifier Selection

Figure 6.8 illustrates the selection of the *weightUnit* modifier. The selection is made in the navigation tree by clicking on the name of a modifier. Once clicked, the line representing the selected modifier is highlighted, as well as the two ovals corresponding to the *From* and *To* semantic types of the modifier.

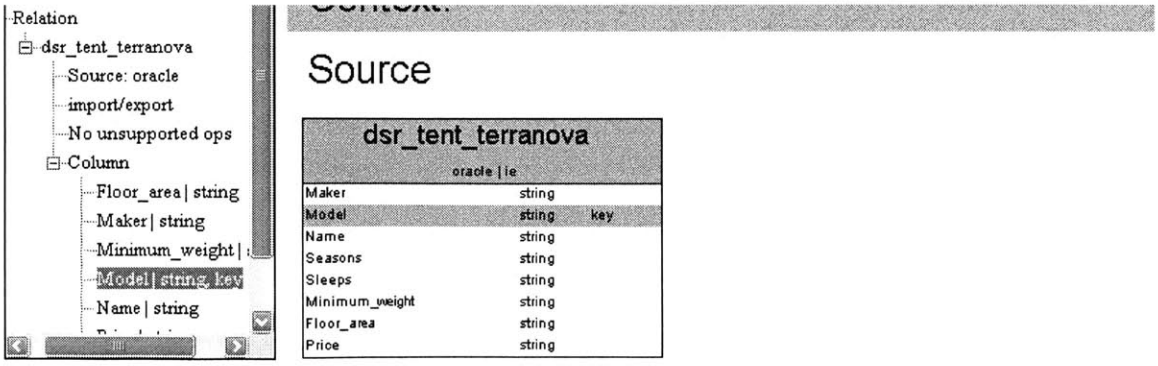


Figure 6.9 – Relation and Column Selection

Figure 6.9 illustrates the selection of the *dsr_tent_terranova* relation. The selection is made in the navigation tree by clicking on the name of a relation, or any of its columns. Once clicked, the title box representing the relation becomes highlighted, as well as the row that represents any selected column.

6.3.2 Context Filter

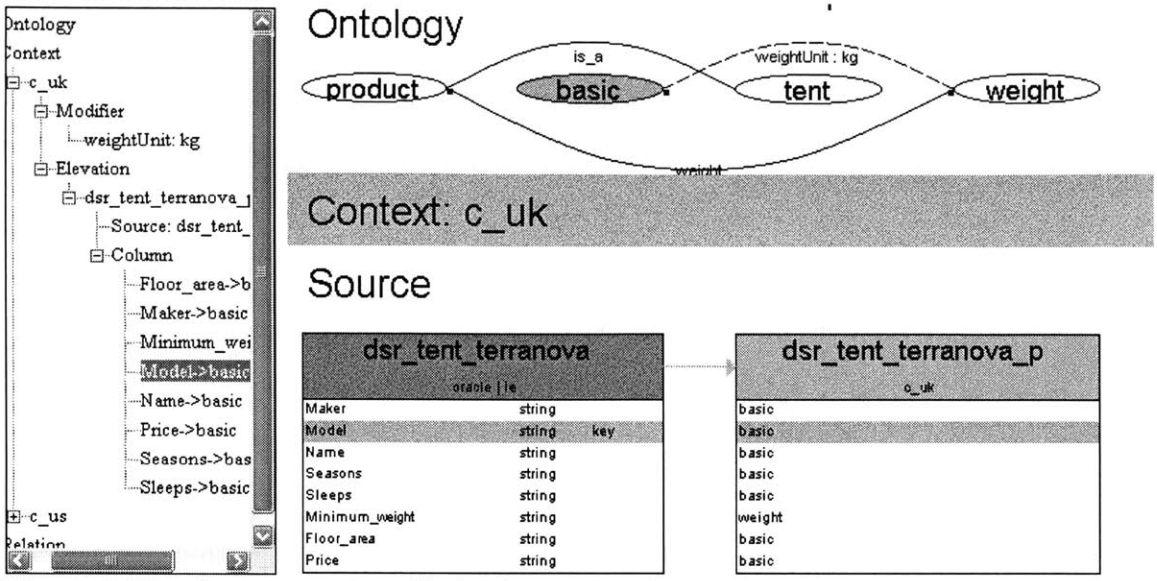


Figure 6.10 – Context *c_uk* applied to image panel

When a context is selected from the navigation tree, the content of the image panel changes to reflect the contextual information associated with the selected context. The contextual information displayed are the modifier values and the elevations. In Figure 6.10, the user has selected the *c_uk* context. Observe that, within the ontology section of the image panel, the label on the *weightUnit* modifier has the value *kg* next to the modifier name. Observe also that the elevated relation *dsr_tent_terranova_p* appears next to its underlying relation in the source section. Modifier values and elevations are regularly hidden until a context is selected. A context is selected when the user clicks on the name of the context in the navigation tree, or when the user clicks on any child nodes of the context node. At the instance the snapshot above was taken, the column named Model was selected. This selection causes the row representing the column, as well as the semantic type in the ontology to be highlighted.

6.4 Manual Layout Adjustment of Graphical Image Panel

In designing the graphical interface for the COIN Application Metadata Layer, a decision was made to forgo a perfectly generated image of an ontology and spend more development time on other aspects of the thesis. This decision is justified by the fact that sophisticated layout algorithms for ontological graphs already exist and there would be little value in reinventing these algorithms. There are graphical layout tools from the computer graphics community which could potentially be incorporated into the graphical interface. Thus, this area of work is saved for future version of the graphical interface.

Although achieving a perfect layout is not a goal of this thesis, much consideration has been made to design for a generally acceptable default layout of an ontology. The lines joining two semantic types in an ontology are curved, instead of straight, for the reason

that curved lines are less likely to overlap the nodes of a graph. This is especially true when the nodes of a graph are drawn in a rectangular lattice fashion, which is the case for the default layout of semantic type nodes. To further reduce the occurrence of crossing lines, the lines representing inheritance relationship between a semantic type and the *basic* semantic type are not drawn out explicitly. When reading the ontology, the reader should know that all semantic types inherit directly from *basic* unless another inheritance relationship is drawn in it place. Another measure to improve the layout is found in the smart font sizes applied for semantic type names. The ovals representing semantic types are fixed in size, but the font size of the labels representing the name of semantic types may shrink to accommodate a slightly longer name. However, the fonts can only shrink so much before hitting the physical limit of visibility. Therefore, it is often inevitable that manual intervention is required to correct some of the visually displeasing outcome of a generated ontological graph.

The most common visual fallbacks of a generated ontological graph are the overlapping of lines, the crossing between lines and the crossing of lines over nodes. Overlapping lines occur when a semantic type has two or more attributes whose ranges are the same semantic type. This common fallback is partially minimized by drawing edges with arcs. Since each arc has a degree of curvature associated with it, overlapping lines can be separated by assigning slightly different curvatures to them. If straight lines were used to draw arcs, overlapping lines cannot be resolveed regardless of the placement of the semantic types. Lines crossing each other or over nodes can be avoided if semantic type nodes are strategically placed on the graph to minimize crossing. The role of strategizing the layout of an ontological graph has been left for the user. The layout adjustment feature of the graphical interface provides the mean for the user to manually specify a layout.

The layout adjustment feature of the graphical interface allows users to manually resize and re-position the objects being displayed in the image panel. The metadata in a COIN application is initially displayed in the image panel using a default layout scheme. To begin adjusting the layout, the user must first export the current layout scheme to a file. This is done via the interface controls displayed in Figure 6.11. This figure corresponds to the area labeled 1 in Figure 6.1.

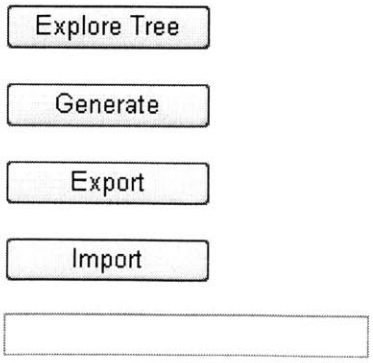


Figure 6.11 – Buttons for Export/Import of Image Panel

The process of exporting the current layout scheme to a file is accomplished by pressing the button labeled "Export". Before pressing the button, the user can supply a name for the export file by using the text box provided on the page. A default name in the format "ep*AppID*.xml", where *AppID* is the COIN application id, is used if no name is specified. The exported file is an XML document and it is shown in details in appendix C. The exported file is stored in the same directory on the server that is hosting the graphical interface. Therefore, the user can retrieve the exported file simply using the same URL path as the graphical interface.

We will illustrate the process of adjusting layout scheme by using the image generated for our tent example in Figure 6.3. Notice in the image that the line representing the modifier relationship *weightUnit*, and the line representing the inheritance relationship between *product* and *basic* semantic types are crossed. To untangle the lines, we can simply switch the locations of the ovals representing *product* and *basic*. From the exported layout scheme file, we find the semantic type nodes on *basic* and *product*, which are listed below.

```
<Ep_SemanticType>
 <name>basic</name>
  <x>160</x>
  <y>85</y>
 <selected>false</selected>
</Ep_SemanticType>
<Ep_SemanticType>
 <name>tent</name>
  <x>310</x>
  <y>85</y>
  <selected>false</selected>
</Ep_SemanticType>
```

Each semantic type is embedded in a tag labeled "Ep_SemanticType". The *name* attribute holds the name of the semantic type. The *x* and *y* attribute store the x and y coordinates, in pixel unit, of the oval in the image. The *selected* attribute denotes whether the semantic type is selected; if it is, the oval will be highlighted. Once we have switched the *x* and *y* attributes between the two semantic types, we can import the file by pressing the button labeled "Import" on the graphical interface page. The image generated from the new layout scheme is shown in Figure 6.12 below.

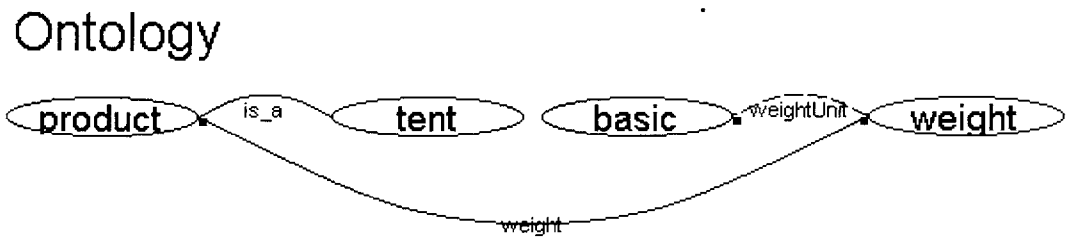


Figure 6.12 – Adjusted Ontology Layout

7 COIN Application Generator

The COIN Application Generator is a program that produces file-based COIN applications. The generator is used by the textual interface to create application files once the users have finished editing the application. The input to the generator is a ICM data structure. Recall that the textual interface stores the application metadata as it is being edited by the user. Once the user is done editing, the ICM is passed on to the generator to persist the metadata from memory to files.

The generator can produce several formats of application files, all of which are XML-based with the exception of Prolog. The XML representations that are currently supported are RDF, RuleML, RFML, and Prolog in HTML. However, any arbitrary format of representation for COIN application can be supported in the future as long as it is XML-based. One simply needs to write XSLT files for transformation between different XML-based representations. Since transformation is file-based rather than code-based, it is a snap to maintain and update the generator as new transformation requirement arises. Figure 7.1 illustrates the chain of metadata representation supported by the generator.

The COIN Application Generator is designed with two requirements in mind. The first requirement is the ability to generate the Prolog file used by the abduction engine. The second requirement is the ability to export and import COIN application in other formats beside Prolog. This is so that input from non-Prolog modeling communities can contribute in creating COIN applications. A suite of XML-based formats are chosen to represent COIN application because XML-based languages are progressively more accepted at representing metadata, and it is simple to transform between XML-based languages. In the following subsections, we will discuss the different formats of representation of COIN application, as well as the transformation from one format to another. The reader can refer to the background section for a brief introduction to the various XML technologies, such as XSLT, mentioned in this section.

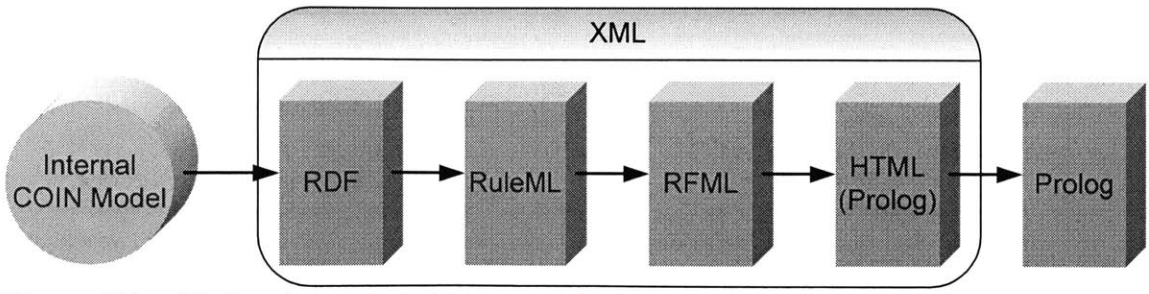


Figure 7.1 – Chain of Metadata Representation

7.1 COIN Application in RDF

Probably the most significant format of representation that the COIN Application Generator can produce, beside Prolog, is the RDF representation. Ontology is an integral part of a COIN application, and RDF is especially suitable for describing ontology because it is a language for describing resources. By having a representation of COIN

model in RDF, we are a step closer to potentially unleashing the rich source of ontology available in RDF.

Recall from the introduction on RDF, *class* and *property* are two common concepts in RDF. Every resource in a RDF document belongs to a RDF class. The relationships that can exist between classes are defined by RDF properties. Recall also that the Internal COIN Model is a collection of classes, where each class is characterized by a set of properties. The strong parallelism between the ICM data structure and the RDF modeling concepts naturally makes these two close neighbors in the chain of metadata representation. It is important to emphasize that the RDF representation of COIN model presented in this thesis does not make use of some common RDF elements in favor of maintaining the parallelism. A common RDF element that is not used is the *subclass* relationship, which is often used to express inheritance relationship. As we will see soon in the RDF schema, every relationship is an explicitly declared RDF property.

The first step in representing COIN model in RDF is to define a RDF Schema. The schema is for both communicating the semantics of the metadata, as well as validating the content of RDF documents. Due to the similarity between ICM and RDF concepts, the RDF schema is primarily a direct mapping from the ICM, class by class, property by property. The entire COIN application RDF schema can be found in appendix E. Figure 7.2 is a summary of the schema.

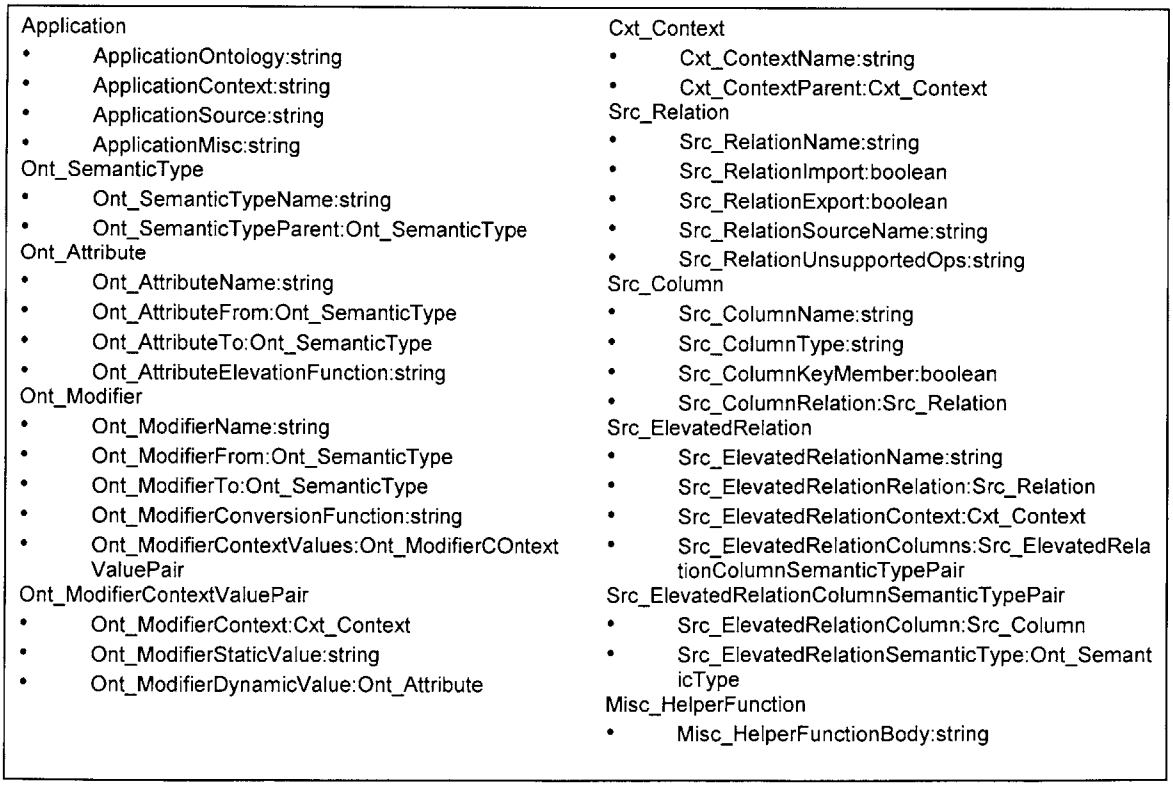


Figure 7.2 – COIN Application RDF Schema

The class *Application* in the RDF schema corresponds to the class *Coin_Model* in ICM. The *id* and *name* property of *Coin_Model* are not in the schema because they are stored in the COIN registry and are not needed for mediation by the abduction engine. Having the id and application name available in the RDF file would help in locating and identifying the files for an application; however, this is not the case in the current version of RDF files. The other three attributes of the *Coin_Model* store the ontology, context, and source information of the application. In RDF, these three groups of information are stored separately in three different RDF documents. The locations of these documents are stored in the properties of the *Application* RDF class. This modular design allows parts of a COIN application to be easily shared between applications. For instance, an ontology defined for one application can be immediately referenced from another by simply specifying the location of a file.

Similar to the ICM, the RDF classes *Ont_SemanticType*, *Ont_Attribute*, and *Ont_Modifier* collectively defines the ontology of a COIN application. The properties of these three classes correspond exactly to those in ICM, but with the following exceptions. The arrays of attributes and modifiers defined under *Ont_SemanticType* in ICM are redundant information created to improve efficiency in manipulation of the data structure. Therefore, they are not necessary in the RDF schema. Another exception is that there is no direct mapping in RDF schema for the hashtable data structure used in ICM. A class named *Ont_ModifierContextValuePair* is created instead in the RDF schema to similate the effect of a hashtable. This class has a property named *Ont_ModifierContext* to represent the key in the hashtable; the rest of the properties are possible types of value for the data to be stored in the hashtable.

The rest of the RDF schema is sufficiently similar to the corresponding ICM data structure. Having defined the RDF schema, the task of creating RDF documents for storing application metadata is simply a matter of complying with the schema. A complete RDF documents of the tent example is found in appendix G. The COIN Application Generator is the software module that is responsible for generate the RDF documents from metadata stored in ICM, as well as populating the ICM from RDF documents.

7.2 COIN Application in RuleML

The further down the chain of metadata representation a format gets, the greater the similarity between it and Prolog. The next format after RDF in the chain is RuleML. As its name suggests, RuleML is a markup language for describing rules. Since every statement in Prolog is a rule, the RuleML representation brings the metadata a step closer to being usable by the abduction engine.

As we have seen in the ICM and RDF representations, Metadata of a COIN application is knowledge that is not restricted to rule-based representation. To make the metadata usable by the abduction engine, however, the metadata must be expressed as a collection of Prolog rule statements. The layout of these Prolog rule statements drive the layout of the RuleML representation of COIN application. There are only a handful of tags that are used to represent Prolog rules in RuleML. These tags are:

<imp>	a rule
<_head>	head of a rule
<_body>	body of a rule
<atom>	a clause inside the head or body of a rule
<var>	a variable
<ind>	a constant
<_opr>	signifies the start of a <rel>
<rel>	the predicate of an <atom>
<cterm>	a collection of constant terms
<_opc>	signifies the start of a <ctor>
<ctor>	the predicate of a <cterm>

Let us demonstrate how these tags are used to represent a Prolog rule. Consider the Prolog rule for declaration of attribute *weight* for the semantic type *product*:

```
rule(attributes(product,[weight]),(true)).
```

The same rule can be represented using RuleML as follows, using the tags we have identified above:

```
<imp>
 <_head>
  <atom>
   <_opr><rel>attributes</rel></_opr>
   <ind>product</ind>
   <cterm>
    <_opc><ctor /></_opc>
    <ind>weight</ind>
   </cterm>
  </atom>
 </_head>
 <_body>
  <atom>
   <_opr><rel /></_opr>
   <ind>true</ind>
  </atom>
 </_body>
</imp>
```

The <imp> tag corresponds to the *rule* predicate in Prolog. The content of each <imp> element is the head and body of a rule. The <_head> tag encapsulates the atom that defines the head of a rule. The head atom consists of relation, or predicate in Prolog terms, and the atoms of the predicate. In the example above, the predicate of the head is *attributes*, and the associated atoms are *product* and *[weight]*. The <ind> tag is a used to indicate that *product* is not a variable. The processor of the Prolog rule knows that

product is the name of the semantic type; however, this fact is not important in the representation level. One only needs to decide whether the atom is a variable or not on this level. The atom *[weight]* is a collection of one constant term, in this case it is the name of an attribute. The brackets around the term signify that it is a collection, so the <cterm> tag is used. Within <cterm>, it is possible to define a predicate with the <ctor> tag, but there is none in this case. Since *weight* is not a variable, the <ind> tag is used to represent it. The <_body> tag encapsulates the atom that defines the body of a rule. The body atom can consist of predicate and atoms associated with the predicate. Since the body is empty in this case, there is no need for a predicate but just the word "true" enclosed in <ind>. The entire tent example expressed in RuleML can be found in appendix G.

So far we have established a schema for representing Prolog rules in RuleML; however, we still need a mechanism to transform the metadata into this representation. Since there already exists a representation of application metadata in RDF, which is a form of XML, it is a natural to use XSLT to transform the metadata from RDF to RuleML. Recall that XSLT is a language for transforming one XML document into another. The XSLT stylesheet that is responsible for the transformation is the very key that separates knowledge from code understandable by abduction engine for context mediation. When the RDF documents and the XSLT stylesheet are processed by a XSLT processor, the output is a RuleML document representing the application metadata in the schema established above. As will be discussed very soon, the RuleML document will eventually be more XLST transformation into a Prolog document. The complete XSLT stylesheet for transforming RDF to RuleML can be found in appendix F.

7.3 COIN Model in RFML

The next stage in the chain of metadata representation is RFML. RFML is more similar to Prolog representation than RuleML because RFML is designed specifically for representing relational language, such as Horn logic, or Prolog. Unlike RuleML, RFML is positional. This means that the positions of elements within a clause can provide information about the elements without the need for special tags. This means that the number of tags needed to represent Prolog rules are even less than RuleML. These tags are:

<hn>	a Horn clause
<pattop>	head of a clause
<callop>	body of a clause
<var>	a variable
<con>	a constant
<struc>	a structure/a collection of <con>, <var>, or <struc>

Let us demonstrate how these tags are used to represent a Prolog rule. Consider the same Prolog rule previously used in RuleML illustration. The following rule is declaration of attribute *weight* for the semantic type *product*:

```
rule(attributes(product,[weight]),(true)).
```

The same rule can be represented using RFML as follows, using the tags we have identified above:

```
<hn>
 <pattop>
  <con>attributes</con><con>product</con>
  <struc><con></con><con>weight</con></struc>
 </pattop>
 <callop><con></con><con>true</con></callop>
</hn>
```

Every Horn clause is enclosed by a <hn> tag, and consists of a <pattop> element and one or more <callop> elements. The first element in a <pattop> is always a predicate. In the example above, the predicate *attributes* is associated with the constant *product* and the structure of one constant, *weight*. Notice that the first <con> element in the structure is empty. This means that there is no predicate associated with this structure. Since the body of the rule is empty, the <callop> element contains just the constant "true" and no predicate. The entire tent example expressed in RFML can be found in appendix G.

Since both RuleML and RFML are XML-based, we can use XSLT again for transforming the metadata represented in RFML into RFML representation. Since there already exists in the modeling community a XSLT stylesheet for transforming from RuleML to RFML [15], some work is saved in that the generator can just use the existing stylesheet. The RuleML to RFML stylesheet can be found in appendix F.

7.4 COIN Model in Prolog (HTML and Regular Text)

The next stage in the chain of metadata representation is Prolog. There are, however, two flavors of Prolog representation that are produced by the generator. One is HTML version, and the other is regular text version. The regular text version of Prolog is the Prolog file that is directly used by the abduction engine. The HTML version is for enhanced visualization of the Prolog file, which could be useful sometimes for debugging and demonstration purposes.

Let us demonstrate how a Prolog rule would look in HTML format. Consider the same Prolog rule previously used in RuleML illustration. The following rule is declaration of attribute *weight* for the semantic type *product*:

```
rule(attributes(product,[weight]),(true)).
```

Below is the same rule in HTML format:

```
rule(<strong><b>attributes</b></strong>(<b>product</b><tt>,</tt><b></b>
[<b>weight</b>])<tt>,</tt><strong><b></b></strong>(<b>true</b>)).<br />
```

If viewed in a web browser, the HTML code above would look like the following:

rule(**attributes(product, [weight]), (true)**).

The HTML version of the Prolog emphasizes certain elements in a rule by painting them differently. The predicates, for example, are painted bold and strong, where as regular items are either bold or plain. The commas are always painted with typewriter font so they stand out more against regular font. The entire tent example expressed in HTML can be found in appendix G.

The transformation from RFML to HTML is done via XSLT stylesheet. This XSLT stylesheet can be found in appendix F. The transformation from HTML to regular Prolog is simply done by removing all tags from the HTML document. This is done by software code within the application generator.

7.5 File-Based Interface

The File-Based Interface component of the COIN Application Metadata Layer is designed to facilitate the assimilation of COIN applications developed outside of the user interface tools. Interacting with the file-based interface amounts to placing the application files in a web accessible location, and updating the COIN registry of that location. All of the file formats from the chain of metadata representation are supported in the file-based interface. The user can supply a COIN applications in any of those file formats, and the COIN application generator will generate the Prolog file necessary for context mediation by the abduction engine.

Notice that the generator only transforms the files down the chain of representation because the generator currently does not have the XSLT stylesheets necessary for upward transformation. For example, if the user supplied the application in RuleML, the generator will produce the RFML, HTML, and Prolog representations of that application. The generator will not, however, produce the RDF representation. If the application is already supplied in Prolog format, then no transformation is necessary. Since the XSLT stylesheets are web accessible, the user can bypass the generator and produce the desired representations via XSLT processor of their own.

8 COIN Registry

The COIN Registry in the COIN Application Metadata Layer is primarily a file-based directory of COIN applications. Information necessary for mediation work by eCOIN are also accessible from the COIN registry. It is important to point out the distinction that making information accessing is not the same as storing the information. The registry is simply a structured set of XML documents that records and maintains the location, or URL, of specific metadata files for COIN applications. The actual content in the metadata files are oblivious to the registry. This design has the benefit of shielding the registry from re-design in the event of any changes to the COIN model. Currently, there are two methods of adding entries to the registry. One way is through the textual interface; the other way is to edit the registry files manually. Any COIN applications created through the textual interface are stored in a default location and the location of the application metadata files are recorded automatically in the registry. Figure 8.1 illustrates the framework of the COIN registry structure.

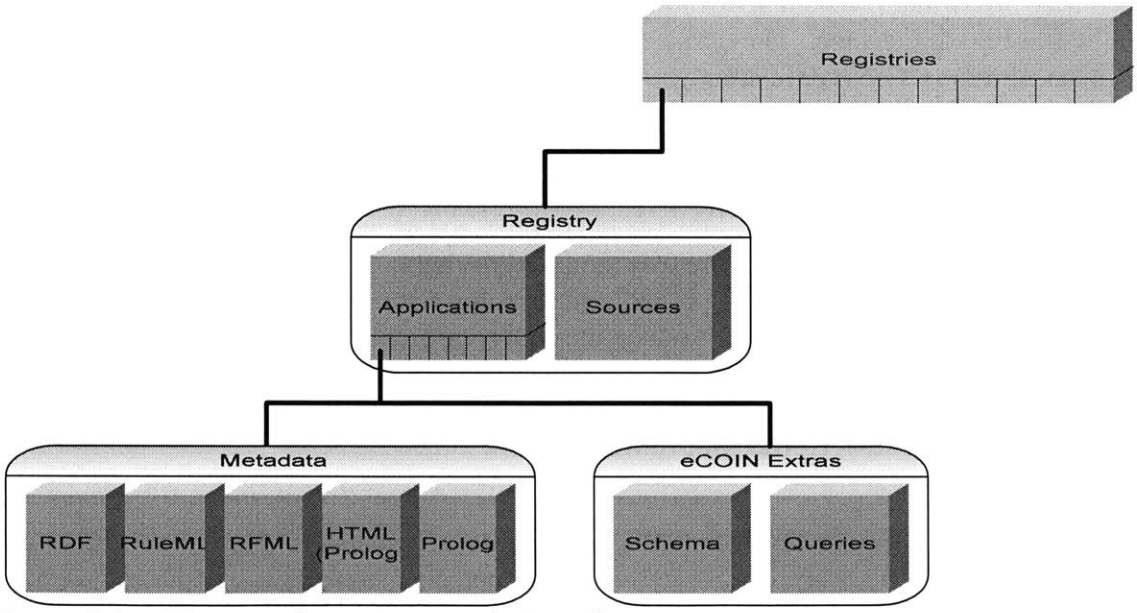


Figure 8.1 – COIN Registry Framework Overview

The COIN registry framework is organized into a hierarchy of files three levels in depth. Each level is populated with one or more XML files. In the following subsections, the individual files that populate the registry framework are discussed. The actual schema, or DTD files, that characterize these XML files are included in appendix D.

8.1 Registries

At the top level of the framework is a master registry of COIN registries. This level is populated by a file named "registries.xml", which stores all COIN registries by location. Under this framework, a separate registry exists for each different location. This

provides the flexibility to support multiple independent environments of COIN application development. Although the two registries are maintained separately, application from one location can be located and queried from another location through the master registry. Below is a XML example that illustrates the master registry. The XML tag labeled "REGISTRIES" encapsulates multiple "REGISTRY" tags. Each "REGISTRY" tag represents a COIN registry. The *name* attribute denotes the location of the registry and uniquely identifies it. In this case, the locations are MIT and MUST. The body of the registry is denoted here by "..." and will be discussed in details in the next section.

```
<REGISTRIES>
 <REGISTRY name="MIT">
 ...
 </REGISTRY>
 <REGISTRY name="MUST">
 ...
 </REGISTRY>
</REGISTRIES>
```

8.2 Registry

The level below the master registry resides the actual COIN registries, one per location. Each registry is a directory of COIN applications. Each registry also has a directory of data sources. These sources are maintained outside of a COIN application because each source could potentially be shared by multiple applications. From this description, it follows that a registry is populated by two files. Below is a XML example that illustrates the registry. The elements within the tag labeled "REGISTRY" are the content of a COIN registry. The element labeled "APPLICATIONS" stores the location of a file about the applications in this registry. The other element, labeled "SOURCES", stores the location of a file describing the sources used by applications of this registry.

```
<REGISTRY name="MIT">
 <APPLICATIONS>http://localhost/appEditor/registry/applications.xml</APPLICATIONS>
 <SOURCES>http://localhost/appEditor/registry/sources.xml</SOURCES>
</REGISTRY>
```

Lets take a closer look at each of these two files in the following subsections.

8.2.1 Applications

The applications directory maintains the location of metadata files for each application under a registry. For each representation of application metadata introduced in the *File-based Interface* section, there is a record of its location in the directory. Below is a XML example of the directory. The tag labeled "APPLICATIONS" encapsulates multiple application element, each labeled "APP". Each of these "APP" tag represents a COIN application. The *id* attributes of the application element denotes an application ID, and this uniquely identifies a COIN application. The *name* attribute denotes the name of an application. Within the "APP" tag are five tags that record the locations of various XML-based representation of application metadata. There is a "RDF" tag for the RDF representation, a "RULEML" tag for RuleML representation, a "RFML" tag for RFML

representation, a PROLOGHTML tag for the HTML representation of the Prolog application file, and finally a PROLOG tag for the actual Prolog application file. The actual application metadata files constitute the bottom layer of the registry framework. These files are referenced in the registry, but they are not part of the registry. This allows the content of these files to change frequently, while remaining at constant locations.

```
<APPLICATIONS>
 <APP id="392" name="Simple Disaster Relief">
  <RDF>http://localhost/appEditor/apps/rdf/application392.rdf</RDF>
  <RULEML>http://localhost/appEditor/apps/ruleml/application392.ruleml</RULEML>
  <RFML>http://localhost/appEditor/apps/rfml/application392.rfml</RFML>
  <PROLOGHTML>http://localhost/appEditor/apps/prologhtml/application392.html</PROLOGHTML>
  <PROLOG>http://localhost/appEditor/apps/prolog/application392.pl</PROLOG>
  <SCHEMA>http://localhost/appEditor/registry/schema/schema392.xml</SCHEMA>
  <QUERIES>http://localhost/appEditor/registry/queries/queries392.xml</QUERIES>
 </APP>
 <APP id="391" name="Disaster Relief">
 ...
 </APP>
</APPLICATIONS>
```

There are two more elements within the "APP" tag, and they are labeled "SCHEMA" and "QUERIES". The "SCHEMA" tag stores the location of the schema file that is needed by the eCOIN in planning and execution of mediated queries. The mediated queries could be pre-defined in files. The location of a queries file is stored inside the "QUERIES" tag.

8.2.2 Sources

In addition to the application directory, there is a source directory in the registry. The source directory is a file that maintains the connection information for every source used by the applications of that registry. Below is a XML example of the source directory. The tag labeled "SOURCES" contains a "SOURCE" element for each source defined in the registry. The *name* attribute denotes the name of the source, and it uniquely identifies a source. The type attribute denotes the type of the source. Within the "SOURCE" element are four elements. The element labeled "URL" stores the location where the source can be reached. The elements labeled "USERNAME" and "PASSWORD" store the username and password needed to connect to a source. Finally, the element labeled "MAXCONNECTION" is a number that tells the eCOIN planner the optimum number of parallel connections to make to a single source.

```
<SOURCES>
 <SOURCE name="oracle" type="DATABASE">
  <URL>jdbc:oracle:thin:@localhost:1521:coin</URL>
  <USERNAME>system</USERNAME>
  <PASSWORD>manager</PASSWORD>
  <MAXCONNECTION>1</MAXCONNECTION>
 </SOURCE>
 <SOURCE name="cameleon" type="CAMELEON">
 ...
 </SOURCE>
</SOURCES>
```

9 Technology Merging

9.1 Integration with eCOIN

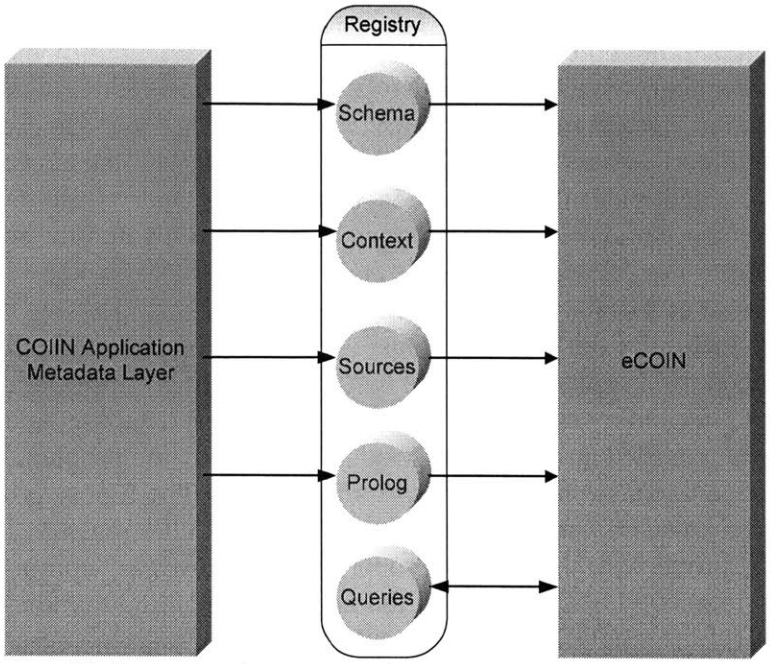


Figure 9.1 – Flow of Information between COIN Layers

The ultimate purpose of the COIN Application Metadata Layer is to produce COIN application that supports mediated query on eCOIN. For this purpose, the metadata layer can be viewed as a producer of context sensitive metadata, while eCOIN is the consumer of metadata. The interaction between the COIN Application Metadata Layer and the eCOIN can be summarized by Figure 9.1. The COIN registry acts as the intermediate resource broker between the metadata layer and eCOIN. Whenever COIN applications are created, the location of the metadata files are recorded in the registry. On the actual context mediation side, COIN applications are retrieved from the registry and loaded into eCOIN. There are five items of information currently needed by eCOIN for mediation work; they are relation schema, context declaration, source connection properties, application metadata, and queries. Since the registry manages resources by storing location of files, each item of information must be file-based, as discussed in the section on COIN Registry. There are actually overlapping areas between the five items of information mentioned. For instance, the schema and context information duplicates the relation and context axioms found in the Prolog file, which is the primary source of application metadata for eCOIN. Reason behind the duplication of information is the lack of a mechanism in the current implementation of eCOIN to extract schema and context information from the Prolog file. More light will be shredded on this topic in the section on future work. Let us now go over how each item of information is used by eCOIN. Please refer to Tarik's thesis [4] for a detail explanation of the concept.

Relation schema is the information that allows eCOIN to query each data source as a relational database. The schema details the name and type of each attribute in a relation.

Source connection properties provide the connection information, such as resource location and authentication, needed for eCOIN to connect to data sources. At the heart of context mediation is COIN application, which is expressed in Prolog and used by the abduction engine in eCOIN. Furthermore, context mediation cannot happen unless there are actually queries to mediate on. The queries are usually supplied by users of eCOIN. Each query is associated with a context, also known as the receiver context. The information found in context declaration provides the needed link. The flow of information is unilateral from the metadata layer to eCOIN, with the exception of queries. Queries are not created under the metadata layer but by the Query Builder under eCOIN. It is the role of Query Builder to broker queries between eCOIN and the registry.

9.2 Modification to eCOIN

Prior to the introduction of the COIN Registry, there was no concerted effort in addressing the need for a central registry of metadata handling. Information that belongs in the registry and needed by eCOIN could be found in files, database, and software code. While the registry provided an organized structure to remedy the problem, modification to eCOIN is needed to enable interaction with the registry. The first step in enabling the interaction is to make eCOIN aware of the location of the COIN registry. This is done by adding the *regurl* property to the eCOIN properties file, known as "coinprops". The *regurl* property stores the URL location of the registry. Recalling that the registry is a hierarchy of XML files, the URL of the file at the top of the registry hierarchy is the value that should be stored in the *regurl* property. At initialization, eCOIN accesses the registry and automatically retrieves all the information it needs to carry out mediation for any COIN application. Since the registry is XML-based, the capability to parse XML is also added to eCOIN.

Figure 9.2 – Navigation Bar for Accessing Textual and Graphical Interfaces

Beside the registry, new to eCOIN are the textual and graphical interfaces. There has always been a navigation on the eCOIN interface that allowed user to quickly access the application metadata through the GMM Graphical Editor. To give eCOIN access to the new interfaces, the navigation bar has to be modified. Figure 9.2 shows the new navigation bar. The first item in the navigation bar is labeled "Text Interface". It takes the user directly to the front page of the textual interface. The next three items, labeled "Ontology", "Context", and "Source", go to the ontology, context, and source page of the textual interface respectively. The item labeled "Graphical Viewer" takes user to the graphical interface. Finally, the one labeled "Internal Representation" brings the user to the application file viewer.

10 Disaster Relief Application

Although the tent example served us well in demonstrating the capabilities of the COIN application metadata layer, it is not a realistic example of a COIN application. To stress test and better assess the capabilities and limitation of the various components of the metadata layer, the Disaster Relief application is used. In this chapter, an overview of the Disaster Relief application is first presented, follow by a discussion on the performance of the metadata layer in various stages of the application development.

10.1 Disaster Relief Application Overview

Whenever natural or man-made disasters strike, the timely deployment of disaster relief aid is crucial to the recovery process. Since disaster relief efforts are often join efforts between multiple agencies spanning multiple countries, the contextual difference between agencies can delay the relief effort significantly. The purpose of the Disaster Relief application is to demonstrate the ability of the COIN framework in resolving contextual differences in the realm of disaster relief. In many disaster scenarios, tents play a big role in disaster relief because victims are often without shelters and tents must be set up to house those victims. Therefore, the disaster relief application focuses on tents and the contextual issues associated with them. The kind of users interested in this application might be logistics personnel who needs to find out which tents are suitable for the given budget and physical constraints. In the following subsections, the ontology, sources and contexts of the application are presented, followed by a discussion on the performance of the user interfaces in capturing those metadata.

10.2 Ontology

The disaster relief application is built with expandability in mind so that more supply items beside tent can easily be supported by the application in the future. This is accomplished on the ontology level by modeling for reusable components. Figure 10.1 shows the disaster relief ontology with semantic types grouped into components, which are indicated by the enclosing boxes. Let us consider the physical measurement component, where the semantic type *physicalMeasurement* resides. There are four semantic types that inherit from *physicalMeasurement*, and they are *weight*, *length*, *area*, and *volume*. The *weight* semantic type has the modifier *weightUnit*, and the *length* semantic type has modifier *lengthUnit*. Both are for converting scalar measurements between different units of measurement. Currently, weightUnit and lengthUnit are distinct modifiers. But due to their similar nature, it is possible to have just one modifier, i.e. *physicalUnit*, on the semantic type *physicalMeasurement*, and let its children inherit the modifier. Since many products can be described in terms of physical quantities, the physical measurement component is highly reusable.

Another useful component is the monetary component where *monetaryValue* semantic type resides. This component models the *price* semantic type and *currency* modifier. In reality, this component can be much richer because of the broad range of use for money amount. Closely related to the monetary value component is the exchange rate component, where the semantic type *exchangeRate* resides. This component has a dateFormat modifier for converting date format. Finally, the product component is where

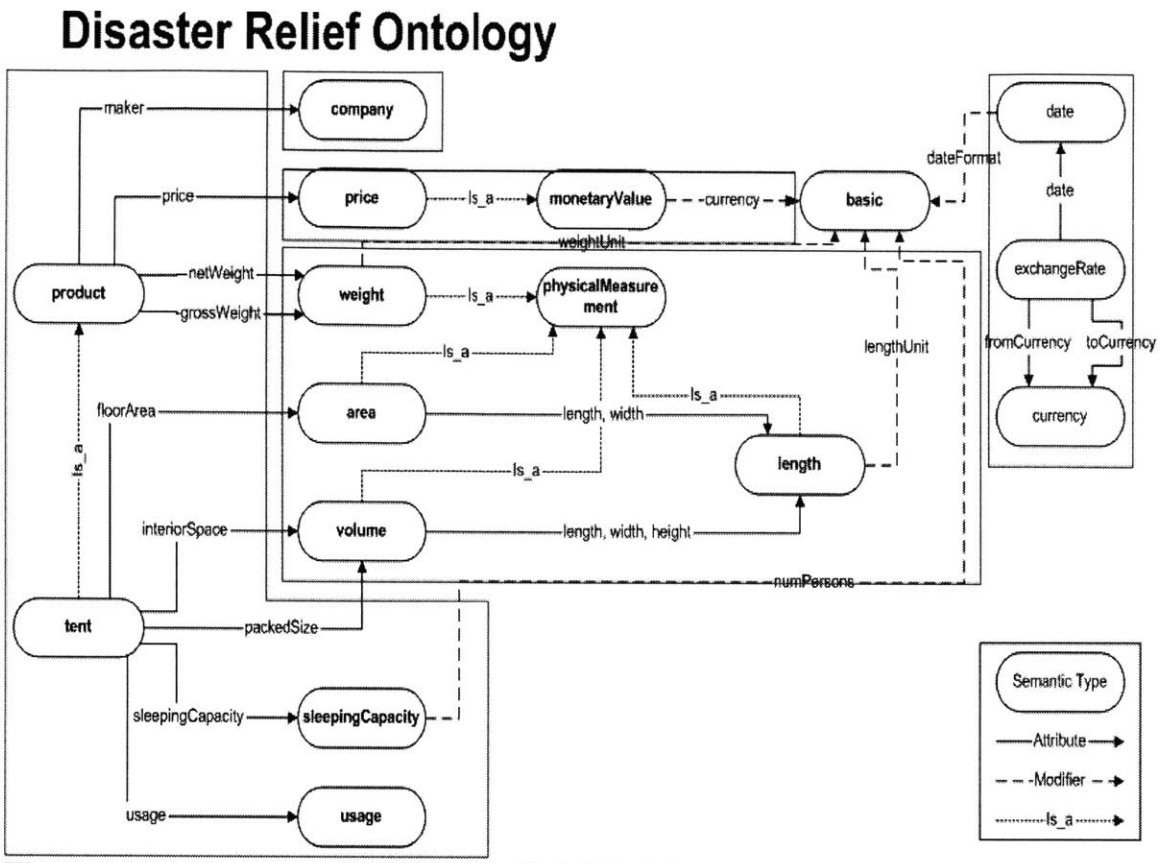


Figure 10.1 – Componentized Disaster Relief Ontology

the semantic type *product* and its child reside. There is only the *tent* semantic type that inherits from product currently. But more relief supply items can be added as the application grows. Four basic attributes are identified for the *product* semantic type, and they are *maker*, *price*, *netWeight*, and *grossWeight*. There are obviously other plausible attributes of *product* that are not modeled in the ontology. For now, we are primarily interested in an ontology that allows for room to grow as more relief items are supported. In addition to the attributes it inherits from *product*, the *tent* semantic type also has attributes of its own, such as *floorArea*, *interiorSpace*, *packetSize*, *sleepingCapacity* and *usage*. The modifier identified for *tent* is numPerson, which is used for converting the number of persons that can fit within a tent in different contexts.

The disaster relief ontology is entered in the system through the textual interface. Figure 10.2 and 10.3 show a summarized view of the ontology through the navigation tree of the graphical interface. Notice that the exchange rate component from the original disaster relief ontology is not explicitly entered into the system. This is because the exchange rate component actually belongs to the currency function from the function library. Recall from the section on conversion function library that, in defining conversion functions, one also needs to define all supporting elements. The exchange rate component plays only a supporting role in currency conversion; therefore, it is defined under the currency library function.

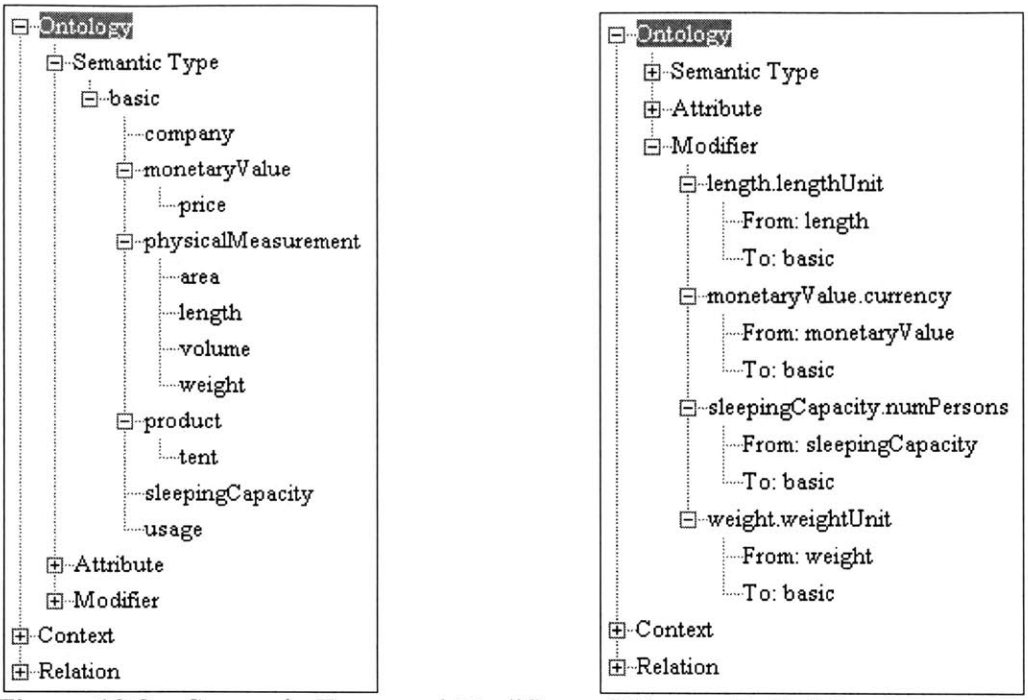


Figure 10.2 – Semantic Types and Modifiers of Disaster Relief Ontology

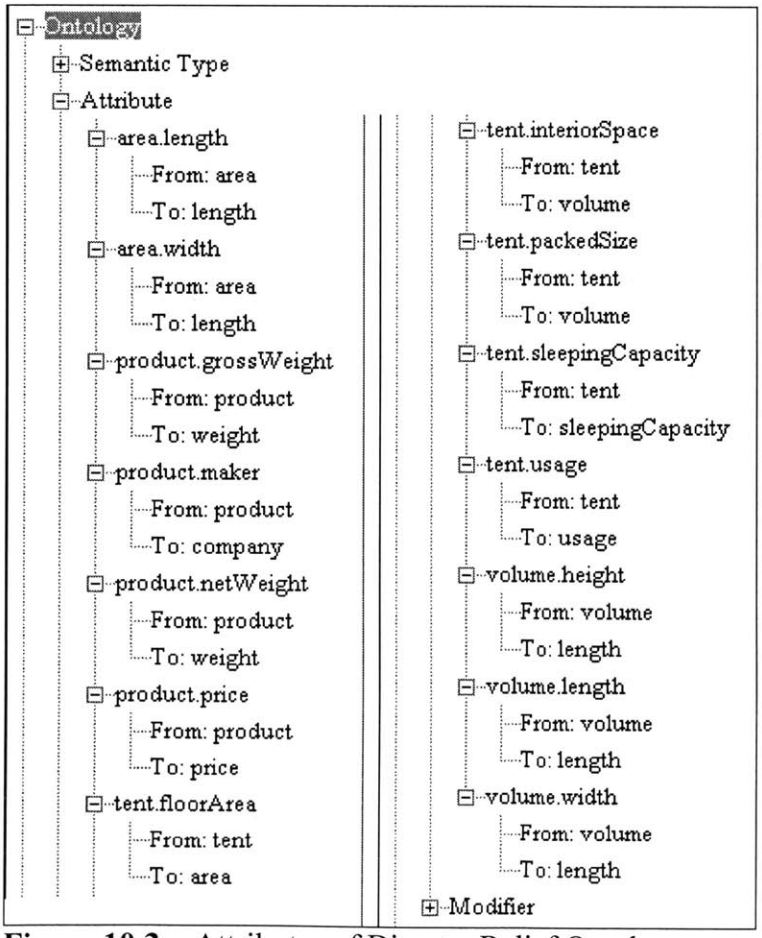


Figure 10.3 – Attributes of Disaster Relief Ontology

10.3 Source

There are four tent suppliers identified for the disaster relief application. The suppliers Terranova and Antarctica are England based companies, and the other two suppliers, eFunctional and Outback, are US based companies. All four suppliers have online presence, which means that we can write Cameleon web wrappers to query those sites as relational data sources. However, to keep things simple in our initial approach, we have decided to model the tent description from each site as tables in a local relational database. We take four different tents from each of the merchant sites to populate the corresponding tables with tent properties. Figure 10.4 shows a snapshot of the tent data as they exist on the database.

Terranova

MODEL	USAGE	PERSONS	WEIGHT	PACKED_LENGTH	PACKED_WIDTH	INTERIOR_LENGTH	INTERIOR_WIDTH	INTERIOR_HEIGHT	PRICE
solar	3	2	1.85	13	44	225	92	105	249
solar_2	4	2	2.4	14	56	216	125	98	299
ultra_quasar	4	2	3.6	18	56	220	132	104	390
ultra_hypersp...	4	3	4.4	18	74	206	183	122	433.9

eFunctional

MAKER	MODEL	NAME	SEASONS	SLEEPS	MINIMUM_WEIGHT	FLOOR_AREA	PRICE
Coleman	9810-806	Inyo	3	2	4	95x52	76.95
Coleman	9830-806	Mantis	4	2	6.14	92x62	129.95
Coleman	9840-808	Lynx	4	3	10.03	94x94	190.95
Coleman	9820-605	Oryx	3	2	5.03	88x56	97.95

Outback

MAKER	MODEL	SEASON	CAPACITY	TRAIL_WEIGHT	PACKED_WEIGHT	FLOOR_AREA	PRICE
Marmot	Area_51	3	2	5.03	5.97	44	189.99
Sierra	Meteor_Light	3	2	6.12	7.06	40	239.99
Sierra	Clip_Flashlight	3	3	4.1	5.08	44	187.99
Sierra	Night_Watch	4	2	6.14	7.1	35	265.94

Antarctica

NAME	NET_WEIGHT	GROSS_WEIGHT	INTERIOR	PRICE
southern_cr...	3.5	4	300x142x110	260
pole_star	5	5.5	358x190x125	400
south_pole	4	4.5	315x150x110	330
northern_light	2.5	3	280x160x160	600

Figure 10.4 – Data from Tent Suppliers

Each of the four suppliers is model as a relation in the disaster relief application. These relations are entered into the system via the textual interface. Figure 10.5 shows the navigation tree with the "Relation" node fully expanded to show details of each relation.

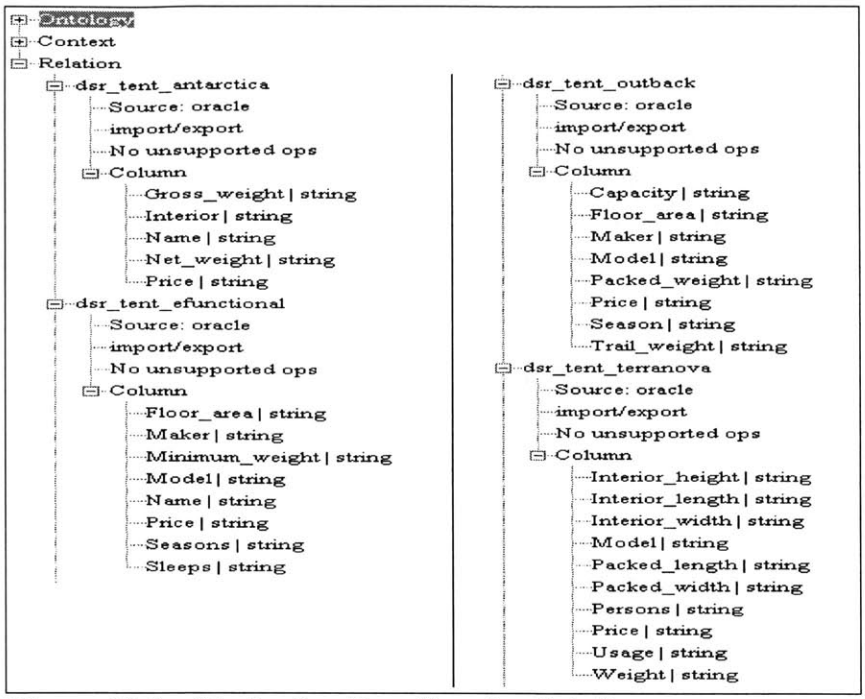


Figure 10.5 – Navigation Tree Showing Relations

Behind the scene, there are two more sources that support the application. They are Cameleon relations for obtaining the currency rate for currency conversion, and unit conversion factor for physical unit conversion. However, the users of the textual interface do not need to know how to define them to use them. Please refer to Appendix I for their definition.

10.4 Context

There are four contextual differences identified for the disaster relief application. They are the currency, weight and length physical units, and sleeping area per person. Contextual difference in currency simply means that the money amount from each source might be expressed in different currency. The physical unit difference means the physical measurement from each source might be expressed in different physical units. Finally, the contextual difference on sleeping area per person refers to the difference in acceptable sleeping floor space per person.

There are four source contexts and three receiver contexts identified for the application. The source contexts are *c_terranova* for the tent supplier Terranova, *c_antarctica* for Antarctica, *c_efunctional* for eFunctional, and *c_outback* for Outback. The receiver contexts are *c_us* for the country United States, *c_uk* for England, and *c_cuba* for Cuba. Let us examine the modifiers and elevations for these source and receiver contexts.

10.4.1 Modifiers

The modifiers which are currently used in the disaster relief application are *currency*, *weightUnit*, *lengthUnit*, and *numPersons*. These modifiers correspond to the four contextual differences we have identified for the application. The modifier value for *currency* is currency symbol, i.e. *USD*. The modifier value for *weightUnit* is a unit for

weight, i.e. *kg* and *lb*. The modifier value for *lengthUnit* is a unit for length, such as *cm* and *in*. Finally, the modifier value for *numPersons* is a string which reads "width x height". The length and width corresponds to the minimum floor size required per person. The unit of length at the end of the string stands for the physical unit the numbers are represented in. Below is a summary of the modifier values for each source and receiver context, listed in the order *currency*, *weightUnit*, *lengthUnit*, and *numPersons*.

Source Contexts:

- Terra Nova (GBP, kg, cm, "122.0x244.0cm")
- Antarctica (GBP, kg, cm, "122.0x244.0cm")
- eFunctional (USD, lb, in, "48.00x96.00in")
- Outback (USD, lb, in, "48.00x96.00in")

Receiver Contexts:

- US (USD, lb, in, "48.00x96.00in")
- UK (GBP, kg, cm, "122.0x244.0cm")
- Cuba (CUP, kg, cm, "91.00x213.0cm")

The modifier metadata are entered through the textual interface. Figure 10.6 shows the navigation tree with all modifier nodes expanded.

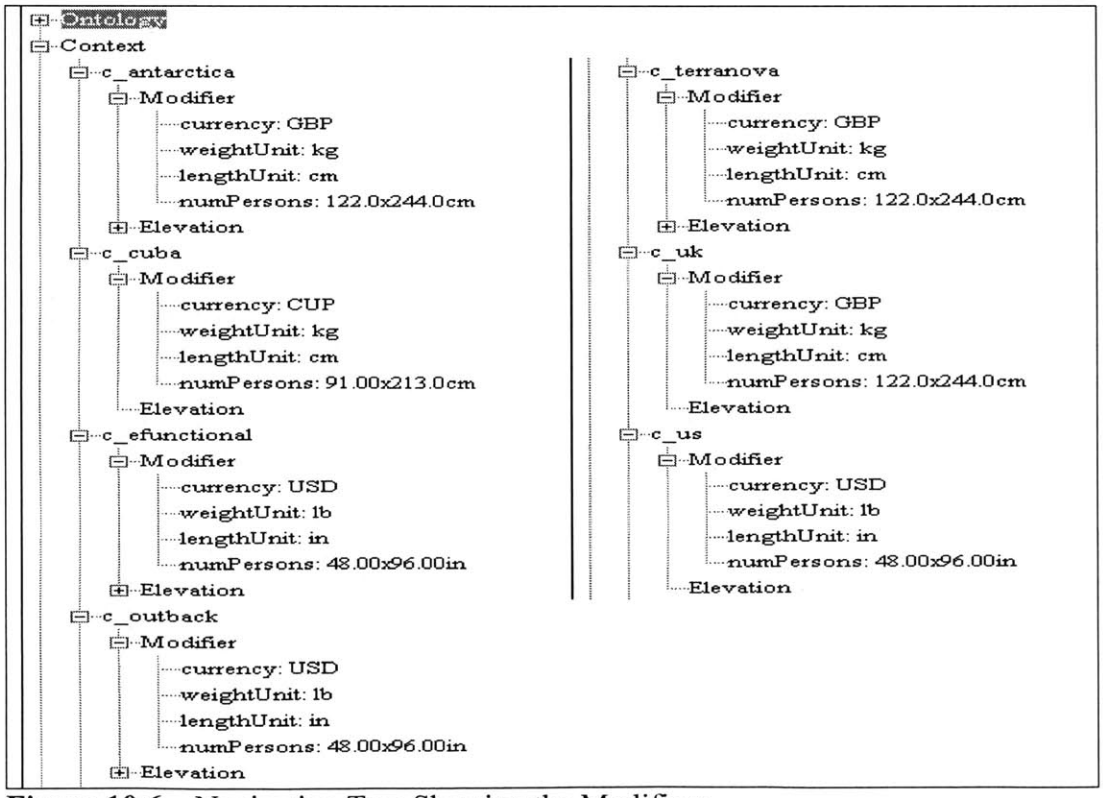


Figure 10.6 – Navigation Tree Showing the Modifiers

Since the modifiers weightUnit and lengthUnit are very similar, there are only three distinct conversion functions in the application. These conversion functions are responsible for currency conversion, physical unit conversion, and conversion on sleeping capacity. The currency and physical unit conversions are already part of the library of functions that are pre-defined for users of the textual interface. Please refer to appendix I for their definition. The remaining conversion function is the one on sleeping capacity. The prolog statement below defines this conversion function.

```
rule(cvt(commutative, sleepingCapacity, _O, numPersons, ctxt, Ms, Vs,
      Mt, Vt),
      ({substring(Ms, 1, 5, Fv1), substring(Ms, 7, 5, Fv2),
      substring(Ms, 12, 2, Fu), substring(Mt, 1, 5, Tv1),
      substring(Mt, 7, 5, Tv2), substring(Mt, 12, 2, Tu)},
      unit_conv_p(Fu2, Tu2, Uf),
      value(Fu2, ctxt, Fu),
      value(Tu2, ctxt, Tu),
      value(Uf, ctxt, Ufv),
      multiply(Fv1, Ufv, Fv11),
      multiply(Fv2, Ufv, Fv21),
      multiply(Fv11, Fv21, Fv3),
      multiply(Vs, Fv3, Ttv),
      divide(Ttv, Tv1, Vt1),
      divide(Vt1, Tv2, Vt))).
```

This is novel conversion function because it uses the substring function in the native Prolog processor to parse the modifier value for width, height and unit values. Recall from earlier that the modifier value for *numPersons* modifier is in the form "width x height unit". Next, the conversion function uses the unit conversion relation to convert the width and height values from both contexts into a common unit of measurement. Finally, it performs the arithmetic to find out how many persons from one context can fit in an area defined for one person in another context. The ratio is the result of the conversion function.

10.4.2 Elevations

The elevations for the columns in our relations are straight forward in that each column is mapped to a dedicated semantic type in the ontology. The only exception is that the weight of a tent can be distinguished into net weight and gross weight. Net weight means the weight of the tent without any packaging, and gross weight means the weight of the tent plus all packaging materials. Since both are fundamentally weight types, they are elevated to the same semantic type *weight*. Figure 10.7 shows the navigation tree with elevation nodes expanded.

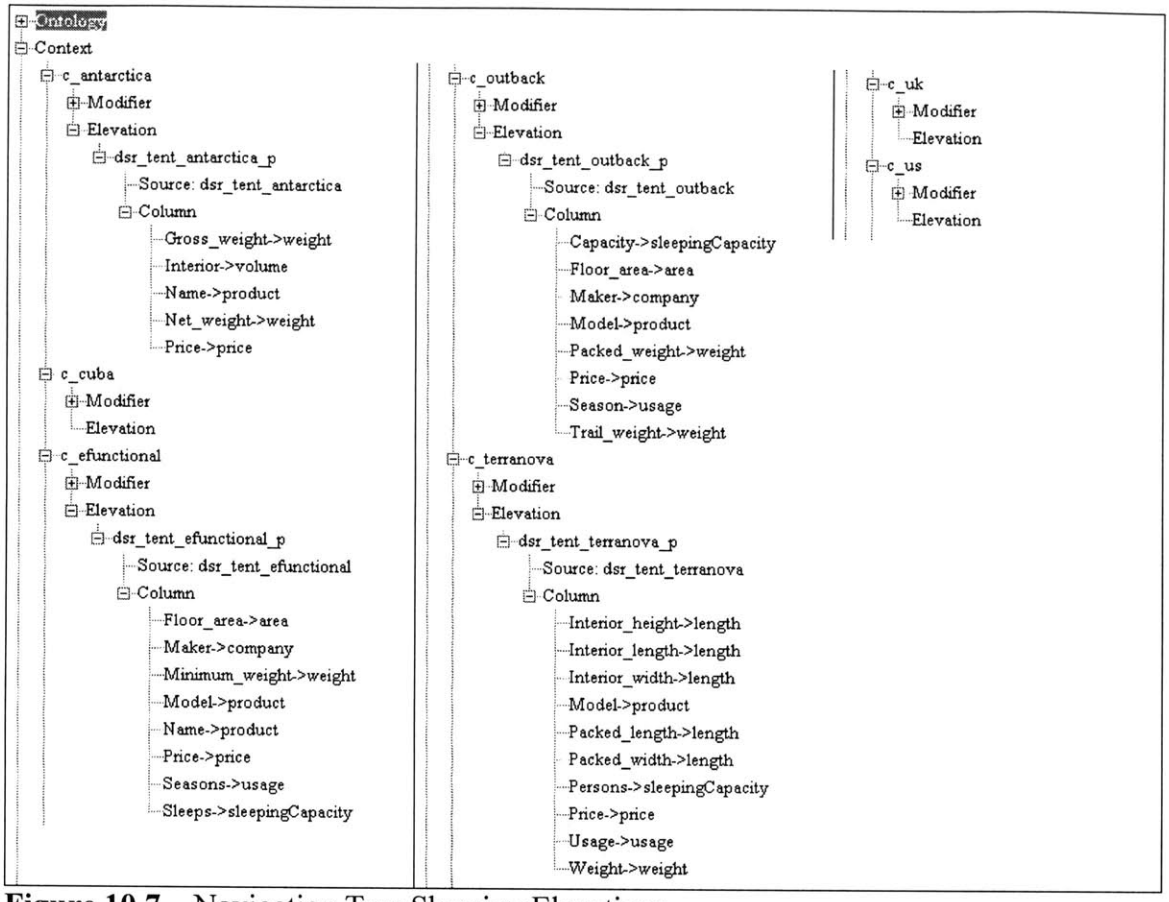


Figure 10.7 – Navigation Tree Showing Elevations

10.5 Performance Analysis

Since the disaster relief example is a slightly more complex example than the tent example seen earlier, it helps to point out certain positive and negative issues about the user interfaces. It is realized that the auto-generated ontological graph becomes increasingly entangled and less readable as the ontology grows. Figure 10.8 shows the auto-generated graph for the disaster relief application. The only remedy to the entanglement of lines in graph is to manually reposition the graph through the exported XML files discussed in Chapter 6. Manually modifying the pixel positions of each element in the graph is a painstaking process. This points to the need for a point-and-click tool for repositioning the ontology. Fortunately, the concept of exporting positional information is ready built into the graphical interface. The remaining work to be done is to find a suitable find a suitable graphical tool to support the point-and-click editing. This topic will be discussed further in the chapter on future work.

As application gets larger, it becomes harder to debug through the Prolog file. This is true especially for spelling errors. A definite advantage of the user interface is that spelling mistakes, like all other metadata, are automatically propagated throughout the entire application. So in effect, it becomes easier to spot. Also, since the user interface is designed not to ask for the same information twice from the user, a lot of tedious work is cut out for the users. It is also discovered in the process of developing the application

that the navigation tree from the graphical user interface is very useful for viewing the entire application at once. This is perhaps due to its ability to hide and show metadata in different depths of details. These evidences point to the conclusion that the user interface does not become handicapped as the size of an application grows.

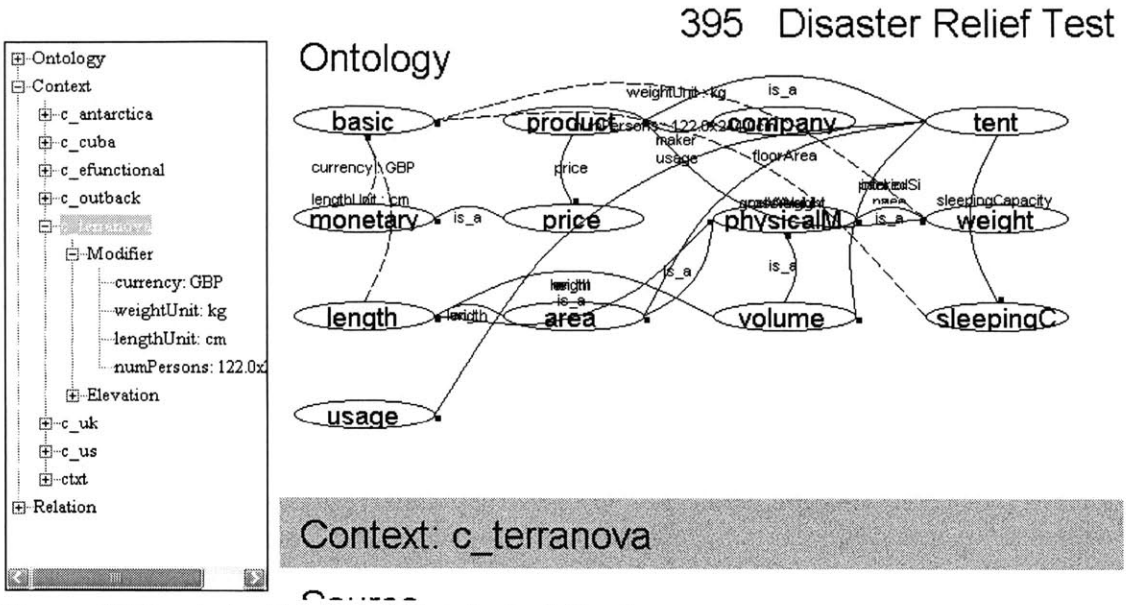


Figure 10.8 – Auto-Generated Ontological Graph

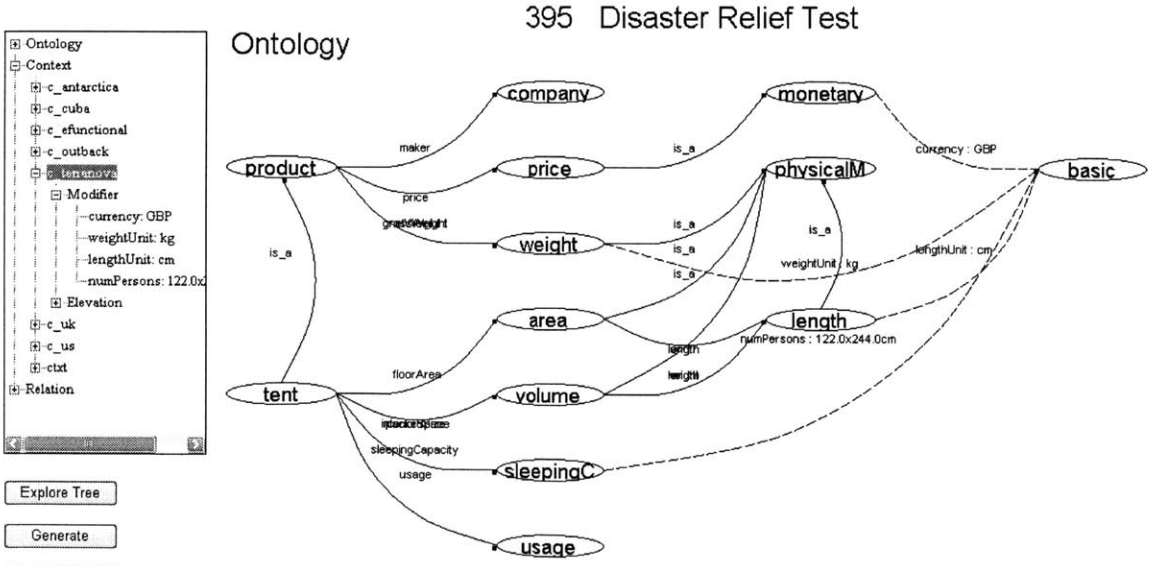


Figure 10.9 – Manually Repositioned Ontological Graph

11 Conclusion and Future Work

The work of this thesis focuses on metadata representation and management. On metadata representation, a suite of XML-based representation of COIN application is fashioned, and an intuitive user interface is created. On metadata management, a registry is set up to provide centralized access of application metadata files. As stated in the introduction chapter, the objectives of this thesis are (1) separation of knowledge and representation, (2) human aspect, and (3) knowledge sharing and reuse. An objective view of the accomplishment of this thesis in terms of these objectives is presented in the following sections. Suggestions are also made in areas where extra work can bring the work started by this thesis closer to the objectives.

11.1 Separation of Knowledge and Representation

Like other information acquisition and management system, the COIN Application Metadata Layer needs to communicate with multiple systems, each with different standard of knowledge representation. Depending on which party the metadata layer is interacting with, the application metadata needs to be packaged differently. In communicating with eCOIN, the metadata must be packaged in Prolog. In interacting with users, the same metadata must be displayed on a set of screens. To add to the complexity, the representation demanded by each party may change over time. But since the metadata layer is designed with separation of knowledge and representation in mind, these requirements are naturally satisfied. Let us take a closer look at how they are satisfied.

The knowledge content of the metadata layer is housed in a data structure known as the Internal COIN Model, or ICM. The ICM is designed purely for use within the metadata layer, and has no built-in mechanism or logic for dealing with representational issues. Therefore, ICM is immune to changes in representation standards in outside sources. Since ICM is a software data structure, it needs to be persisted for permanent storage. To keep in line with isolating knowledge from representation, the exact ICM schema is duplicated onto a RDF schema. This means that every class and property in the ICM has a one-to-one correspondent in RDF. Thus, we are able to maintain the isolation of knowledge in ICM, as well as in its persisted form, the RDF file.

Having achieved knowledge isolation, there still needs to be an easy and effective way of representing the metadata to external parties. There are two classes of consumer for the application metadata; one is the interface user, and the other is non-interface user. Interface users are human users who interact with the metadata on screen. Since the user interfaces are software rendered, they can interact with the ICM through the same software channel. To both facilitate the interaction and maintain the separation of knowledge and representation, an API is developed for interacting with the ICM. If no API were used and the ICM data structure were exposed to the user interfaces, then the user interfaces may develop to rely on the internal representation of ICM. When that happens, the user interfaces become adversely linked to the ICM. This linkage will break the separation of knowledge and representation because the ICM can no longer be redesigned without affecting the user interfaces. Thus, the presence of an API enforces the separation.

Non-interface users are either external system processes that act on the metadata, or human users who prefer not to interact with the user interfaces provided. Since the application metadata stored in ICM is already being persisted in RDF format, it is natural to use XSLT to satisfy the different representation needs of these non-interface users. Recall that XSLT is a language for the transformation of XML documents between different XML-based formats. In using XSLT to encode the transformation needed for various representations, we are able to maintain the separation of knowledge and representation. Simply put, we have provided an efficient mean of manipulating knowledge into different representations; and at the same time, shifted the responsibility of representation to metadata consumers. A few XSLT documents are built with this thesis, and are primarily for demonstrating the capability of transforming knowledge into different representations. These representations include RuleML, RFML, and Prolog (HTML). Notice that the transformation is bound within the XML domain. However, this is hardly a limitation because XML has proven itself to be a very capable language for communicating and storing data. This is apparent in the proliferation of XML documents that appear on the internet in recent years. In real usage, metadata consumers can simply obtain the RDF files containing COIN applications, and then apply their own XSLT to produce their preferred choice of metadata representation.

11.1.1 Conversion Function Builder

The ICM can be improved to better capture conversion function for COIN application. Currently, the ICM stores each conversion function in Prolog without breaking down the conversion function into individual, manageable pieces. Upon initial analysis of the structure of conversion functions, it is realized that each conversion function can be a miniature COIN application of its own. This is so because conversion functions are allowed to draw upon separately defined functions for support during a conversion. These supporting functions can in turn draw upon existing or newly defined semantic types, attributes, relations, and any other elements of COIN application to accomplish its role in the conversion. With this understanding of conversion function, it is perhaps feasible to model each conversion function as a miniature version of COIN application. In doing so, we can reuse much of the infrastructure we have already built for capturing a real COIN application.

Although much more work is needed to fully characterize conversion functions, a minor step that we can take toward this goal is to create a simple conversion function builder. This conversion function builder can operate under the assumptions that most conversions in COIN are purely mathematical in nature. This assumption restricts the definition of conversion functions to simple relationships between operands and mathematical operators. In designing the user interface for such a conversion function builder, much wisdom can be borrowed from the Query Builder [18] of eCOIN. The constraint portion of relational queries and mathematical formulas are alike in that condition of equality is a recurring theme for both.

11.2 Human Aspect

Another objective of this thesis is to bring human aspect into perspective in the design of the user interfaces. This is accomplished by introducing a number of easy-to-use and

highly intuitive features on the user interfaces, which aid the users in building COIN application tremendously. One of these features is the navigation tree that compliments the graphical interface. The navigation tree allows user one-click access to all aspects of a COIN application. Furthermore, the elements of COIN application are displayed in a hierarchical fashion to provide extra insight into the relationships among elements of the application. However, there has not been enough actual users who have interacted with the interfaces long enough to declare victory on this front. A number of possible improvements for the user interfaces are discussed in the following subsections for future work.

11.2.1 Interactive Graphical Image Panel

One way to improve the graphical interface is to have a fully interactive image panel for interpreting user's mouse pointer actions over the ontology graph. Currently, the image panel in the graphical interface is only a static area for displaying information graphically. There is much to be desired in a image panel that can react to user commands. For instance, an interactive panel can allow the user to select and change elements of the COIN application right over the graphical representation. Simple reshaping of the ontological graph to untangle lines can be supported through an interactive panel as well.

11.2.2 Scalar Vector Graphics

Another area of improvement for the graphical interface is in the reshaping of the ontological graph. Currently, the ontological graph is generated using simple positional logic that is not robust enough to avoid crossing lines and overlapping objects. The current interface does allow the positional information of an ontological graph to be exported in XML format; however, this is only a feasible solution but not an optimal one. There needs to be some way for the user to graphically modify the exported information. To build such a graph reshaping interface from scratch requires a lot of effort that are unrelated to context mediation. Therefore, an ideal solution is to employ third party software to do the job for us. This is where Scalar Vector Graphics (SVG) fits into the solution. SVG is a language for describing two-dimensional graphics in XML [19]. There are already graphical viewers and editors developed for SVG. By exporting the positional information in SVG format, we can point the user to such software for full graphical manipulation of the ontological graph. An even better solution is to incorporate one such software into our graphical interface.

11.2.3 COIN Application Metadata in English

To some human users, perhaps a description of a COIN application in English is the most intuitive way of communicating the metadata. A description of metadata in English is currently scattered through out the textual interface. For example, the phrase "tent is a product" is used in the ontology page in describing the existence of semantic types and their inheritances. A great addition to the textual interface is to provide a consolidated view of the application by gathering all the description phrases from the various pages in the textual interface, and produce a consolidated view through a multi-line text box control.

11.3 Knowledge Sharing and Reuse

An important objective of this thesis is to bring about the sharing and reuse of COIN application metadata. This objective is accomplished by a combination of design ingredients introduced in this thesis. The first ingredient is the use of XML-based format for metadata representation. Having COIN application in RDF greatly increases the acceptability of COIN metadata in other domains where similar information is used. A great number of ontology are done in XML languages such as RDF by the modeling community. Through a common XML-based communication channel, existing ontology from outside communities can potentially be imported into COIN, and vice versa. In fact, a research effort here at COIN to investigate the use of XSLT for transforming RDF ontology into COIN ontology is already underway. Contextual information within COIN applications is another potentially exportable resource.

The second ingredient that encourages knowledge sharing and reuse is the file-based registry implemented by this thesis. The registry is a great resource publishing tool that increases the accessibility of COIN metadata for both internal and external use. Furthermore, recall that the registry only stores locations of the files and not the files themselves. This makes it easy for anyone who wants to share their resources via internet, but wishes to maintain the actual content of the resources.

11.3.1 COIN Registry Manager

An area of future work in resource sharing is the need for a user interface for the file-based COIN registry. Currently, the registry spans over a collection of files that are accessible via the internet. Although interacting with these files is not difficult, managing the content of these files from a human user perspective is. As the number of COIN applications that are registered in the registry increases, there needs a better way of browsing the information effectively. A system that manages the registry should allow for online modification of registry content, such as the registration and removal of COIN applications.

11.4 File Concurrency

While much of the benefits of file-based design has been discussed, the danger of such design is rarely mentioned. The inherent danger in a file-based design is that, without some file-locking and concurrency-resolving mechanism, the content of these files is at risk of being corrupted. For instance, when two processes both open the same file to write within the same window of time, the process that finishes the writing last can overwrite the information written by the earlier process. This is indeed a very undesirable situation and is one that must be dealt with swiftly. The only reason that it has not been dealt with in this thesis yet is that the number of users on the system is still low, low enough to warrant that concurrent file access is rare.

12 References

[1] Cheng Hian Goh. Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Systems. PhD dissertation, Massachusetts Institute of Technology, Sloan School of Management, December, 1996.

[2] Stephane Bressan, Cheng Goh, Natalia Levina, Stuart Madnick, Ahmed Shah, and Michael Siegel. Context Knowledge Representation and Reasoning in the Context Interchange System. Applied Intelligence 13, 165-180, 2000.

[3] Cheng Hian Goh, Stephane Bressan, Stuart Madnick, and Michael Siegel. Context Interchange: New Features and Formalisms for the Intelligent Integration of Information. ACM Transactions on Information Systems, Vol. 17, No. 3, July 1999, Pages 270-293.

[4] Tarik Alatovic. Capabilities Aware Planner/Optimizer/Executioner for Context Interchange Project. Master thesis, Massachusetts Institute of Technology, Sloan School of Management, February, 2002.

[5] Usman Y. Mobin. Graphical Metadata Management for the Context Mediation System. Master thesis, Massachusetts Institute of Technology, Sloan School of Management, January, 2002.

[6] Aykut Firat, Stuart Madnick, Michael Siegel. The Cameleon Approach to the Interoperability of Web Sources and Traditional Relational DataBases. Proceedings of the 10th Annual Workshop On Information Technologies and Systems, Brisbane, Queensland, Australia, 2000.

[7] The W3C XML Extensible Markup Language Working Group Home Page, http://www.w3.org/XML/

[8] The World Wide Web Consortium Home Page, http://www.w3.org/

[9] The World Wide Web Consortium Metadata and Resource Description, http://www.w3.org/Metadata/

[10] The W3C RDF Primer, http://www.w3.org/TR/rdf-primer/

[11] The W3C Uniform Resource Identifier (URI) Activity Statement, http://www.w3.org/Addressing/Activity

[12] The RuleML Initiative, http://www.dfki.uni-kl.de/ruleml/

[13] Harold Boley, Said Tabet, Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. Proceedings of the Semantic Web Working Symposium, California, 2001.

[14] Harold Boley. The Relational-Functional Markup Language RFML – Draft Specification. Germany, 2000.

[15] Translators between RuleML and RFML, http://www.relfun.org/ruleml/rfml-ruleml.html

[16] W3C Extensible Stylesheet Language Home Page, http://www.w3.org/Style/XSL/

[17] Leon Sterling. The Art of Prolog: Advanced Programming Techniques. MIT Press, Cambridge, Massachusetts, 1994.

[18] http://interchange.mit.edu:8080/gcms/qb/qbe.jsp

[19] Scalar Vector Graphics homepage, http://www.w3.org/Graphics/SVG/Overview.htm8

[20] Microsoft .NET Framework homepage, http://www.microsoft.com/net/basics/framework.asp

Appendix A – Internal COIN Model API

// Coin Model
// Constructors
public Coin_Model()
// Remarks: Simple constructor for Coin_Model
public Coin_Model(string cid, string cname)
// Parameters: cid - COIN application id, cname - COIN application name
// Remarks: Constructor for Coin_Model with parameters

// Methods
public string GetAppId()
// Return value: COIN application id

public void SetAppId(string aid)
// Parameters: aid - COIN application id

public string GetAppName()
// Return value: COIN application name

public void SetAppName(string aname)
// Parameters: aname - COIN application name

// Ontology
public void AddSemanticType(string semName)
// Parameters: semName - name of semantic type to be added
// Remarks: Add a new semantic type with the supplied name to the COIN application

public void RemoveSemanticType(string semTypeName)
// Parameters: semTypeName - name of semantic type to be removed
// Remarks: Remove a semantic type given a semantic type name

public void AddSemanticTypeInheritance(string childName, string parentName)
// Parameters: childName - name of the child semantic type, parentName - name of the parent semantic type
// Remarks: Add a new inheritance relationship to the COIN application

public void RemoveSemanticTypeInheritance(string childName)
// Parameters: childName - name of the child semantic type in the inheritance relationship
// Remarks: Remove an inheritance relationship given the child semantic type

public void AddSemanticTypeAttribute(string attrName, string semFromName, string semToName)
// Parameters: attrName - name of attribute, semFromName - name of the originating semantic type, semToName - name of the destination semantic type
// Remarks: Add an attribute relationship to the COIN application

public void RemoveAttribute(string semtypeName, string attrName)
// Parameters: semtypeName - name of the originating semantic type of the attribute to be removed, attrName - name of the attribute to be removed
// Remarks: Remove an attribute given the attribute and a semantic type name

public void AddSemanticTypeModifier(string modName, string semFromName, string semToName)
// Parameters: modName - name of modifier , semFromName - name of the originating semantic type, semToName - name of the destination semantic type
// Remarks: Add a modifier relationship to the COIN application

public void RemoveModifier(string semtypeName, string modName)
// Parameters: semtypeName - name of the originating semantic type of the modifier to be removed, modName - name of the modifier
// Remarks: Remove a modifier relationship given the modifier name and a semantic type name

public Ont_SemanticType GetSemanticType(string semtypeName)
// Parameters: semtypeName - the name of the semantic type to retrieve
// Return value: Ont_Semantic type object
// Remarks: Retrieve a semantic type given the name

public IEnumerator GetAllSemanticType()
// Return value: an enumeration of semantic types
// Remarks: Retrieve all semantic types in the COIN application

public IEnumerator GetAllAttribute()
// Return value: an enumeration of attributes
// Remarks: Retrieve all attributes of the COIN application

public IEnumerator GetAllAttribute(string semtypeName)
// Parameters: semtypeName - the name of the originating semantic type
// Return value: an enumeration of Ont_Attribute
// Remarks: Retrieve all attributes originating for a given semantic type

public Ont_Attribute GetAttribute(string semtypeName, string attrName)
// Parameters: semtypeName - name of the attribute to be retrieved, semtypeName - name of the originating semantic type of the attribute
// Return value: Ont_Attribute object
// Remarks: Retrieve an attribute given an attribute name and a semantic type name

public IEnumerator GetAllModifier()
// Return value: an enumeration of Ont_Modifier
// Remarks: Retrieve all modifiers from the COIN application

public IEnumerator GetAllModifier(string semtypeName)
// Parameters: semtypeName - name of the originating semantic type
// Return value: an enumeration of Ont_Modifier
// Remarks: Retrieve all modifiers originating from a given semantic type

public Ont_Modifier GetModifier(string semtypeName, string modName)
// Parameters: semtypeName - name of originating semantic type, modName - name of the modifier
// Return value: Ont_Modifier object
// Remarks: Retrieve a modifier given the modifier and a semantic type

// Context
public void AddContext(string cxtName)
// Parameters: cxtName - the name of the context
// Remarks: Add a context by name

public void AddContextSharing(string childName, string parentName)
// Parameters: childName - name of the new context, parentName - name of the context for sharing
// Remarks: Add a context given a name and the context to be shared

public void RemoveContext(string cxtName)
// Parameters: cxtName - name of the context to be removed
// Remarks: Remove a context given a context name

```
public Cxt_Context GetContext(string cxtName)
// Parameters: cxtName - name of the context to be retrieved
// Return value: Cxt_Context object
// Remarks: Retrieve the context from a given name

public IEnumerator GetAllContext()
// Return value: an enumeration of Cxt_Context
// Remarks: Retrieve all contexts from the COIN application

public void SetModifierValue(string modifiedTypeName, string modifierName, string cxtName, ArrayList
modValueArray)
// Parameters: modifiedTypeName - the name of the semantic type being modified, modifierName - the
name of the modifier,
//          cxtName - the name of the context for which the modifier value is defined
//          modValueArray - an array of Ont_Attribute or string value which represents the value of a
modifier
// Remarks: Set the modifier value for a modifier for a given context. Dynamic modifier value must an
array of Ont_Attribute. Static modifier value must be an array with one string element.

public void RemoveModifierValue(string semtypeName, string modifierName, string contextName)
// Parameters: semtypeName - the name of the semantic type being modified, modifierName - the name of
the modifier, contextName - the name of the context
// Remarks: Remove the modifier value from a modifier for a given context

public ArrayList GetModifierValue(string semtypeName, string modName, string cxtName)
// Parameters: semtypeName - the name of the originating semantic type, modName - the name of the
modifier, cxtName - the name of the context
// Return value: the modifier value in a array of Ont_Attribute or string
// Remarks: Retrieve the modifier value for a given context. Dynamic modifier value must an array of
Ont_Attribute. Static modifier value must be an array with one string element.

public IEnumerator GetAllModifierByContext(string cxtName)
// Parameters: cxtName - name of the context
// Return value: an enumeration of Ont_Modifier
// Remarks: Retrieve all modifiers with values defined for a given context

public void SetConversionFunction(string modifiedTypeName, string modifierName, string convfunc)
// Parameters: modifiedTypeName - the name of the semantic type being modified, modifierName - the
name of the modifier, convfunc - the conversion function as a string
// Remarks: Set the conversion function for a given modifier. The Conversion function string can be either
a Prolog rule, or the name of a function in the function library.

public void SetRelationElevation(string relationName, string contextName)
// Parameters: reltaionName - name of the relation, contextName - name of the context
// Remarks: Elevate a relation for a given context. The elevated relation will have the relation name with
"_p" attached at the end.

public void RemoveRelationElevation(string relationName, string contextName)
// Parameters: reltaionName - name of the relation, contextName - name of the context
// Remarks: Remove the elevated realtion for a given context

public Src_ElevatedRelation GetElevatedRelation(string relationName, string contextName)
// Parameters: reltaionName - name of the relation, contextName - name of the context
// Return value: Src_ElevatedRelation object
// Remarks: Retrieve the elevated relation for a given context
```

public Src_ElevatedRelation GetElevatedRelation(string relationName)
// Parameters: relationName - name of the relation
// Return value: Src_ElevatedRelation object
// Remarks: Retrieve the elevated relation for a given relation, assuming that the relation has only been elevated under one context. If the relation is elevated over multiple context, then an elevated relation from one of those contexts will be returned.

public void SetColumnElevation(string relationName, string columnName, string semtypeName)
// Parameters: relationName - name of the relation, columnName - name of the column, semtypeName - name of the semantic type
// Remarks: Elevate the column to a semantic type for a given column in a relation. This assumes that the relation has been elevated under one context only.

public void SetColumnElevation(string relationName, string columnName, string semtypeName, string contextName)
// Parameters: relationName - name of the relation, columnName - name of the column, semtypeName - name of the semantic type, contextName - name of the context
// Remarks: Elevate the column to a semantic type under a given context for a given column in a relation.

public void RemoveColumnElevation(string relationName, string columnName, string contextName)
// Parameters: relationName - name of the relation, columnName - name of the column, contextName - name of the context
// Remarks: Remove the column elevation for a given column in a relation under a given context.

public void AddElevatedRelation(string erelName, string relationName, string contextName, bool noContext)
// Parameters: erelName - name of the elevated relation, relationName - name of the relation, contextName - name of the context, noContext - true if no specific context is associated with the elevation
// Remarks: Elevates a relation for a given context. If noContext is true, contextName can be null.

public void AddElevatedRelation(string relationName, string contextName, bool noContext)
// Parameters: relationName - name of the relation, contextName - name of the context, noContext - true if no specific context is associated with the elevation
// Remarks: Elevates a relation for a given context where the default elevation name is assigned. The elevated relation will have the relation name with "_p" attached at the end. If noContext is true, contextName can be null.

// Source
public void AddRelation(string relationName, bool import, bool export, string sourceName, string unsupportedOps)
// Parameters: relationName - name of the relation, import - true if relation is importable, export - true if relation is exportable, sourceName - name of the source, unsupportedOps - comma delimited list of unsupported operation.
// Remarks: Add a relation with the specified attributes. In the unsupportedOps string, the symbols <,>,=,<=,=> must be entered as lt,gt,et,le,ge respectively.

public void RemoveRelation(string relName)
// Parameters: relName - name of the relation
// Remarks: Remove the relation with the given name

public void AddColumn(string columnName, string relationName, string columnType, bool isKey)
// Parameters: columnName - name of the column, relationName - name of the relation, columnType - column type, isKey - true if column is part of key
// Remarks: Add a column to the relation with the supplied column attributes. columnType is commonly "string" or "number" depending on the data source

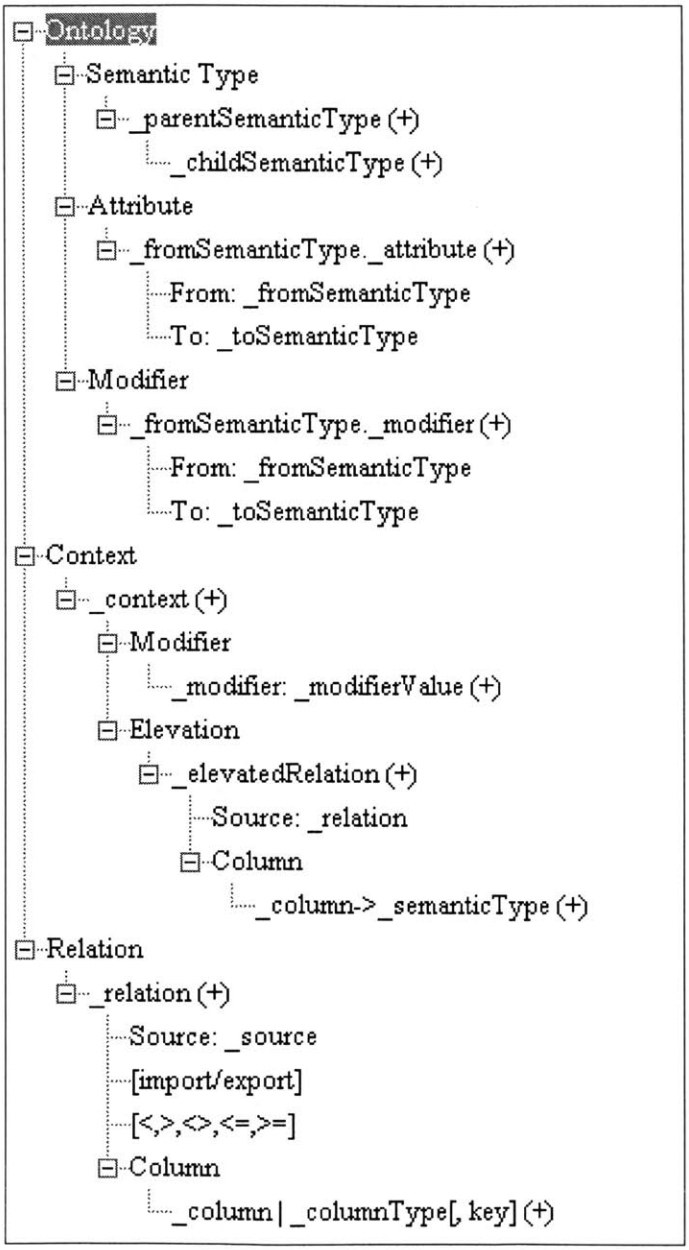
```
public void RemoveColumn(string relationName, string columnName)
// Parameters: relationName - name of the relation, columnName - name of the column
// Remarks: Remove the column given a column name and a relation name

public Src_Relation GetRelation(string relName)
// Parameters: relName - name of relation
// Return value: Src_Relation object
// Remarks: Retrieve a relation given the relation name

public IEnumerator GetAllRelation()
// Return value: an enumeration of Src_Relation object in the COIN application
// Remarks: Retrieve all relations in the COIN application
```

Appendix B – Navigation Tree Template

Legends
_x : name or string representation of the object
(+) : multiple objects allowed
[x] : optional value
Everything else is invariable text.



Appendix C – Graphical Layout Scheme File

```
<?xml version="1.0" encoding="utf-8"?>
<EntityPosition
xmlns:xsd="http://www.w3.org/2001/XMLSche
ma"
xmlns:xsi="http://www.w3.org/2001/XMLSche
ma-instance">
 <appid>392</appid>
 <appname>Simple Disaster Relief</appname>
 <width>1000</width>
 <height>2000</height>
 <offsetX>0</offsetX>
 <offsetY>25</offsetY>
 <semtypeWidth>100</semtypeWidth>
 <semtypeHeight>20</semtypeHeight>
 <semtypeStartX>10</semtypeStartX>
 <semtypeStartY>60</semtypeStartY>
 <semtypeSpace>50</semtypeSpace>
 <ontologyWidth>600</ontologyWidth>
 <ontologyHeight>225</ontologyHeight>
 <ontArrowFontSize>8</ontArrowFontSize>
 <titleOffsetX>10</titleOffsetX>
 <titleOffsetY>10</titleOffsetY>
 <titleWidth>500</titleWidth>
 <titleHeight>35</titleHeight>
 <titleFontSize>20</titleFontSize>
 <contextStartX>0</contextStartX>
 <contextStartY>225</contextStartY>
 <contextWidth>600</contextWidth>
 <contextHeight>50</contextHeight>
 <sourceStartX>0</sourceStartX>
 <sourceStartY>275</sourceStartY>
 <sourceWidth>600</sourceWidth>
 <sourceHeight>600</sourceHeight>
 <relStartX>10</relStartX>
 <relStartY>60</relStartY>
 <relColWidth>50</relColWidth>
 <relColHeight>15</relColHeight>
 <relSpace>50</relSpace>

<displayBasicInheritance>false</displayBasicIn
heritance>
 <semanticTypes>
  <Ep_SemanticType>
   <name>product</name>
   <x>10</x>
   <y>85</y>
   <selected>false</selected>
  </Ep_SemanticType>
  <Ep_SemanticType>
   <name>basic</name>
   <x>160</x>
   <y>85</y>
   <selected>false</selected>
  </Ep_SemanticType>
  <Ep_SemanticType>
   <name>tent</name>
   <x>310</x>
   <y>85</y>
   <selected>false</selected>
  </Ep_SemanticType>
  <Ep_SemanticType>
   <name>weight</name>
   <x>460</x>
   <y>85</y>
   <selected>false</selected>
  </Ep_SemanticType>
 </semanticTypes>
 <attributes>
  <Ep_Attribute>
   <name>weight</name>
   <fromSem>
    <name>product</name>
    <x>10</x>
    <y>85</y>
    <selected>false</selected>
   </fromSem>
   <xfrom>110</xfrom>
   <yfrom>95</yfrom>
   <xto>460</xto>
   <yto>95</yto>
   <spline>
    <PointF>
     <X>110</X>
     <Y>95</Y>
    </PointF>
    <PointF>
     <X>226.666672</X>
     <Y>145</Y>
    </PointF>
    <PointF>
     <X>343.333344</X>
     <Y>145</Y>
    </PointF>
    <PointF>
     <X>460</X>
     <Y>95</Y>
    </PointF>
   </spline>
   <selected>false</selected>
  </Ep_Attribute>
 </attributes>
 <modifiers>
  <Ep_Modifier>
   <name>weightUnit</name>
   <fromSem>
    <name>weight</name>
    <x>460</x>
    <y>85</y>
```

```
    <selected>false</selected>
   </fromSem>
   <xfrom>460</xfrom>
   <yfrom>95</yfrom>
   <xto>260</xto>
   <yto>95</yto>
   <spline>
    <PointF>
     <X>460</X>
     <Y>95</Y>
    </PointF>
    <PointF>
     <X>393.333344</X>
     <Y>66.42857</Y>
    </PointF>
    <PointF>
     <X>326.666656</X>
     <Y>66.42857</Y>
    </PointF>
    <PointF>
     <X>260</X>
     <Y>95</Y>
    </PointF>
   </spline>
   <selected>false</selected>
  </Ep_Modifier>
</modifiers>
<inheritances>
  <Ep_Inheritance>
   <name>is_a</name>
   <xfrom>110</xfrom>
   <yfrom>95</yfrom>
   <xto>160</xto>
   <yto>95</yto>
   <spline>
    <PointF>
     <X>110</X>
     <Y>95</Y>
    </PointF>
    <PointF>
     <X>126.666664</X>
     <Y>102.14286</Y>
    </PointF>
    <PointF>
     <X>143.333328</X>
     <Y>102.14286</Y>
    </PointF>
    <PointF>
     <X>160</X>
     <Y>95</Y>
    </PointF>
   </spline>
   <selected>false</selected>
   <isBasic>true</isBasic>
  </Ep_Inheritance>
  <Ep_Inheritance>
   <name>is_a</name>
   <xfrom>310</xfrom>
   <yfrom>95</yfrom>
   <xto>110</xto>
   <yto>95</yto>
   <spline>
    <PointF>
     <X>310</X>
     <Y>95</Y>
    </PointF>
    <PointF>
     <X>243.333328</X>
     <Y>66.42857</Y>
    </PointF>
    <PointF>
     <X>176.666672</X>
     <Y>66.42857</Y>
    </PointF>
    <PointF>
     <X>110</X>
     <Y>95</Y>
    </PointF>
   </spline>
   <selected>false</selected>
   <isBasic>false</isBasic>
  </Ep_Inheritance>
  <Ep_Inheritance>
   <name>is_a</name>
   <xfrom>460</xfrom>
   <yfrom>95</yfrom>
   <xto>260</xto>
   <yto>95</yto>
   <spline>
    <PointF>
     <X>460</X>
     <Y>95</Y>
    </PointF>
    <PointF>
     <X>393.333344</X>
     <Y>66.42857</Y>
    </PointF>
    <PointF>
     <X>326.666656</X>
     <Y>66.42857</Y>
    </PointF>
    <PointF>
     <X>260</X>
     <Y>95</Y>
    </PointF>
   </spline>
   <selected>false</selected>
   <isBasic>true</isBasic>
  </Ep_Inheritance>
</inheritances>
<context>
  <selected>false</selected>
```

```
</context>
<relations>
 <Ep_Relation>
  <name>dsr_tent_terranova</name>
  <desc>oracle | ie</desc>
  <x>10</x>
  <y>335</y>
  <selected>false</selected>
  <nameCells>
   <Ep_RelationCell>
    <name>Maker</name>
    <x>10</x>
    <y>380</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Model</name>
    <x>10</x>
    <y>395</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Name</name>
    <x>10</x>
    <y>410</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Seasons</name>
    <x>10</x>
    <y>425</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Sleeps</name>
    <x>10</x>
    <y>440</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Minimum_weight</name>
    <x>10</x>
    <y>455</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Floor_area</name>
    <x>10</x>
    <y>470</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>Price</name>
    <x>10</x>
    <y>485</y>
    <selected>false</selected>
   </Ep_RelationCell>
  </nameCells>
  <typeCells>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>380</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>395</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>410</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>425</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>440</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>455</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>470</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>string</name>
    <x>160</x>
    <y>485</y>
    <selected>false</selected>
   </Ep_RelationCell>
  </typeCells>
  <keyCells>
   <Ep_RelationCell>
    <name />
    <x>60</x>
```

```
    <y>380</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name>key</name>
    <x>60</x>
    <y>395</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name />
    <x>60</x>
    <y>410</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name />
    <x>60</x>
    <y>425</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name />
    <x>60</x>
    <y>440</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name />
    <x>60</x>
    <y>455</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name />
    <x>60</x>
    <y>470</y>
    <selected>false</selected>
   </Ep_RelationCell>
   <Ep_RelationCell>
    <name />
    <x>60</x>
    <y>485</y>
    <selected>false</selected>
   </Ep_RelationCell>
  </keyCells>
  <erels>
   <Ep_ElevatedRelation>
    <name>dsr_tent_terranova_p</name>
    <desc>c_uk</desc>
    <x>310</x>
    <y>335</y>
    <selected>false</selected>
    <display>true</display>
    <semCells>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>380</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>395</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>410</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>425</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>440</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>weight</name>
      <x>310</x>
      <y>455</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>470</y>
      <selected>false</selected>
     </Ep_RelationCell>
     <Ep_RelationCell>
      <name>basic</name>
      <x>310</x>
      <y>485</y>
      <selected>false</selected>
     </Ep_RelationCell>
    </semCells>
   </Ep_ElevatedRelation>
  </erels>
 </Ep_Relation>
</relations>
</EntityPosition>
```

Appendix D – COIN Registry DTDs

registries.dtd

```
<!ELEMENT REGISTRIES (REGISTRY*)>
<!ELEMENT REGISTRY (APPLICAITONS, SOURCES)>
<!ATTLIST REGISTRY name CDATA #REQUIRED>
<!ELEMENT APPLICATIONS (#PCDATA)>
<!ELEMENT SOURCES (#PCDATA)>
```

applications.dtd

```
<!ELEMENT APPLICATIONS (APP*)>
<!ELEMENT APP (RDF*, RULEML*, RFML*, PROLOGHTML*, PROLOG*, SCHEMA*)>
<!ATTLIST APP id CDATA #REQUIRED name CDATA #REQUIRED>
<!ELEMENT RDF (#PCDATA)>
<!ELEMENT RULEML (#PCDATA)>
<!ELEMENT RFML (#PCDATA)>
<!ELEMENT PROLOGHTML (#PCDATA)>
<!ELEMENT PROLOG (#PCDATA)>
<!ELEMENT SCHEMA (#PCDATA)>
```

sources.dtd

```
<!ELEMENT SOURCES (SOURCE*)>
<!ELEMENT SOURCE (URL, USERNAME?, PASSWORD?, MAXCONNECTION)>
<!ATTLIST SOURCE name CDATA #REQUIRED type CDATA #REQUIRED>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT USERNAME (#PCDATA)>
<!ELEMENT PASSWORD (#PCDATA)>
<!ELEMENT MAXCONNECTION (#PCDATA)>
```

schema.dtd

```
<!ELEMENT SCHEMA (SOURCE?, RELATION*, CONTEXT*)>
<!ELEMENT SOURCE (URL, USERNAME?, PASSWORD?, MAXCONNECTION)>
<!ATTLIST SOURCE name CDATA #REQUIRED type CDATA #REQUIRED>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT USERNAME (#PCDATA)>
<!ELEMENT PASSWORD (#PCDATA)>
<!ELEMENT MAXCONNECTION (#PCDATA)>
<!ELEMENT RELATION (SOURCENAME, ATTRIBUTE*, CAPRECORD)>
<!ATTLIST RELATION name CDATA #REQUIRED>
<!ELEMENT SOURCENAME (#PCDATA)>
<!ELEMENT ATTRIBUTE (#PCDATA)>
<!ATTLIST ATTRIBUTE type CDATA #REQUIRED>
<!ELEMENT CAPRECORD (BOUND, UNOPS)>
<!ELEMENT BOUND (#PCDATA)>
<!ELEMENT UNOPS (#PCDATA)>
<!ELEMENT CONTEXT (DESC?)>
<!ATTLIST CONTEXT name CDATA #REQUIRED>
<!ELEMENT DESC (#PCDATA)>
```

queries.dtd

```
<!ELEMENT QUERIES (QUERY*)>
<!ELEMENT QUERY (SQL, CONTEXT, DESC)>
<!ATTLIST QUERY name CDATA #REQUIRED>
<!ELEMENT SQL (#PCDATA)>
<!ELEMENT CONTEXT (#PCDATA)>
<!ELEMENT DESC (#PCDATA)>
```

Appendix E – COIN Application RDF Schema

```
<?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#">

<!-- Application -->
 <rdfs:Class rdf:ID="Application">
 </rdfs:Class>
 <rdf:Property rdf:ID="ApplicationOntology">
  <rdfs:domain rdf:resource="#Application"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
 <rdf:Property rdf:ID="ApplicationContext">
  <rdfs:domain rdf:resource="#Application"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
 <rdf:Property rdf:ID="ApplicationSource">
  <rdfs:domain rdf:resource="#Application"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
 <rdf:Property rdf:ID="ApplicationMisc">
  <rdfs:domain rdf:resource="#Application"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>

<!-- Semantic Type -->
 <rdfs:Class rdf:ID="Ont_SemanticType">
 </rdfs:Class>
 <rdf:Property rdf:ID="Ont_SemanticTypeName">
  <rdfs:domain rdf:resource="#Ont_SemanticType"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
 <rdf:Property rdf:ID="Ont_SemanticTypeParent">
  <rdfs:domain rdf:resource="#Ont_SemanticType"/>
  <rdfs:range rdf:resource="#Ont_SemanticType"/>
 </rdf:Property>

<!-- Attribute -->
 <rdfs:Class rdf:ID="Ont_Attribute">
 </rdfs:Class>
 <rdf:Property rdf:ID="Ont_AttributeName">
  <rdfs:domain rdf:resource="#Ont_Attribute"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>
 <rdf:Property rdf:ID="Ont_AttributeFrom">
  <rdfs:domain rdf:resource="#Ont_Attribute"/>
  <rdfs:range rdf:resource="#Ont_SemanticType"/>
 </rdf:Property>
 <rdf:Property rdf:ID="Ont_AttributeTo">
  <rdfs:domain rdf:resource="#Ont_Attribute"/>
  <rdfs:range rdf:resource="#Ont_SemanticType"/>
 </rdf:Property>
 <rdf:Property rdf:ID="Ont_AttributeElevationFunction">
  <rdfs:domain rdf:resource="#Ont_Attribute"/>
```

```
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>

 <!-- Modifier -->
  <rdfs:Class rdf:ID="Ont_Modifier">
  </rdfs:Class>
  <rdf:Property rdf:ID="Ont_ModifierName">
    <rdfs:domain rdf:resource="#Ont_Modifier"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Ont_ModifierFrom">
    <rdfs:domain rdf:resource="#Ont_Modifier"/>
    <rdfs:range rdf:resource="#Ont_SemanticType"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Ont_ModifierTo">
    <rdfs:domain rdf:resource="#Ont_Modifier"/>
    <rdfs:range rdf:resource="#Ont_SemanticType"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Ont_ModifierConversionFunction">
    <rdfs:domain rdf:resource="#Ont_Modifier"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Ont_ModifierContextValues">
    <rdfs:domain rdf:resource="#Ont_Modifier"/>
    <rdfs:range rdf:resource="#Ont_ModifierContextValuePair"/>
  </rdf:Property>
  <rdfs:Class rdf:ID="Ont_ModifierContextValuePair">
  </rdfs:Class>
  <rdf:Property rdf:ID="Ont_ModifierContext">
    <rdfs:domain rdf:resource="#Ont_ModifierContextValuePair"/>
    <rdfs:range rdf:resource="#Cxt_Context"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Ont_ModifierStaticValue">
    <rdfs:domain rdf:resource="#Ont_ModifierContextValuePair"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Ont_ModifierDynamicValue">
    <rdfs:domain rdf:resource="#Ont_ModifierContextValuePair"/>
    <rdfs:range rdf:resource="#Ont_Attribute"/>
  </rdf:Property>

 <!-- Context -->
  <rdfs:Class rdf:ID="Cxt_Context">
  </rdfs:Class>
  <rdf:Property rdf:ID="Cxt_ContextName">
    <rdfs:domain rdf:resource="#Cxt_Context"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </rdf:Property>
  <rdf:Property rdf:ID="Cxt_ContextParent">
    <rdfs:domain rdf:resource="#Cxt_Context"/>
    <rdfs:range rdf:resource="#Cxt_Context"/>
  </rdf:Property>

 <!-- Relation -->
  <rdfs:Class rdf:ID="Src_Relation">
  </rdfs:Class>
```

```
<rdf:Property rdf:ID="Src_RelationName">
  <rdfs:domain rdf:resource="#Src_Relation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_RelationImport">
  <rdfs:domain rdf:resource="#Src_Relation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_RelationExport">
  <rdfs:domain rdf:resource="#Src_Relation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_RelationSourceName">
  <rdfs:domain rdf:resource="#Src_Relation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_RelationUnsupportedOps">
  <rdfs:domain rdf:resource="#Src_Relation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>

<!-- Column -->
<rdfs:Class rdf:ID="Src_Column">
</rdfs:Class>
<rdf:Property rdf:ID="Src_ColumnName">
  <rdfs:domain rdf:resource="#Src_Column"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_ColumnType">
  <rdfs:domain rdf:resource="#Src_Column"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_ColumnKeyMember">
  <rdfs:domain rdf:resource="#Src_Column"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_ColumnRelation">
  <rdfs:domain rdf:resource="#Src_Column"/>
  <rdfs:range rdf:resource="#Src_Relation"/>
</rdf:Property>

<!-- Elevated Relation -->
<rdfs:Class rdf:ID="Src_ElevatedRelation">
</rdfs:Class>
<rdf:Property rdf:ID="Src_ElevatedRelationName">
  <rdfs:domain rdf:resource="#Src_ElevatedRelation"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_ElevatedRelationRelation">
  <rdfs:domain rdf:resource="#Src_ElevatedRelation"/>
  <rdfs:range rdf:resource="#Src_Relation"/>
</rdf:Property>
<rdf:Property rdf:ID="Src_ElevatedRelationContext">
  <rdfs:domain rdf:resource="#Src_ElevatedRelation"/>
  <rdfs:range rdf:resource="#Cxt_Context"/>
</rdf:Property>
```

```
 <rdf:Property rdf:ID="Src_ElevatedRelationColumns">
  <rdfs:domain rdf:resource="#Src_ElevatedRelation"/>
  <rdfs:range rdf:resource="#Src_ElevatedRelationColumnSemanticTypePair"/>
 </rdf:Property>
 <rdfs:Class rdf:ID="Src_ElevatedRelationColumnSemanticTypePair">
 </rdfs:Class>
 <rdf:Property rdf:ID="Src_ElevatedRelationColumn">
  <rdfs:domain rdf:resource="#Src_ElevatedRelationColumnSemanticTypePair"/>
  <rdfs:range rdf:resource="#Src_Column"/>
 </rdf:Property>
 <rdf:Property rdf:ID="Src_ElevatedRelationSemanticType">
  <rdfs:domain rdf:resource="#Src_ElevatedRelationColumnSemanticTypePair"/>
  <rdfs:range rdf:resource="#Ont_SemanticType"/>
 </rdf:Property>

<!-- Helper Function-->
 <rdfs:Class rdf:ID="Misc_HelperFunction">
 </rdfs:Class>
 <rdf:Property rdf:ID="Misc_HelperFunctionBody">
  <rdfs:domain rdf:resource="#Misc_HelperFunction"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
 </rdf:Property>

</rdf:RDF>
```

Appendix F – XSLT

F.1 XSLT for RDF -> RuleML

```
<?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#"
 xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#"
 exclude-result-prefixes="rdf coin"
 version="1.0">
<xsl:output
 omit-xml-declaration="no"
 method="xml" indent="yes" version="1.0" encoding="ISO-8859-1"/>
<?cocoon-process type="xslt"?>

<xsl:variable name="ontology"
select="document(//coin:Application/coin:ApplicationOntology/@rdf:resource)"/>
<xsl:variable name="context"
select="document(//coin:Application/coin:ApplicationContext/@rdf:resource)"/>
<xsl:variable name="source"
select="document(//coin:Application/coin:ApplicationSource/@rdf:resource)"/>
<xsl:variable name="misc" select="document(//coin:Application/coin:ApplicationMisc/@rdf:resource)"/>

<!-- Application -->
<xsl:template match="coin:Application">
 <rulebase>
  <xsl:apply-templates select="$ontology/*/coin:Ont_SemanticType" mode="semanticType"/>
  <xsl:apply-templates select="$ontology/*/coin:Ont_SemanticType" mode="attribute"/>
  <xsl:apply-templates select="$ontology/*/coin:Ont_SemanticType" mode="modifier"/>
  <xsl:apply-templates select="$source/*/coin:Src_Relation" mode="relation"/>
  <xsl:apply-templates select="$source/*/coin:Src_ElevatedRelation" mode="elevatedRelation"/>
  <xsl:apply-templates select="$context/*/coin:Cxt_Context" mode="context"/>
  <xsl:apply-templates select="$ontology/*/coin:Ont_Modifier" mode="modifier"/>
  <xsl:apply-templates select="$ontology/*/coin:Ont_Attribute" mode="attribute"/>
  <xsl:apply-templates select="$ontology/*/coin:Ont_Modifier" mode="convfunc"/>
  <xsl:apply-templates select="$misc/*/coin:Misc_HelperFunction" mode="helpfunc"/>
 </rulebase>
</xsl:template>

<!-- Semantic Types -->
<xsl:template match="coin:Ont_SemanticType" mode="semanticType">
 <imp>
  <_head><atom><_opr><rel>is_a</rel></_opr>
   <xsl:apply-templates select="coin:Ont_SemanticTypeName"/>
   <xsl:choose>
    <xsl:when test="coin:Ont_SemanticTypeParent"><xsl:apply-templates
select="coin:Ont_SemanticTypeParent"/></xsl:when>
    <xsl:otherwise><ind>basic</ind></xsl:otherwise>
   </xsl:choose>
   </atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
</xsl:template>
```

```
<xsl:template match="coin:Ont_SemanticTypeName">
 <ind><xsl:value-of select="."/></ind>
</xsl:template>

<xsl:template match="coin:Ont_SemanticTypeParent">
 <ind>
  <xsl:apply-templates select="//coin:Ont_SemanticType" mode="matchSemTypeID">
   <xsl:with-param name="semTypeID" select="substring-after(@rdf:resource, '#')"/>
  </xsl:apply-templates>
 </ind>
</xsl:template>

<xsl:template match="coin:Ont_SemanticType" mode="matchSemTypeID">
 <xsl:param name="semTypeID"/>
 <xsl:if test="@rdf:ID=$semTypeID">
  <xsl:value-of select="coin:Ont_SemanticTypeName"/>
 </xsl:if>
</xsl:template>

<!-- Attributes -->
<xsl:template match="coin:Ont_SemanticType" mode="attribute">
 <imp>
  <_head><atom><_opr><rel>attributes</rel></_opr>
   <ind><xsl:value-of select="coin:Ont_SemanticTypeName"/></ind>
   <cterm><_opc><ctor/></_opc>
    <xsl:apply-templates select="//coin:Ont_Attribute" mode="matchSemType">
     <xsl:with-param name="semTypeID" select="@rdf:ID"/>
    </xsl:apply-templates>
   </cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
</xsl:template>

<xsl:template match="coin:Ont_Attribute" mode="matchSemType">
 <xsl:param name="semTypeID"/>
 <xsl:if test="substring-after(coin:Ont_AttributeFrom/@rdf:resource, '#')=$semTypeID">
  <ind><xsl:value-of select="coin:Ont_AttributeName"/></ind>
 </xsl:if>
</xsl:template>

<!-- Modifiers-->
<xsl:template match="coin:Ont_SemanticType" mode="modifier">
 <imp>
  <_head><atom><_opr><rel>modifiers</rel></_opr>
   <ind><xsl:value-of select="coin:Ont_SemanticTypeName"/></ind>
   <cterm><_opc><ctor/></_opc>
    <xsl:apply-templates select="//coin:Ont_Modifier" mode="matchSemType">
     <xsl:with-param name="semTypeID" select="@rdf:ID"/>
    </xsl:apply-templates>
   </cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
</xsl:template>

<xsl:template match="coin:Ont_Modifier" mode="matchSemType">
 <xsl:param name="semTypeID"/>
```

```
  <xsl:if test="substring-after(coin:Ont_ModifierFrom/@rdf:resource, '#')=$semTypeID">
    <ind><xsl:value-of select="coin:Ont_ModifierName"/></ind>
  </xsl:if>
</xsl:template>

<!-- Relations-->
<xsl:template match="coin:Src_Relation" mode="relation">
  <imp>
    <_head><atom><_opr><rel>relation</rel></_opr>
      <ind><xsl:value-of select="coin:Src_RelationSourceName"/></ind>
      <ind><xsl:value-of select="coin:Src_RelationName"/></ind>
      <xsl:variable name="import" select="coin:Src_RelationImport"/>
      <xsl:variable name="export" select="coin:Src_RelationExport"/>
      <xsl:choose>
        <xsl:when test="$import='true' and $export='true'"><ind>ie</ind></xsl:when>
        <xsl:when test="$import='true'"><ind>i</ind></xsl:when>
        <xsl:otherwise><ind>e</ind></xsl:otherwise>
      </xsl:choose>
      <cterm><_opc><ctor/></_opc>
        <xsl:apply-templates select="//coin:Src_Column" mode="column">
          <xsl:with-param name="relationID" select="@rdf:ID"/>
        </xsl:apply-templates>
      </cterm>
      <atom><_opr><rel>cap</rel></_opr>
        <cterm><_opc><ctor/></_opc>
          <cterm><_opc><ctor/></_opc>
            <xsl:apply-templates select="//coin:Src_Column" mode="cap">
              <xsl:with-param name="relationID" select="@rdf:ID"/>
            </xsl:apply-templates>
          </cterm>
        </cterm>
        <cterm><_opc><ctor/></_opc>
          <ind><xsl:value-of select="coin:Src_RelationUnsupportedOps"/></ind>
        </cterm>
      </atom>
    </atom></_head>
    <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
  </imp>
</xsl:template>

<xsl:template match="coin:Src_Column" mode="column">
  <xsl:param name="relationID"/>
  <xsl:if test="substring-after(coin:Src_ColumnRelation/@rdf:resource, '#')=$relationID">
    <cterm><_opc><ctor/></_opc>
      <ind>'<xsl:value-of select="coin:Src_ColumnName"/>'</ind>
      <ind><xsl:value-of select="coin:Src_ColumnType"/></ind>
    </cterm>
  </xsl:if>
</xsl:template>

<xsl:template match="coin:Src_Column" mode="cap">
  <xsl:param name="relationID"/>
  <xsl:if test="substring-after(coin:Src_ColumnRelation/@rdf:resource, '#')=$relationID">
    <xsl:choose>
      <xsl:when test="coin:Src_ColumnKeyMember='true'"><ind>1</ind></xsl:when>
      <xsl:otherwise><ind>0</ind></xsl:otherwise>
```

```
    </xsl:choose>
  </xsl:if>
</xsl:template>

<!-- Elevations-->
<xsl:template match="coin:Src_ElevatedRelation" mode="elevatedRelation">
  <imp>
    <_head><atom><_opr><rel><xsl:value-of select="coin:Src_ElevatedRelationName"/></rel></_opr>
      <xsl:apply-templates select="." mode="skolemStr"/>
    </atom></_head>
    <_body><atom><_opr><rel/></_opr>
      <xsl:apply-templates select="." mode="relationStr"/>
    </atom></_body>
  </imp>
</xsl:template>

<xsl:template match="coin:Src_ElevatedRelation" mode="relationStr">
  <xsl:variable name="relationID" select="substring-
after(coin:Src_ElevatedRelationRelation/@rdf:resource,'#')"/>
  <xsl:variable name="relationName"
select="//coin:Src_Relation[@rdf:ID=$relationID]/coin:Src_RelationName"/>
  <atom><_opr><rel><xsl:value-of select="$relationName"/></rel></_opr>
  <xsl:apply-templates select="coin:Src_ElevatedRelationColumns" mode="relationStr"/></atom>
</xsl:template>

<xsl:template match="coin:Src_ElevatedRelationColumns" mode="relationStr">
  <xsl:variable name="pairID" select="substring-after(@rdf:resource,'#')"/>
  <xsl:variable name="colID" select="substring-
after(//coin:Src_ElevatedRelationColumnSemanticTypePair[@rdf:ID=$pairID]/coin:Src_ElevatedRelation
Column/@rdf:resource,'#')"/>
  <var><xsl:value-of select="//coin:Src_Column[@rdf:ID=$colID]/coin:Src_ColumnName"/></var>
</xsl:template>

<xsl:template match="coin:Src_ElevatedRelation" mode="skolemStr">
  <xsl:variable name="cxtID" select="substring-
after(coin:Src_ElevatedRelationContext/@rdf:resource,'#')"/>
  <xsl:variable name="cxtName"
select="$context/*/coin:Cxt_Context[@rdf:ID=$cxtID]/coin:Cxt_ContextName"/>
  <xsl:apply-templates select="coin:Src_ElevatedRelationColumns" mode="skolemStr"><xsl:with-param
name="cxtName" select="$cxtName"/></xsl:apply-templates>
</xsl:template>

<xsl:template match="coin:Src_ElevatedRelationColumns" mode="skolemStr">
  <xsl:param name="cxtName"/>
  <xsl:variable name="pairID" select="substring-after(@rdf:resource,'#')"/>
  <xsl:variable name="colID" select="substring-
after(//coin:Src_ElevatedRelationColumnSemanticTypePair[@rdf:ID=$pairID]/coin:Src_ElevatedRelation
Column/@rdf:resource,'#')"/>
  <xsl:variable name="semTypeID" select="substring-
after(//coin:Src_ElevatedRelationColumnSemanticTypePair[@rdf:ID=$pairID]/coin:Src_ElevatedRelation
SemanticType/@rdf:resource,'#')"/>
  <atom><_opr><rel>skolem</rel></_opr>
  <ind><xsl:value-of
select="$ontology/*/coin:Ont_SemanticType[@rdf:ID=$semTypeID]/coin:Ont_SemanticTypeName"/></i
nd>
  <var><xsl:value-of select="//coin:Src_Column[@rdf:ID=$colID]/coin:Src_ColumnName"/></var>
```

```
  <ind><xsl:value-of select="$cxtName"/></ind>
  <ind><xsl:value-of select="position()"/></ind>
  <xsl:apply-templates select=".." mode="relationStr"></xsl:apply-templates>
  </atom>
</xsl:template>

<!-- Context-->
<xsl:template match="coin:Cxt_Context" mode="context">
  <imp>
   <_head><atom><_opr><rel>is_a</rel></_opr>
    <ind><xsl:value-of select="coin:Cxt_ContextName"/></ind>
    <xsl:choose>
     <xsl:when test="coin:Cxt_ContextParent">
      <xsl:variable name="parentID" select="substring-after(coin:Cxt_ContextParent/@rdf:resource,'#')"/>
      <ind><xsl:value-of
select="//coin:Cxt_Context[@rdf:ID=$parentID]/coin:Cxt_ContextName"/></ind>
     </xsl:when>
     <xsl:otherwise><ind>basic</ind></xsl:otherwise>
    </xsl:choose>
    </atom></_head>
   <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
  </imp>
</xsl:template>

<!-- Modifier-->
<xsl:template match="coin:Ont_Modifier" mode="modifier">
  <xsl:apply-templates select="coin:Ont_ModifierContextValues" mode="modifier">
   <xsl:with-param name="semTypeFromID" select="substring-
after(coin:Ont_ModifierFrom/@rdf:resource,'#')"/>
   <xsl:with-param name="semTypeToID" select="substring-
after(coin:Ont_ModifierTo/@rdf:resource,'#')"/>
   <xsl:with-param name="modifierID" select="@rdf:ID"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="coin:Ont_ModifierContextValues" mode="modifier">
  <xsl:param name="semTypeFromID"/>
  <xsl:param name="semTypeToID"/>
  <xsl:param name="modifierID"/>
  <xsl:variable name="pairID" select="substring-after(@rdf:resource,'#')"/>
  <xsl:apply-templates select="//coin:Ont_ModifierContextValuePair[@rdf:ID=$pairID]"
mode="modifier">
   <xsl:with-param name="semTypeFromID" select="$semTypeFromID"/>
   <xsl:with-param name="semTypeToID" select="$semTypeToID"/>
   <xsl:with-param name="modifierID" select="$modifierID"/>
  </xsl:apply-templates>
</xsl:template>

<xsl:template match="coin:Ont_ModifierContextValuePair" mode="modifier">
  <xsl:param name="semTypeFromID"/>
  <xsl:param name="semTypeToID"/>
  <xsl:param name="modifierID"/>
  <xsl:variable name="cxtID" select="substring-after(coin:Ont_ModifierContext/@rdf:resource,'#')"/>
  <xsl:variable name="cxtName"
select="$context/*/coin:Cxt_Context[@rdf:ID=$cxtID]/coin:Cxt_ContextName"/>
  <imp>
```

```
   <_head><atom><_opr><rel>modifier</rel></_opr>
    <ind><xsl:value-of
select="//coin:Ont_SemanticType[@rdf:ID=$semTypeFromID]/coin:Ont_SemanticTypeName"/></ind>
    <var>_O</var>
    <ind><xsl:value-of
select="//coin:Ont_Modifier[@rdf:ID=$modifierID]/coin:Ont_ModifierName"/></ind>
    <ind><xsl:value-of select="$cxtName"/></ind>
    <var>M</var>
   </atom></_head>
   <_body><atom><_opr><rel /></_opr>
    <xsl:choose>
     <xsl:when test="coin:Ont_ModifierStaticValue">
      <atom><_opr><rel>cste</rel></_opr><ind>basic</ind><var>M</var><ind><xsl:value-of
select="$cxtName"/></ind>
       <xsl:variable name="modValue" select="coin:Ont_ModifierStaticValue"/>
       <xsl:choose>
        <xsl:when test="string($modValue)='NaN'"><ind>"<xsl:value-of
select="$modValue"/>"</ind></xsl:when>
        <xsl:otherwise><ind><xsl:value-of select="$modValue"/></ind></xsl:otherwise>
       </xsl:choose>
       </atom>
     </xsl:when>
     <xsl:when test="coin:Ont_ModifierDynamicValue">
      <xsl:apply-templates select="coin:Ont_ModifierDynamicValue" mode="modifier">
       <xsl:with-param name="semTypeFromID" select="$semTypeFromID"/>
       <xsl:with-param name="semTypeToID" select="$semTypeToID"/>
      </xsl:apply-templates>
     </xsl:when>
    </xsl:choose>
   </atom></_body>
  </imp>
</xsl:template>

<xsl:template match="coin:Ont_ModifierDynamicValue" mode="modifier">
  <xsl:param name="semTypeFromID"/>
  <xsl:param name="semTypeToID"/>
  <xsl:variable name="attrID" select="substring-after(@rdf:resource,'#')"/>
  <xsl:variable name="attr" select="//coin:Ont_Attribute[@rdf:ID=$attrID]"/>
  <xsl:variable name="attrFromID" select="substring-after($attr/coin:Ont_AttributeFrom/@rdf:resource,
'#')"/>
  <xsl:variable name="attrToID" select="substring-after($attr/coin:Ont_AttributeTo/@rdf:resource,'#')"/>
  <atom><_opr><rel>attr</rel></_opr>
  <xsl:choose>
    <xsl:when test="$semTypeFromID=$attrFromID"><var>_O</var></xsl:when>
    <xsl:otherwise><ind><xsl:value-of
select="//coin:Ont_SemanticType[@rdf:ID=$attrFromID]/coin:Ont_SemanticTypeName"/></ind></xsl:ot
herwise>
  </xsl:choose>
  <ind><xsl:value-of select="$attr/coin:Ont_AttributeName"/></ind>
  <xsl:choose>
    <xsl:when test="$semTypeToID=$attrToID"><var>M</var></xsl:when>
    <xsl:otherwise><ind><xsl:value-of
select="//coin:Ont_SemanticType[@rdf:ID=$attrToID]/coin:Ont_SemanticTypeName"/></ind></xsl:other
wise>
  </xsl:choose>
  </atom>
```

```
</xsl:template>

<!-- Attribute-->
<xsl:template match="coin:Ont_Attribute" mode="attribute">
 <imp>
  <_head><atom><_opr><rel>attr</rel></_opr>
   <var>X</var>
   <ind><xsl:value-of select="coin:Ont_AttributeName"/></ind>
   <var>Y</var></atom></_head>
  <_body><atom><_opr><rel /></_opr>
   <xsl:choose>
    <xsl:when test="coin:Ont_AttributeElevationFunction">
     <ind><xsl:value-of select="coin:Ont_AttributeElevationFunction"/></ind>
    </xsl:when>
    <xsl:otherwise><ind>true</ind></xsl:otherwise>
   </xsl:choose>
  </atom></_body>
 </imp>
</xsl:template>

<!-- Conversion Function-->
<xsl:template match="coin:Ont_Modifier" mode="convfunc">
 <imp>
  <_head><xsl:value-of select="coin:Ont_ModifierConversionFunction"/></_head>
 </imp>
</xsl:template>

<!-- Helper Function-->

</xsl:stylesheet>
```

F.2 XSLT for RuleML -> RFML

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<?cocoon-process type="xslt"?>

 <!-- process rulebase and position fact/imp transformers -->
 <xsl:template match="/rulebase">
  <rfml>
   <xsl:apply-templates/>
  </rfml>
 </xsl:template>

 <!-- process a fact, transforming it to a hn clause without premises -->
 <xsl:template match="fact">
  <hn>
   <xsl:apply-templates select="_head" mode="pattop"/>
  </hn>
 </xsl:template>

 <!-- process an imp, transforming it to a hn clause with at least one premise -->
 <xsl:template match="imp">
  <hn>
   <xsl:apply-templates select="_head" mode="pattop"/>
   <xsl:apply-templates select="_body" mode="callop"/>
  </hn>
 </xsl:template>

 <!-- process _head atom as pattop -->
 <xsl:template match="atom" mode="pattop">
  <xsl:call-template name="atomfun">
   <xsl:with-param name="pattorcall">pattop</xsl:with-param>
  </xsl:call-template>
 </xsl:template>

 <!-- process _body atom (skipping possible and) as callop -->
 <xsl:template match="atom" mode="callop">
  <xsl:call-template name="atomfun">
   <xsl:with-param name="pattorcall">callop</xsl:with-param>
  </xsl:call-template>
 </xsl:template>

 <!-- process atom and transform to pattop or callop -->
 <xsl:template name="atomfun">
  <xsl:param name="pattorcall"></xsl:param>
  <xsl:element name="{$pattorcall}">
   <con>
    <xsl:value-of select="_opr/rel"/>
   </con>
   <xsl:for-each select="ind|var|cterm">
    <xsl:apply-templates select="."/>
   </xsl:for-each>
   <xsl:apply-templates select="atom" mode="callop"/>
  </xsl:element>
 </xsl:template>
```

```
 <xsl:template match="cterm">
  <struc>
   <con>
    <xsl:value-of select="_opc/ctor"/>
   </con>
   <xsl:for-each select="ind|var|cterm">
    <xsl:apply-templates select="."/>
   </xsl:for-each>
  </struc>
 </xsl:template>

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

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

</xsl:stylesheet>
```

F.3 XSLT for RFML -> Prolog in HTML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<!-- <xsl:strip-space elements="*"/> -->
 <xsl:template match="rfml">
  <xsl:processing-instruction name="cocoon-format">type="text/html"</xsl:processing-instruction>
  <html>
   <head>
    <style type="text/css">
    </style>
   </head>
   <body bgcolor="#EEEEEE">
    <xsl:apply-templates/>
   </body>
  </html>
 </xsl:template>

 <xsl:template match="hn">
  <xsl:choose>
   <xsl:when test="pattop">
     rule(<xsl:apply-templates select="pattop"/><xsl:if test="count(child::*)>1"><tt>, </tt><xsl:for-each
select="(con|var|struc|callop)"><xsl:apply-templates select="."/><xsl:if test="not(position()=last())"><tt>,
</tt></xsl:if></xsl:for-each></xsl:if>).<br/>
    </xsl:when>
    <xsl:otherwise>
        rule(<xsl:value-of select="."/>).<br/>
    </xsl:otherwise>
  </xsl:choose>
 </xsl:template>

 <xsl:template match="ft">
  <xsl:apply-templates select="pattop"/><xsl:choose><xsl:when test="count(child::*)>2"><tt> :-
</tt><xsl:for-each select="(con|var|struc|callop)[not(position()=last())]"><xsl:apply-templates
select="."/><xsl:if test="not(position()=last())"><tt>, </tt></xsl:if></xsl:for-each><tt> &amp;
</tt></xsl:when><xsl:otherwise><tt> :&amp; </tt></xsl:otherwise></xsl:choose><xsl:apply-templates
select="(con|var|struc|callop)[position()=last()]"/><tt>.</tt><br/>
 </xsl:template>

 <xsl:template match="callop">
  <strong><xsl:apply-templates
select="(con|var|struc|pattop|callop|op)[position()=1]"/></strong>(<xsl:for-each
select="(con|var|struc|pattop|callop|op)[position()>1]"><xsl:apply-templates select="."/><xsl:if
test="not(position()=last())"><tt>,</tt></xsl:if></xsl:for-each>)</xsl:template>

 <xsl:template match="pattop">
  <strong><xsl:apply-templates select="(con|var|struc|pattop|callop)[position()=1]"/></strong>(<xsl:for-
each select="(con|var|struc|pattop|callop)[position()>1]"><xsl:apply-templates select="."/><xsl:if
test="not(position()=last())"><tt>,</tt></xsl:if></xsl:for-each>)</xsl:template>

 <xsl:template match="struc">
  <xsl:apply-templates select="(con|var|struc)[position()=1]"/>[<xsl:for-each
select="(con|var|struc)[position()>1]"><xsl:apply-templates select="."/><xsl:if
test="not(position()=last())"><tt>,</tt></xsl:if></xsl:for-each>]</xsl:template>
```

```
  <xsl:template match="op">{<xsl:for-each select="(con|var|struc|callop|op)[position()>0]"><xsl:apply-
templates select="."/><xsl:if test="not(position()=last())"><tt>,</tt></xsl:if></xsl:for-
each>}</xsl:template>

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

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

</xsl:stylesheet>
```

Appendix G – Tent Example in Various Formats

G.1 Tent Example in RDF

application392.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
 <coin:Application rdf:ID="application392">
  <coin:ApplicationOntology rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf" />
  <coin:ApplicationContext rdf:resource="http://localhost/appEditor/apps/rdf/context392.rdf" />
  <coin:ApplicationSource rdf:resource="http://localhost/appEditor/apps/rdf/source392.rdf" />
  <coin:ApplicationMisc rdf:resource="http://localhost/appEditor/apps/rdf/misc392.rdf" />
 </coin:Application>
</rdf:RDF>
```

ontology392.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
 <!--Semantic Types-->
 <coin:Ont_SemanticType rdf:ID="product">
  <coin:Ont_SemanticTypeName>product</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="basic">
  <coin:Ont_SemanticTypeName>basic</coin:Ont_SemanticTypeName>
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="tent">
  <coin:Ont_SemanticTypeName>tent</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#product" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="weight">
  <coin:Ont_SemanticTypeName>weight</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
 </coin:Ont_SemanticType>
 <!--Attributes-->
 <coin:Ont_Attribute rdf:ID="weight">
  <coin:Ont_AttributeName>weight</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#product" />
  <coin:Ont_AttributeTo rdf:resource="#weight" />
  <coin:Ont_AttributeElevationFunction />
 </coin:Ont_Attribute>
 <!--Modifiers-->
 <coin:Ont_Modifier rdf:ID="weightUnit">
  <coin:Ont_ModifierName>weightUnit</coin:Ont_ModifierName>
  <coin:Ont_ModifierFrom rdf:resource="#weight" />
  <coin:Ont_ModifierTo rdf:resource="#basic" />

<coin:Ont_ModifierConversionFunction>lib_physical_unit|weight|weightUnit</coin:Ont_ModifierConvers
ionFunction>
  <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_us" />
```

```
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_uk" />
  </coin:Ont_Modifier>
  <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_us">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context392.rdf#c_us" />
    <coin:Ont_ModifierStaticValue>lb</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_uk">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context392.rdf#c_uk" />
    <coin:Ont_ModifierStaticValue>kg</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
</rdf:RDF>
```

context392.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
  <!--Contexts-->
  <coin:Cxt_Context rdf:ID="c_us">
    <coin:Cxt_ContextName>c_us</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_uk">
    <coin:Cxt_ContextName>c_uk</coin:Cxt_ContextName>
  </coin:Cxt_Context>
</rdf:RDF>
```

source392.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
  <!--Relations-->
  <coin:Src_Relation rdf:ID="dsr_tent_terranova">
    <coin:Src_RelationName>dsr_tent_terranova</coin:Src_RelationName>
    <coin:Src_RelationImport>true</coin:Src_RelationImport>
    <coin:Src_RelationExport>true</coin:Src_RelationExport>
    <coin:Src_RelationSourceName>oracle</coin:Src_RelationSourceName>
    <coin:Src_RelationUnsupportedOps />
  </coin:Src_Relation>
  <!--Columns-->
  <coin:Src_Column rdf:ID="Maker">
    <coin:Src_ColumnName>Maker</coin:Src_ColumnName>
    <coin:Src_ColumnType>string</coin:Src_ColumnType>
    <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
    <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
  </coin:Src_Column>
  <coin:Src_Column rdf:ID="Model">
    <coin:Src_ColumnName>Model</coin:Src_ColumnName>
    <coin:Src_ColumnType>string</coin:Src_ColumnType>
    <coin:Src_ColumnKeyMember>true</coin:Src_ColumnKeyMember>
    <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
  </coin:Src_Column>
  <coin:Src_Column rdf:ID="Name">
    <coin:Src_ColumnName>Name</coin:Src_ColumnName>
    <coin:Src_ColumnType>string</coin:Src_ColumnType>
```

```
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
 </coin:Src_Column>
 <coin:Src_Column rdf:ID="Seasons">
  <coin:Src_ColumnName>Seasons</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
 </coin:Src_Column>
 <coin:Src_Column rdf:ID="Sleeps">
  <coin:Src_ColumnName>Sleeps</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
 </coin:Src_Column>
 <coin:Src_Column rdf:ID="Minimum_weight">
  <coin:Src_ColumnName>Minimum_weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
 </coin:Src_Column>
 <coin:Src_Column rdf:ID="Floor_area">
  <coin:Src_ColumnName>Floor_area</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
 </coin:Src_Column>
 <coin:Src_Column rdf:ID="Price">
  <coin:Src_ColumnName>Price</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
 </coin:Src_Column>
 <!--Elevated Relations-->
 <coin:Src_ElevatedRelation rdf:ID="dsr_tent_terranova_p">
  <coin:Src_ElevatedRelationName>dsr_tent_terranova_p</coin:Src_ElevatedRelationName>
  <coin:Src_ElevatedRelationRelation rdf:resource="#dsr_tent_terranova" />
  <coin:Src_ElevatedRelationContext rdf:resource="#c_uk" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pMaker" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pModel" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pName" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pSeasons" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pSleeps" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pMinimum_weight" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pFloor_area" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pPrice" />
 </coin:Src_ElevatedRelation>
 <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pMaker">
  <coin:Src_ElevatedRelationColumn rdf:resource="#Maker" />
  <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
 </coin:Src_ElevatedRelationColumnSemanticTypePair>
 <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pModel">
  <coin:Src_ElevatedRelationColumn rdf:resource="#Model" />
  <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
```

```
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pName">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Name" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pSeasons">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Seasons" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pSleeps">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Sleeps" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pMinimum_weight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Minimum_weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pFloor_area">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Floor_area" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pPrice">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Price" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology392.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
</rdf:RDF>
```

misc392.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
  <!--Miscellaneous-->
</rdf:RDF>
```

G.2 Tent Example in RuleML

```
<?xml version="1.0"?>
<rulebase>
 <imp>
  <_head><atom><_opr><rel>is_a</rel></_opr><ind>product</ind><ind>basic</ind></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>is_a</rel></_opr><ind>basic</ind><ind>basic</ind></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>is_a</rel></_opr><ind>tent</ind><ind>product</ind></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>is_a</rel></_opr><ind>weight</ind><ind>basic</ind></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>attributes</rel></_opr><ind>product</ind>
    <cterm><_opc><ctor /></_opc><ind>weight</ind></cterm></atom>
  </_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>attributes</rel></_opr><ind>basic</ind>
    <cterm><_opc><ctor /></_opc></cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>attributes</rel></_opr><ind>tent</ind>
    <cterm><_opc><ctor /></_opc></cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>attributes</rel></_opr><ind>weight</ind>
    <cterm><_opc><ctor /></_opc></cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>modifiers</rel></_opr><ind>product</ind>
    <cterm><_opc><ctor /></_opc></cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>modifiers</rel></_opr><ind>basic</ind>
    <cterm><_opc><ctor /></_opc></cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
 </imp>
 <imp>
  <_head><atom><_opr><rel>modifiers</rel></_opr><ind>tent</ind>
    <cterm><_opc><ctor /></_opc></cterm></atom></_head>
  <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
```

```
</imp>
<imp>
 <_head><atom><_opr><rel>modifiers</rel></_opr><ind>weight</ind>
   <cterm><_opc><ctor /></_opc><ind>weightUnit</ind></cterm></atom></_head>
 <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
</imp>
<imp>
 <_head><atom><_opr><rel>relation</rel></_opr><ind>oracle</ind><ind>dsr_tent_terranova</ind>
   <ind>ie</ind>
   <cterm><_opc><ctor /></_opc>
    <cterm><_opc><ctor /></_opc><ind>'Maker'</ind><ind>string</ind></cterm>
    <cterm><_opc><ctor /></_opc><ind>'Model'</ind><ind>string</ind></cterm>
    <cterm><_opc><ctor /></_opc><ind>'Name'</ind><ind>string</ind></cterm>
    <cterm><_opc><ctor /></_opc><ind>'Seasons'</ind><ind>string</ind>/cterm>
    <cterm><_opc><ctor /></_opc><ind>'Sleeps'</ind><ind>string</ind></cterm>
    <cterm><_opc><ctor /></_opc><ind>'Minimum_weight'</ind><ind>string</ind></cterm>
    <cterm><_opc><ctor /></_opc><ind>'Floor_area'</ind><ind>string</ind></cterm>
    <cterm><_opc><ctor /></_opc><ind>'Price'</ind><ind>string</ind></cterm>
   </cterm>
   <atom><_opr><rel>cap</rel></_opr>
    <cterm><_opc><ctor /></_opc>
     <cterm><_opc><ctor /> </_opc>
      <ind>0</ind><ind>1</ind><ind>0</ind><ind>0</ind><ind>0</ind>
      <ind>0</ind><ind>0</ind><ind>0</ind>
     </cterm>
    </cterm>
    <cterm><_opc><ctor /></_opc><ind></ind></cterm>
   </atom></atom></_head>
 <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
</imp>
<imp>
 <_head>
  <atom>
   <_opr><rel>dsr_tent_terranova_p</rel></_opr>
   <atom><_opr><rel>skolem</rel></_opr>
    <ind>basic</ind><var>Maker</var><ind>c_uk</ind><ind>1</ind>
    <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
     <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
     <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
   <atom><_opr><rel>skolem</rel></_opr>
    <ind>basic</ind><var>Model</var><ind>c_uk</ind><ind>2</ind>
    <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
     <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
     <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
   <atom><_opr><rel>skolem</rel></_opr>
    <ind>basic</ind><var>Name</var><ind>c_uk</ind><ind>3</ind>
    <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
     <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
     <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
   <atom><_opr><rel>skolem</rel></_opr>
    <ind>basic</ind><var>Seasons</var><ind>c_uk</ind><ind>4</ind>
          <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
     <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
     <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
   <atom><_opr><rel>skolem</rel></_opr>
    <ind>basic</ind><var>Sleeps</var><ind>c_uk</ind><ind>5</ind>
```

```
            <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
          <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
          <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
        <atom><_opr><rel>skolem</rel></_opr>
         <ind>weight</ind><var>Minimum_weight</var><ind>c_uk</ind><ind>6</ind>
            <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
          <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
          <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
        <atom><_opr><rel>skolem</rel></_opr>
         <ind>basic</ind><var>Floor_area</var><ind>c_uk</ind><ind>7</ind>
         <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
          <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
          <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
        <atom><_opr><rel>skolem</rel></_opr>
         <ind>basic</ind><var>Price</var><ind>c_uk</ind><ind>8</ind>
            <atom><_opr><rel>dsr_tent_terranova</rel></_opr><var>Maker</var>
          <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
          <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></atom></atom>
      </atom></_head>
    <_body><atom><_opr><rel /></_opr><atom><_opr><rel>dsr_tent_terranova</rel></_opr>
        <var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var>
        <var>Sleeps</var><var>Minimum_weight</var><var>Floor_area</var><var>Price</var>
        </atom></atom></_body>
  </imp>
  <imp>
   <_head><atom><_opr><rel>is_a</rel></_opr><ind>c_us</ind><ind>basic</ind></atom></_head>
   <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
  </imp>
  <imp>
   <_head><atom><_opr><rel>is_a</rel></_opr><ind>c_uk</ind><ind>basic</ind></atom></_head>
   <_body><atom><_opr><rel /></_opr><ind>true</ind></atom></_body>
  </imp>
  <imp>
   <_head><atom><_opr><rel>modifier</rel></_opr><ind>weight</ind>
      <var>_O</var><ind>weightUnit</ind><ind>c_us</ind><var>M</var></atom></_head>
   <_body><atom><_opr><rel /></_opr><atom><_opr><rel>cste</rel></_opr>
       <ind>basic</ind><var>M</var><ind>c_us</ind><ind>lb</ind></atom></atom></_body>
  </imp>
  <imp>
   <_head><atom><_opr><rel>modifier</rel></_opr><ind>weight</ind>
      <var>_O</var><ind>weightUnit</ind><ind>c_uk</ind><var>M</var></atom></_head>
   <_body><atom><_opr><rel /></_opr><atom><_opr><rel>cste</rel></_opr>
       <ind>basic</ind><var>M</var><ind>c_uk</ind><ind>kg</ind></atom></atom></_body>
  </imp>
  <imp>
   <_head><atom><_opr><rel>attr</rel></_opr><var>X</var><ind>weight</ind>
      <var>Y</var></atom></_head>
   <_body><atom><_opr><rel /></_opr><ind></ind></atom></_body>
  </imp>
  <imp>
   <_head>lib_physical_unit|weight|weightUnit</_head>
  </imp>
</rulebase>
```

G.3 Tent Example in RFML

```
<?xml version="1.0"?>
<rfml>
 <hn>
  <pattop><con>is_a</con><con>product</con><con>basic</con></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>is_a</con><con>basic</con><con>basic</con></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>is_a</con><con>tent</con><con>product</con></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>is_a</con><con>weight</con><con>basic</con></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>attributes</con><con>product</con>
    <struc><con></con><con>weight</con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>attributes</con><con>basic</con><struc><con></con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>attributes</con><con>tent</con><struc><con></con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>attributes</con><con>weight</con><struc><con></con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>modifiers</con><con>product</con><struc><con></con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>modifiers</con><con>basic</con><struc> <con></con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>modifiers</con><con>tent</con><struc><con></con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>modifiers</con><con>weight</con>
    <struc><con></con><con>weightUnit</con></struc></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
```

<pattop><con>relation</con><con>oracle</con><con>dsr_tent_terranova</con><con>ie</con>
<struc><con></con>
<struc><con></con><con>'Maker'</con><con>string</con></struc>
<struc><con></con><con>'Model'</con><con>string</con></struc>
<struc><con></con><con>'Name'</con><con>string</con></struc>
<struc><con></con><con>'Seasons'</con><con>string</con> </struc>
<struc><con></con><con>'Sleeps'</con><con>string</con></struc>
<struc><con></con><con>'Minimum_weight'</con><con>string</con></struc>
<struc><con></con><con>'Floor_area'</con><con>string</con></struc>
<struc><con></con><con>'Price'</con><con>string</con></struc>
</struc>
<callop><con>cap</con>
<struc><con></con><struc><con></con><con>0</con><con>1</con><con>0</con><con>0</con><con>0</con>
<con>0</con><con>0</con><con>0</con><con>0</con></struc>
</struc>
<struc><con></con><con></con></struc></callop>
</pattop>
<callop><con></con><con>true</con></callop>
</hn>
<hn>
<pattop><con>dsr_tent_terranova_p</con>
<callop><con>skolem</con><con>basic</con><var>Maker</var><con>c_uk</con><con>1</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>basic</con><var>Model</var><con>c_uk</con>con>2</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>basic</con><var>Name</var><con>c_uk</con><con>3</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>basic</con><var>Seasons</var><con>c_uk</con><con>4</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>basic</con><var>Sleeps</var><con>c_uk</con><con>5</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>weight</con><var>Minimum_weight</var><con>c_uk</con>
<con>6</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>basic</con><var>Floor_area</var><con>c_uk</con>
<con>7</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
<callop><con>skolem</con><con>basic</con> <var>Price</var><con>c_uk</con><con>8</con>
<callop><con>dsr_tent_terranova</con>
<var>Maker</var><var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
<var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
</pattop>

```
  <callop><con></con><callop><con>dsr_tent_terranova</con><var>Maker</var>
    <var>Model</var><var>Name</var><var>Seasons</var><var>Sleeps</var>
    <var>Minimum_weight</var><var>Floor_area</var><var>Price</var></callop></callop>
 </hn>
 <hn>
  <pattop><con>is_a</con><con>c_us</con><con>basic</con></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>is_a</con><con>c_uk</con><con>basic</con></pattop>
  <callop><con></con><con>true</con></callop>
 </hn>
 <hn>
  <pattop><con>modifier</con><con>weight</con><var>_O</var>
   <con>weightUnit</con><con>c_us</con><var>M</var></pattop>
  <callop>con></con>
   <callop><con>cste</con><con>basic</con><var>M</var><con>c_us</con>
    <con>lb</con></callop></callop>
 </hn>
 <hn>
  <pattop><con>modifier</con><con>weight</con><var>_O</var>
   <con>weightUnit</con><con>c_uk</con><var>M</var></pattop>
  <callop><con></con>
   <callop><con>cste</con><con>basic</con><var>M</var><con>c_uk</con><con>kg</con></callop>
  </callop>
 </hn>
 <hn>
  <pattop><con>attr</con><var>X</var><con>weight</con><var>Y</var></pattop>
  <callop><con></con><con></con></callop>
 </hn>
 <hn>lib_physical_unit|weight|weightUnit</hn>
</rfml>
```

G.4 Tent Example in HTML (Prolog)

<?xml version="1.0"?>
<?cocoon-format type="text/html"?>
<html>
 <head>
  <style type="text/css">
  </style>
 </head>
 <body bgcolor="#EEEEEE">
    rule(is_a(product<tt>,</tt>basic)<tt>,
</tt>(true)).
    rule(is_a(basic<tt>,</tt>basic)<tt>,
</tt>(true)).
    rule(is_a(tent<tt>,</tt>product)<tt>,
</tt>(true)).
    rule(is_a(weight<tt>,</tt>basic)<tt>,
</tt>(true)).
    rule(attributes(product<tt>,</tt>[weight])<tt>,
</tt>(true)).
    rule(attributes(basic<tt>,</tt>[])<tt>,
</tt>(true)).
    rule(attributes(tent<tt>,</tt>[])<tt>,
</tt>(true)).
    rule(attributes(weight<tt>,</tt>[])<tt>,
</tt>(true)).
    rule(modifiers(product<tt>,</tt>[])<tt>,
</tt>(true)).
    rule(modifiers(basic<tt>,</tt>[])<tt>,
</tt>(true)).
    rule(modifiers(tent<tt>,</tt>[])<tt>,
</tt>(true)).

rule(modifiers(weight<tt>,</tt>[weightUnit])<tt>,
</tt>(true)).

rule(relation(oracle<tt>,</tt>dsr_tent_terranova<tt>,</tt>i
e<tt>,</tt>[['Maker'<tt>,</tt>string]<tt>,</tt>['Model'
<tt>,</tt>string]<tt>,</tt>['Name'<tt>,</tt>string]<tt>,</tt>['Seasons'<tt>,</tt>string]<tt>,</tt>['Sleeps'<tt>,</tt>string]<
tt>,</tt>['Minimum_weight'<tt>,</tt>string]<tt>,</tt>['Floor_area'<
/b><tt>,</tt>string]<tt>,</tt>['Price'<tt>,</tt>string]]<tt>,</tt>
cap([[0<tt>,</tt>1<tt>,</tt>0<tt>,</tt>0
<tt>,</tt>0<tt>,</tt>0<tt>,</tt>0<tt>,</tt>0]]<tt>,</tt>[<b
>]))<tt>, </tt>(true)).

rule(dsr_tent_terranova_p(skolem(basic<tt
>,</tt>Maker<tt>,</tt>c_uk<tt>,</tt>1<tt>,</tt>dsr_tent_terranova<
/b>(Maker<tt>,</tt>Model<tt>,</tt>Name<tt>,</tt>Seasons<tt>,<
/tt>Sleeps<tt>,</tt>Minimum_weight<tt>,</tt>Floor_area<tt>,</tt>Price))<t
t>,</tt>skolem(basic<tt>,</tt>Model<tt>,</tt>c_uk<tt
>,</tt>2<tt>,</tt>dsr_tent_terranova(Maker<tt>,</tt>Mode
l<tt>,</tt>Name<tt>,</tt>Seasons<tt>,</tt>Sleeps<tt>,</tt>Minimum_weig
ht<tt>,</tt>Floor_area<tt>,</tt>Price))<tt>,</tt>skolem(<b
>basic<tt>,</tt>Name<tt>,</tt>c_uk<tt>,</tt>3<tt>,</tt>dsr_te

nt_terranova(Maker<tt>,</tt>Model<tt>,</tt>Name<tt>,</tt>Seas
ons<tt>,</tt>Sleeps<tt>,</tt>Minimum_weight<tt>,</tt>Floor_area<tt>,</tt><i
>Price))<tt>,</tt>skolem(basic<tt>,</tt>Seasons<tt>,</tt>
c_uk<tt>,</tt>4<tt>,</tt>dsr_tent_terranova(Maker<tt
>,</tt>Model<tt>,</tt>Name<tt>,</tt>Seasons<tt>,</tt>Sleeps<tt>,</tt>
Minimum_weight<tt>,</tt>Floor_area<tt>,</tt>Price))<tt>,</tt>skolem</
b>(basic<tt>,</tt>Sleeps<tt>,</tt>c_uk<tt>,</tt>5<tt>,</tt><st
rong>dsr_tent_terranova(Maker<tt>,</tt>Model<tt>,</tt>Name<t
t>,</tt>Seasons<tt>,</tt>Sleeps<tt>,</tt>Minimum_weight<tt>,</tt>Floor_area
<tt>,</tt>Price))<tt>,</tt>skolem(weight<tt>,</tt>Mini
mum_weight<tt>,</tt>c_uk<tt>,</tt>6<tt>,</tt>dsr_tent_terranova
(Maker<tt>,</tt>Model<tt>,</tt>Name<tt>,</tt>Seasons<tt>,</tt>
Sleeps<tt>,</tt>Minimum_weight<tt>,</tt>Floor_area<tt>,</tt>Price))<tt>,
</tt>skolem(basic<tt>,</tt>Floor_area<tt>,</tt>c_uk<
tt>,</tt>7<tt>,</tt>dsr_tent_terranova(Maker<tt>,</tt>Mod
el<tt>,</tt>Name<tt>,</tt>Seasons<tt>,</tt>Sleeps<tt>,</tt>Minimum_wei
ght<tt>,</tt>Floor_area<tt>,</tt>Price))<tt>,</tt>skolem(<
b>basic<tt>,</tt>Price<tt>,</tt>c_uk<tt>,</tt>8<tt>,</tt>dsr_te
nt_terranova(Maker<tt>,</tt>Model<tt>,</tt>Name<tt>,</tt>Seas
ons<tt>,</tt>Sleeps<tt>,</tt>Minimum_weight<tt>,</tt>Floor_area<tt>,</tt><i
>Price)))<tt>,
</tt>(dsr_tent_terranova(Maker<tt>,</tt><i
>Model<tt>,</tt>Name<tt>,</tt>Seasons<tt>,</tt>Sleeps<tt>,</tt>Minimu
m_weight<tt>,</tt>Floor_area<tt>,</tt>Price))).

rule(is_a(c_us<tt>,</tt>basic)<tt>,
</tt>(true)).

rule(is_a(c_uk<tt>,</tt>basic)<tt>,
</tt>(true)).

rule(modifier(weight<tt>,</tt>_O<tt>,</tt>weightUnit
<tt>,</tt>c_us<tt>,</tt>M)<tt>,
</tt>(cste(basic<tt>,</tt>M<tt>,</tt
>c_us<tt>,</tt>lb))).

rule(modifier(weight<tt>,</tt>_O<tt>,</tt>weightUnit
<tt>,</tt>c_uk<tt>,</tt>M)<tt>,
</tt>(cste(basic<tt>,</tt>M<tt>,</tt
>c_uk<tt>,</tt>kg))).
rule(attr(X<tt>,</tt>weight<tt>,</tt>Y)<tt>,
</tt>()).

rule(lib_physical_unit|weight|weightUnit).
</body>
</html>

G.5 Tent Example in Prolog

```
% ontology
rule(is_a(product,basic),(true)).
rule(is_a(basic,basic),(true)).
rule(is_a(tent,product),(true)).
rule(is_a(weight,basic),(true)).
rule(attributes(product,[weight]),(true)).
rule(attributes(basic,[]),(true)).
rule(attributes(tent,[]),(true)).
rule(attributes(weight,[]),(true)).
rule(modifiers(product,[]),(true)).
rule(modifiers(basic,[]),(true)).
rule(modifiers(tent,[]),(true)).
rule(modifiers(weight,[weightUnit]),(true)).
rule(relation(oracle,dsr_tent_terranova,ie,[['Maker',string],['Model',s
tring],['Name',string],['Seasons',string],['Sleeps',string],['Minimum_w
eight',string],['Floor_area',string],['Price',string]],cap([[0,1,0,0,0,
0,0,0]],[])),(true)).

% elevation
rule(dsr_tent_terranova_p(skolem(basic,Maker,c_uk,1,dsr_tent_terranova(
Maker,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,Price)),skole
m(basic,Model,c_uk,2,dsr_tent_terranova(Maker,Model,Name,Seasons,Sleeps
,Minimum_weight,Floor_area,Price)),skolem(basic,Name,c_uk,3,dsr_tent_te
rranova(Maker,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,Price
)),skolem(basic,Seasons,c_uk,4,dsr_tent_terranova(Maker,Model,Name,Seas
ons,Sleeps,Minimum_weight,Floor_area,Price)),skolem(basic,Sleeps,c_uk,5
,dsr_tent_terranova(Maker,Model,Name,Seasons,Sleeps,Minimum_weight,Floo
r_area,Price)),skolem(weight,Minimum_weight,c_uk,6,dsr_tent_terranova(M
aker,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,Price)),skolem
(basic,Floor_area,c_uk,7,dsr_tent_terranova(Maker,Model,Name,Seasons,Sl
eeps,Minimum_weight,Floor_area,Price)),skolem(basic,Price,c_uk,8,dsr_te
nt_terranova(Maker,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,
Price))),(dsr_tent_terranova(Maker,Model,Name,Seasons,Sleeps,Minimum_we
ight,Floor_area,Price))).

% context
rule(is_a(c_us,basic),(true)).
rule(is_a(c_uk,basic),(true)).

% modifier
rule(modifier(weight,_O,weightUnit,c_us,M),(cste(basic,M,c_us,lb))).
rule(modifier(weight,_O,weightUnit,c_uk,M),(cste(basic,M,c_uk,kg))).

% attributes
rule(attr(X,weight,Y),()).

%% conversion functions
%%
%% lib_physical_unit
%%
rule(relation(cameleon, lib_physical_unit, i,
[['FrmUnit',string],['ToUnit',string],['UnitFactor',string]],
cap([[0,0,0]],[])), (true)).
rule(lib_physical_unit_p(
```

```
skolem(basic, FrmUnit, Ctxt, 1, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor)),
skolem(basic, ToUnit, Ctxt, 2, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor)),
skolem(basic, UnitFactor, Ctxt, 3, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor))),
(lib_physical_unit(FrmUnit, ToUnit, UnitFactor))).
rule(cvt(commutative, length, Object, lengthUnit, Ctxt, Ms, Vs, Mt,
Vt),
(lib_physical_unit_p(Fu, Tu, Uf),value(Fu, Ctxt, Ms),value(Tu, Ctxt,
Mt),value(Uf, Ctxt, Ufv),multiply(Vs, Ufv, Vt))).
%%
%% lib_physical_unit end
%%
```

Appendix H – Interface Manual

COIN Textual Interface Manual

Introduction

The COIN Textual Interface is created to simplify the entry of ontology and other elements of the COIN models. The textual interface is intended to provide to same functionality as a graphical one, but without the overhead associated with graphical components. While a graphical interface can convey more visual information than a textual interface, it is not feasible in some client settings to install additional browser support software for the graphical interface to work. The COIN textual interface organizes the information in form-based format, which contains only textual components found in simple web forms. The sections that follow are presented in the order that better illustrates the relationship between the components of COIN model. New users are encouraged to follow the manual in the prescribed order.

1. Selecting an Application/Creating an Application

Application ID is a string that uniquely identifies the COIN model for an application. Once an application's model is retrieved/created, all subsequent changes made to the model will be applied to the application specified in the session until a new application is retrieved/created. The application ID is kept as the different screens are traversed so the ID only needs to be identified once by the user. Upon entering the starting page TextInterface.aspx, a list is existing applications are presented for the use to select.

To retrieve an application's model from starting page:

a. Select the application from the list box.
b. Press the "Get" button next to the list box

To retrieve an application's model from subsequent pages:

a. Enter the application ID into the text box labeled "App ID"
b. Press the "Get" button next to the text box.

To create a new application's model on starting page:

a. Enter the application name into the text box labeled "App Name"
b. Press the "New" button next to the text box. An app ID will be generated and the new application can now be accessed through its ID.

2. Ontology

The ontology screen can be reached by pressing the "Ontology" button at the button of every screen. In this section we describe how to define an ontology on this screen.

2.1 Semantic Types

Semantic types already created are displayed in the list box. Deleting a semantic removes all the inheritance and attribute relationships associated with it.

To create a new semantic type:

a. Enter the semantic type name in the text box
b. Press the "Add" button next to the text box

To delete a semantic type:

a. Select the semantic type name in the list box
b. Press the "Delete" button next to the list box

2.2 Inheritances

Inheritance relationships between semantic types are displayed in the list box.
To create a new inheritance relationship:

a. Select the parent semantic type from the left drop down box
b. Select the child semantic type from the right drop down box
c. Press the "Add" button next to the drop down box

To delete an existing inheritance relationship:

a. Select the inheritance relationship from the list box
b. Press the "Delete" button next to the list box

2.3 Attributes

Attribute relationships between semantic types are displayed in the list box. Modifiers are treated as special attributes in the ontology.
To create a new attribute relationship:

a. Enter the attribute name in the text box labeled "Attribute Name"
b. Select the semantic type, which the new attribute describes, in the drop down box labeled "Domain"
c. Select the semantic type, which is the type of the attribute, in the drop down box labeled "Range"
d. If the attribute being added is a modifier, check the "modifier" check box
e. Press the "Add" button next to the list box.

To delete an existing attribute relationship:

a. Select the attribute relation from the list box
b. Press the "Delete" button next to the list box

3. Source

The source screen describes the physical (database, website, etc) data source that are available to the application. Each data source is labeled a relation, which is similar to a table in a database. Relation contains columns, which are like the columns of a database table.

3.1 Relation

A relation must first be defined before its columns can be defined. Relations already defined are displayed in the list box. Removing a relation removes all the columns associated with it.
To create a new relation:

a. Enter the name of the relation in the text box labeled "Relation Name"
b. Select the import/export type of the relation by selecting one of the radial buttons in the group labeled "Relation Type"
c. Select the unsupported operation of the relation by checking one or more of the checkboxes in the group labeled "Unsupported Operation"
d. Enter the source name (oracle, cameleon, etc) of the relation in the text box labeled "Source Name"

To delete an existing relation:

a. Select the relation from the list box

b. Press the "Delete" button next to the list box

3.2 Column

Columns of a relation are created/deleted here.

To create a new column for a relation:

a. Select the relation which the column is to be added from the drop down box labeled "Relation"

b. Press the "Get" button. All the existing columns of the selected relation are displayed in the list box.

c. Enter the name of the new column in the text box labeled "Column Name"

d. Select the data type of the column from the drop down list labeled "Column Type"

e. If the column is a member of the primary key, check the checkbox labeled "Member of Primary Key"

f. Press the "Add" button next to the checkbox

To delete an existing column:

a. Select the column from the list box

b. Press the "Delete" button next to the list box

4. Context

The context screen is where all context related information are specified. A context has to be created before its relevant information can be specified. All existing contexts are displayed in the list box.

To create a new context:

a. Enter the name of the context in the text box labeled "Context Name"

b. If the new context shares contextual information with an existing context, select the existing context from the drop down box

c. Press the "Add" button

To delete an existing context:

a. Select the context from the list box

b. Press the "Delete" button next to the list box

Removing a context will remove only the context sharing information associated with it. Other information (elevations, modifiers) related to the deleted context remains in the application.

4.1 Elevation

Elevations are defined per column in a relation. "No context" is the default relation context. A context has to be added to a relation before elevations can be defined. Elevation for a relation can only be defined for one context, the relation context.

To display the elevations of a relation:

a. Select a context from the drop down box labeled "Context".

b. Press the "Get" button. A list of relation already associated with the selected context is displayed in the list box.

c. Select the relation for which elevations are to be retrieved from the list box labeled "Relation"

d. Press the "Get" button. A list of columns belonging to the relation are displayed in the list box labeled "Column"; and a list of all available semantic types are displayed in the list box labeled "Semantic Type". All defined elevations of the relation under the selected context are displayed in the list box.

To add a relation context:

a. Select a context from the drop down box labeled "Context"
b. Select a relation from the drop down box labeled "Relation" (it is under the relation list box)
c. Press the "Add Relation Context" button. The relation is now ready for elevations to be defined under the selected context.

To delete a relation context:

a. Select a context from the drop down box labeled "Context"

b. Select a relation from the list box labeled "Relation"

c. Press the "Delete Relation Context" button. The relation context is deleted, but the elevations for its columns still exist.

To add an elevation:

a. Follow the instruction for displaying the elevations of a relation
b. Select a column from the list box labeled "Column"
c. Select a semantic type from the list box labeled "Semantic Type"
d. Press the "Add Elevation" button.

To delete an elevation:

a. Follow the instruction for displaying the elevations of a relation
b. Select a elevation from the list box labeled "Relation Elevation"
c. Press the "Delete Elevation" button

4.2 Modifier

Modifiers are defined per context. A static modifier has modifier value that is static. A dynamic modifier has modifier value that is based on values on a semantic type in the ontology.

To retrieve all modifier values for a context:

a. Select a context from the drop down box labeled "Context"

b. Press the "Get" button. All the modifier values associated with the selected context are displayed in the list box labeled "Modifier".

To retrieve the modifier values for a modifier across all contexts:

a. Select a modifier from the drop down box labeled "Modifier"

b. Press the "Get" button. All the modifier values associated with the selected modifier are displayed in the list box labeled "Modifier".

To add a modifier value:

a. Select a context from the drop down box labeled "Context"
b. Select a modifier from the drop down box labeled "Modifier"

c. Specify the type of modifier value by selecting either Static or Dynamic from the radial button group.

d. Enter the modifier value in the text box labeled "Modifier Value". For a static modifier value, any text string are allowed. For a dynamic modifier value, the character ">" is reserved for linking attributes.

e. Press the "Add" button. The modifier values just created is displayed in the list box labeled "Modifier".

To delete a modifier value:

a. Follow the instructions on retrieving the modifier values.

b. Select the modifier value to be deleted from the list box labeled "Modifier".

c. Press the "Delete" button.

4.3 Conversion Function

Conversion functions are defined per modifier. Currently, the content of the conversion function has to be in prolog.

To add a conversion function:

a. Select the modifier from the drop down list labeled "Modifier"

b. Press the "Get" button.

c. Enter the conversion function in the text box labeled "Conversion Function"

d. Press the "Add" button. The conversion function just added is displayed in the list box below.

To delete a conversion function:

a. Select the modifier from the drop down list labeled "Modifier"

b. Press the "Get" button.

c. Select the conversion function to be deleted in the list box under the label "Conversion Function"

d. Press the "Delete" button.

Appendix I – Disaster Relief Metadata File

I.1 RDF

application395.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
 <coin:Application rdf:ID="application395">
  <coin:ApplicationOntology rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf" />
  <coin:ApplicationContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf" />
  <coin:ApplicationSource rdf:resource="http://localhost/appEditor/apps/rdf/source395.rdf" />
  <coin:ApplicationMisc rdf:resource="http://localhost/appEditor/apps/rdf/misc395.rdf" />
 </coin:Application>
</rdf:RDF>
```

ontology395.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
 <!--Semantic Types-->
 <coin:Ont_SemanticType rdf:ID="basic">
  <coin:Ont_SemanticTypeName>basic</coin:Ont_SemanticTypeName>
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="product">
  <coin:Ont_SemanticTypeName>product</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="company">
  <coin:Ont_SemanticTypeName>company</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="tent">
  <coin:Ont_SemanticTypeName>tent</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#product" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="monetaryValue">
  <coin:Ont_SemanticTypeName>monetaryValue</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="price">
  <coin:Ont_SemanticTypeName>price</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#monetaryValue" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="physicalMeasurement">
  <coin:Ont_SemanticTypeName>physicalMeasurement</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="weight">
  <coin:Ont_SemanticTypeName>weight</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#physicalMeasurement" />
 </coin:Ont_SemanticType>
 <coin:Ont_SemanticType rdf:ID="length">
  <coin:Ont_SemanticTypeName>length</coin:Ont_SemanticTypeName>
```

```
  <coin:Ont_SemanticTypeParent rdf:resource="#physicalMeasurement" />
</coin:Ont_SemanticType>
<coin:Ont_SemanticType rdf:ID="area">
  <coin:Ont_SemanticTypeName>area</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#physicalMeasurement" />
</coin:Ont_SemanticType>
<coin:Ont_SemanticType rdf:ID="volume">
  <coin:Ont_SemanticTypeName>volume</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#physicalMeasurement" />
</coin:Ont_SemanticType>
<coin:Ont_SemanticType rdf:ID="sleepingCapacity">
  <coin:Ont_SemanticTypeName>sleepingCapacity</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
</coin:Ont_SemanticType>
<coin:Ont_SemanticType rdf:ID="usage">
  <coin:Ont_SemanticTypeName>usage</coin:Ont_SemanticTypeName>
  <coin:Ont_SemanticTypeParent rdf:resource="#basic" />
</coin:Ont_SemanticType>
<!--Attributes-->
<coin:Ont_Attribute rdf:ID="maker">
  <coin:Ont_AttributeName>maker</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#product" />
  <coin:Ont_AttributeTo rdf:resource="#company" />
  <coin:Ont_AttributeElevationFunction />
</coin:Ont_Attribute>
<coin:Ont_Attribute rdf:ID="price">
  <coin:Ont_AttributeName>price</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#product" />
  <coin:Ont_AttributeTo rdf:resource="#price" />
  <coin:Ont_AttributeElevationFunction />
</coin:Ont_Attribute>
<coin:Ont_Attribute rdf:ID="netWeight">
  <coin:Ont_AttributeName>netWeight</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#product" />
  <coin:Ont_AttributeTo rdf:resource="#weight" />
  <coin:Ont_AttributeElevationFunction />
</coin:Ont_Attribute>
<coin:Ont_Attribute rdf:ID="grossWeight">
  <coin:Ont_AttributeName>grossWeight</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#product" />
  <coin:Ont_AttributeTo rdf:resource="#weight" />
  <coin:Ont_AttributeElevationFunction />
</coin:Ont_Attribute>
<coin:Ont_Attribute rdf:ID="sleepingCapacity">
  <coin:Ont_AttributeName>sleepingCapacity</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#tent" />
  <coin:Ont_AttributeTo rdf:resource="#sleepingCapacity" />
  <coin:Ont_AttributeElevationFunction />
</coin:Ont_Attribute>
<coin:Ont_Attribute rdf:ID="usage">
  <coin:Ont_AttributeName>usage</coin:Ont_AttributeName>
  <coin:Ont_AttributeFrom rdf:resource="#tent" />
  <coin:Ont_AttributeTo rdf:resource="#usage" />
  <coin:Ont_AttributeElevationFunction />
</coin:Ont_Attribute>
<coin:Ont_Attribute rdf:ID="packedSize">
```

```
    <coin:Ont_AttributeName>packedSize</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#tent" />
    <coin:Ont_AttributeTo rdf:resource="#volume" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="floorArea">
    <coin:Ont_AttributeName>floorArea</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#tent" />
    <coin:Ont_AttributeTo rdf:resource="#area" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="interiorSpace">
    <coin:Ont_AttributeName>interiorSpace</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#tent" />
    <coin:Ont_AttributeTo rdf:resource="#volume" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="length">
    <coin:Ont_AttributeName>length</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#area" />
    <coin:Ont_AttributeTo rdf:resource="#length" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="width">
    <coin:Ont_AttributeName>width</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#area" />
    <coin:Ont_AttributeTo rdf:resource="#length" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="length">
    <coin:Ont_AttributeName>length</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#volume" />
    <coin:Ont_AttributeTo rdf:resource="#length" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="width">
    <coin:Ont_AttributeName>width</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#volume" />
    <coin:Ont_AttributeTo rdf:resource="#length" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <coin:Ont_Attribute rdf:ID="height">
    <coin:Ont_AttributeName>height</coin:Ont_AttributeName>
    <coin:Ont_AttributeFrom rdf:resource="#volume" />
    <coin:Ont_AttributeTo rdf:resource="#length" />
    <coin:Ont_AttributeElevationFunction />
   </coin:Ont_Attribute>
   <!--Modifiers-->
   <coin:Ont_Modifier rdf:ID="currency">
    <coin:Ont_ModifierName>currency</coin:Ont_ModifierName>
    <coin:Ont_ModifierFrom rdf:resource="#monetaryValue" />
    <coin:Ont_ModifierTo rdf:resource="#basic" />

<coin:Ont_ModifierConversionFunction>convfunc|lib_currency|monetaryValue|currency</coin:Ont_Modi
fierConversionFunction>
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_antarctica" />
```

```
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_efunctional" />
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_outback" />
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_terranova" />
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_us" />
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_cuba" />
    <coin:Ont_ModifierContextValues rdf:resource="#currencyc_uk" />
  </coin:Ont_Modifier>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_antarctica">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_antarctica" />
    <coin:Ont_ModifierStaticValue>GBP</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_efunctional">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_efunctional" />
    <coin:Ont_ModifierStaticValue>USD</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_outback">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_outback" />
    <coin:Ont_ModifierStaticValue>USD</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_terranova">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_terranova" />
    <coin:Ont_ModifierStaticValue>GBP</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_us">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_us" />
    <coin:Ont_ModifierStaticValue>USD</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_cuba">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_cuba" />
    <coin:Ont_ModifierStaticValue>CUP</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="currencyc_uk">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_uk" />
    <coin:Ont_ModifierStaticValue>GBP</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_Modifier rdf:ID="weightUnit">
    <coin:Ont_ModifierName>weightUnit</coin:Ont_ModifierName>
    <coin:Ont_ModifierFrom rdf:resource="#weight" />
    <coin:Ont_ModifierTo rdf:resource="#basic" />

<coin:Ont_ModifierConversionFunction>convfunc|lib_physical_unit|weight|weightUnit</coin:Ont_Modifi
erConversionFunction>
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_antarctica" />
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_efunctional" />
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_outback" />
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_terranova" />
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_us" />
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_cuba" />
    <coin:Ont_ModifierContextValues rdf:resource="#weightUnitc_uk" />
  </coin:Ont_Modifier>
  <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_antarctica">
```

```
    <coin:Ont_ModifierContext
 rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_antarctica" />
    <coin:Ont_ModifierStaticValue>kg</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_efunctional">
    <coin:Ont_ModifierContext
 rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_efunctional" />
    <coin:Ont_ModifierStaticValue>lb</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_outback">
    <coin:Ont_ModifierContext
 rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_outback" />
    <coin:Ont_ModifierStaticValue>lb</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_terranova">
    <coin:Ont_ModifierContext
 rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_terranova" />
    <coin:Ont_ModifierStaticValue>kg</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_us">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_us" />
    <coin:Ont_ModifierStaticValue>lb</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_cuba">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_cuba" />
    <coin:Ont_ModifierStaticValue>kg</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="weightUnitc_uk">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_uk" />
    <coin:Ont_ModifierStaticValue>kg</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_Modifier rdf:ID="lengthUnit">
    <coin:Ont_ModifierName>lengthUnit</coin:Ont_ModifierName>
    <coin:Ont_ModifierFrom rdf:resource="#length" />
    <coin:Ont_ModifierTo rdf:resource="#basic" />

<coin:Ont_ModifierConversionFunction>convfunc|lib_physical_unit|length|lengthUnit</coin:Ont_Modifie
rConversionFunction>
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_antarctica" />
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_efunctional" />
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_outback" />
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_terranova" />
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_us" />
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_cuba" />
    <coin:Ont_ModifierContextValues rdf:resource="#lengthUnitc_uk" />
   </coin:Ont_Modifier>
   <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_antarctica">
    <coin:Ont_ModifierContext
 rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_antarctica" />
    <coin:Ont_ModifierStaticValue>cm</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
   <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_efunctional">
    <coin:Ont_ModifierContext
 rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_efunctional" />
    <coin:Ont_ModifierStaticValue>in</coin:Ont_ModifierStaticValue>
   </coin:Ont_ModifierContextValuePair>
```

```
  <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_outback">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_outback" />
    <coin:Ont_ModifierStaticValue>in</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_terranova">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_terranova" />
    <coin:Ont_ModifierStaticValue>cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_us">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_us" />
    <coin:Ont_ModifierStaticValue>in</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_cuba">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_cuba" />
    <coin:Ont_ModifierStaticValue>cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="lengthUnitc_uk">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_uk" />
    <coin:Ont_ModifierStaticValue>cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_Modifier rdf:ID="numPersons">
    <coin:Ont_ModifierName>numPersons</coin:Ont_ModifierName>
    <coin:Ont_ModifierFrom rdf:resource="#sleepingCapacity" />
    <coin:Ont_ModifierTo rdf:resource="#basic" />
    <coin:Ont_ModifierConversionFunction>convfunc|rule(cvt(commutative, sleepingCapacity, _O,
numPersons, ctxt, Ms, Vs, Mt, Vt),
          ({substring(Ms, 1, 5, Fv1), substring(Ms, 7, 5, Fv2), substring(Ms, 12, 2, Fu), substring(Mt, 1, 5,
Tv1), substring(Mt, 7, 5, Tv2), substring(Mt, 12, 2, Tu)},
          unit_conv_p(Fu2, Tu2, Uf),
          value(Fu2, ctxt, Fu),
          value(Tu2, ctxt, Tu),
          value(Uf, ctxt, Ufv),
          multiply(Fv1, Ufv, Fv11),
          multiply(Fv2, Ufv, Fv21),
          multiply(Fv11, Fv21, Fv3),
          multiply(Vs, Fv3, Ttv),
          divide(Ttv, Tv1, Vt1),
          divide(Vt1, Tv2, Vt))).|sleepingCapacity|numPersons</coin:Ont_ModifierConversionFunction>
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_antarctica" />
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_efunctional" />
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_outback" />
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_terranova" />
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_us" />
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_cuba" />
    <coin:Ont_ModifierContextValues rdf:resource="#numPersonsc_uk" />
  </coin:Ont_Modifier>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_antarctica">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_antarctica" />
    <coin:Ont_ModifierStaticValue>122.0x244.0cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_efunctional">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_efunctional" />
```

```
    <coin:Ont_ModifierStaticValue>48.00x96.00in</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_outback">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_outback" />
    <coin:Ont_ModifierStaticValue>48.00x96.00in</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_terranova">
    <coin:Ont_ModifierContext
rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_terranova" />
    <coin:Ont_ModifierStaticValue>122.0x244.0cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_us">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_us" />
    <coin:Ont_ModifierStaticValue>48.00x96.00in</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_cuba">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_cuba" />
    <coin:Ont_ModifierStaticValue>91.00x213.0cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
  <coin:Ont_ModifierContextValuePair rdf:ID="numPersonsc_uk">
    <coin:Ont_ModifierContext rdf:resource="http://localhost/appEditor/apps/rdf/context395.rdf#c_uk" />
    <coin:Ont_ModifierStaticValue>122.0x244.0cm</coin:Ont_ModifierStaticValue>
  </coin:Ont_ModifierContextValuePair>
</rdf:RDF>
```

context395.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
  <!--Contexts-->
  <coin:Cxt_Context rdf:ID="c_uk">
    <coin:Cxt_ContextName>c_uk</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_us">
    <coin:Cxt_ContextName>c_us</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_cuba">
    <coin:Cxt_ContextName>c_cuba</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_terranova">
    <coin:Cxt_ContextName>c_terranova</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_efunctional">
    <coin:Cxt_ContextName>c_efunctional</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_antarctica">
    <coin:Cxt_ContextName>c_antarctica</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="c_outback">
    <coin:Cxt_ContextName>c_outback</coin:Cxt_ContextName>
  </coin:Cxt_Context>
  <coin:Cxt_Context rdf:ID="ctxt">
    <coin:Cxt_ContextName>ctxt</coin:Cxt_ContextName>
  </coin:Cxt_Context>
```

```
</rdf:RDF>
```

source395.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
 <!--Relations-->
 <coin:Src_Relation rdf:ID="dsr_tent_antarctica">
  <coin:Src_RelationName>dsr_tent_antarctica</coin:Src_RelationName>
  <coin:Src_RelationImport>true</coin:Src_RelationImport>
  <coin:Src_RelationExport>true</coin:Src_RelationExport>
  <coin:Src_RelationSourceName>oracle</coin:Src_RelationSourceName>
  <coin:Src_RelationUnsupportedOps />
 </coin:Src_Relation>
 <coin:Src_Relation rdf:ID="dsr_tent_efunctional">
  <coin:Src_RelationName>dsr_tent_efunctional</coin:Src_RelationName>
  <coin:Src_RelationImport>true</coin:Src_RelationImport>
  <coin:Src_RelationExport>true</coin:Src_RelationExport>
  <coin:Src_RelationSourceName>oracle</coin:Src_RelationSourceName>
  <coin:Src_RelationUnsupportedOps />
 </coin:Src_Relation>
 <coin:Src_Relation rdf:ID="dsr_tent_outback">
  <coin:Src_RelationName>dsr_tent_outback</coin:Src_RelationName>
  <coin:Src_RelationImport>true</coin:Src_RelationImport>
  <coin:Src_RelationExport>true</coin:Src_RelationExport>
  <coin:Src_RelationSourceName>oracle</coin:Src_RelationSourceName>
  <coin:Src_RelationUnsupportedOps />
 </coin:Src_Relation>
 <coin:Src_Relation rdf:ID="dsr_tent_terranova">
  <coin:Src_RelationName>dsr_tent_terranova</coin:Src_RelationName>
  <coin:Src_RelationImport>true</coin:Src_RelationImport>
  <coin:Src_RelationExport>true</coin:Src_RelationExport>
  <coin:Src_RelationSourceName>oracle</coin:Src_RelationSourceName>
  <coin:Src_RelationUnsupportedOps />
 </coin:Src_Relation>
 <coin:Src_Relation rdf:ID="tent_usage_map">
  <coin:Src_RelationName>tent_usage_map</coin:Src_RelationName>
  <coin:Src_RelationImport>true</coin:Src_RelationImport>
  <coin:Src_RelationExport>false</coin:Src_RelationExport>
  <coin:Src_RelationSourceName>oracle</coin:Src_RelationSourceName>
  <coin:Src_RelationUnsupportedOps />
 </coin:Src_Relation>
 <!--Columns-->
 <coin:Src_Column rdf:ID="Name">
  <coin:Src_ColumnName>Name</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_antarctica" />
 </coin:Src_Column>
 <coin:Src_Column rdf:ID="Net_weight">
  <coin:Src_ColumnName>Net_weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_antarctica" />
 </coin:Src_Column>
```

```
<coin:Src_Column rdf:ID="Gross_weight">
  <coin:Src_ColumnName>Gross_weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_antarctica" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Interior">
  <coin:Src_ColumnName>Interior</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_antarctica" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Price">
  <coin:Src_ColumnName>Price</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_antarctica" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Maker">
  <coin:Src_ColumnName>Maker</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Model">
  <coin:Src_ColumnName>Model</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Name">
  <coin:Src_ColumnName>Name</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Seasons">
  <coin:Src_ColumnName>Seasons</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Sleeps">
  <coin:Src_ColumnName>Sleeps</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Minimum_weight">
  <coin:Src_ColumnName>Minimum_weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Floor_area">
  <coin:Src_ColumnName>Floor_area</coin:Src_ColumnName>
```

```
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Price">
  <coin:Src_ColumnName>Price</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_efunctional" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Maker">
  <coin:Src_ColumnName>Maker</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Model">
  <coin:Src_ColumnName>Model</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Season">
  <coin:Src_ColumnName>Season</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Capacity">
  <coin:Src_ColumnName>Capacity</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Trail_weight">
  <coin:Src_ColumnName>Trail_weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Packed_weight">
  <coin:Src_ColumnName>Packed_weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Floor_area">
  <coin:Src_ColumnName>Floor_area</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Price">
  <coin:Src_ColumnName>Price</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
```

```
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_outback" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Model">
  <coin:Src_ColumnName>Model</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Usage">
  <coin:Src_ColumnName>Usage</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Persons">
  <coin:Src_ColumnName>Persons</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Weight">
  <coin:Src_ColumnName>Weight</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Packed_length">
  <coin:Src_ColumnName>Packed_length</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Packed_width">
  <coin:Src_ColumnName>Packed_width</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Interior_length">
  <coin:Src_ColumnName>Interior_length</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Interior_width">
  <coin:Src_ColumnName>Interior_width</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="Interior_height">
  <coin:Src_ColumnName>Interior_height</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
```

```
<coin:Src_Column rdf:ID="Price">
  <coin:Src_ColumnName>Price</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#dsr_tent_terranova" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="FromUsage">
  <coin:Src_ColumnName>FromUsage</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#tent_usage_map" />
</coin:Src_Column>
<coin:Src_Column rdf:ID="ToUsage">
  <coin:Src_ColumnName>ToUsage</coin:Src_ColumnName>
  <coin:Src_ColumnType>string</coin:Src_ColumnType>
  <coin:Src_ColumnKeyMember>false</coin:Src_ColumnKeyMember>
  <coin:Src_ColumnRelation rdf:resource="#tent_usage_map" />
</coin:Src_Column>
<!--Elevated Relations-->
<coin:Src_ElevatedRelation rdf:ID="dsr_tent_antarctica_p">
  <coin:Src_ElevatedRelationName>dsr_tent_antarctica_p</coin:Src_ElevatedRelationName>
  <coin:Src_ElevatedRelationRelation rdf:resource="#dsr_tent_antarctica" />
  <coin:Src_ElevatedRelationContext rdf:resource="#c_antarctica" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_antarctica_pName" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_antarctica_pNet_weight" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_antarctica_pGross_weight" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_antarctica_pInterior" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_antarctica_pPrice" />
</coin:Src_ElevatedRelation>
<coin:Src_ElevatedRelation rdf:ID="dsr_tent_efunctional_p">
  <coin:Src_ElevatedRelationName>dsr_tent_efunctional_p</coin:Src_ElevatedRelationName>
  <coin:Src_ElevatedRelationRelation rdf:resource="#dsr_tent_efunctional" />
  <coin:Src_ElevatedRelationContext rdf:resource="#c_efunctional" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pMaker" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pModel" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pName" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pSeasons" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pSleeps" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pMinimum_weight" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pFloor_area" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_efunctional_pPrice" />
</coin:Src_ElevatedRelation>
<coin:Src_ElevatedRelation rdf:ID="dsr_tent_outback_p">
  <coin:Src_ElevatedRelationName>dsr_tent_outback_p</coin:Src_ElevatedRelationName>
  <coin:Src_ElevatedRelationRelation rdf:resource="#dsr_tent_outback" />
  <coin:Src_ElevatedRelationContext rdf:resource="#c_outback" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pMaker" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pModel" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pSeason" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pCapacity" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pTrail_weight" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pPacked_weight" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pFloor_area" />
  <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_outback_pPrice" />
</coin:Src_ElevatedRelation>
<coin:Src_ElevatedRelation rdf:ID="dsr_tent_terranova_p">
```

```
    <coin:Src_ElevatedRelationName>dsr_tent_terranova_p</coin:Src_ElevatedRelationName>
    <coin:Src_ElevatedRelationRelation rdf:resource="#dsr_tent_terranova" />
    <coin:Src_ElevatedRelationContext rdf:resource="#c_terranova" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pModel" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pUsage" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pPersons" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pWeight" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pPacked_length" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pPacked_width" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pInterior_length" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pInterior_width" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pInterior_height" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#dsr_tent_terranova_pPrice" />
  </coin:Src_ElevatedRelation>
  <coin:Src_ElevatedRelation rdf:ID="tent_usage_map_p">
    <coin:Src_ElevatedRelationName>tent_usage_map_p</coin:Src_ElevatedRelationName>
    <coin:Src_ElevatedRelationRelation rdf:resource="#tent_usage_map" />
    <coin:Src_ElevatedRelationContext rdf:resource="#ctxt" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#tent_usage_map_pFromUsage" />
    <coin:Src_ElevatedRelationColumns rdf:resource="#tent_usage_map_pToUsage" />
  </coin:Src_ElevatedRelation>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_antarctica_pName">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Name" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#product" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_antarctica_pNet_weight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Net_weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_antarctica_pGross_weight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Gross_weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_antarctica_pInterior">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Interior" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#volume" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_antarctica_pPrice">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Price" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#price" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pMaker">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Maker" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#company" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pModel">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Model" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#product" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
```

```
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pName">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Name" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#product" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pSeasons">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Seasons" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#usage" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pSleeps">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Sleeps" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#sleepingCapacity" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair
rdf:ID="dsr_tent_efunctional_pMinimum_weight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Minimum_weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pFloor_area">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Floor_area" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#area" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_efunctional_pPrice">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Price" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#price" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pMaker">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Maker" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#company" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pModel">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Model" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#product" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pSeason">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Season" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#usage" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pCapacity">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Capacity" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#sleepingCapacity" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pTrail_weight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Trail_weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
```

```
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pPacked_weight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Packed_weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pFloor_area">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Floor_area" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#area" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_outback_pPrice">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Price" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#price" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pModel">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Model" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#product" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pUsage">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Usage" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#usage" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pPersons">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Persons" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#sleepingCapacity" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pWeight">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Weight" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#weight" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pPacked_length">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Packed_length" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#length" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pPacked_width">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Packed_width" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#length" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pInterior_length">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Interior_length" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#length" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pInterior_width">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Interior_width" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#length" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pInterior_height">
```

```
    <coin:Src_ElevatedRelationColumn rdf:resource="#Interior_height" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#length" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="dsr_tent_terranova_pPrice">
    <coin:Src_ElevatedRelationColumn rdf:resource="#Price" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#price" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="tent_usage_map_pFromUsage">
    <coin:Src_ElevatedRelationColumn rdf:resource="#FromUsage" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
  <coin:Src_ElevatedRelationColumnSemanticTypePair rdf:ID="tent_usage_map_pToUsage">
    <coin:Src_ElevatedRelationColumn rdf:resource="#ToUsage" />
    <coin:Src_ElevatedRelationSemanticType
rdf:resource="http://localhost/appEditor/apps/rdf/ontology395.rdf#basic" />
  </coin:Src_ElevatedRelationColumnSemanticTypePair>
</rdf:RDF>
```

misc395.rdf

```
<?xml version="1.0"?>
<!--prolog engine version 1-->
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:coin="http://localhost/appEditor/apps/rdf/coin_schema#">
  <!--Miscellaneous-->
</rdf:RDF>
```

I.2 Prolog

```
%% Ontology
rule(is_a(basic,basic),(true)).
rule(is_a(product,basic),(true)).
rule(is_a(company,basic),(true)).
rule(is_a(tent,product),(true)).
rule(is_a(monetaryValue,basic),(true)).
rule(is_a(price,monetaryValue),(true)).
rule(is_a(physicalMeasurement,basic),(true)).
rule(is_a(weight,physicalMeasurement),(true)).
rule(is_a(length,physicalMeasurement),(true)).
rule(is_a(area,physicalMeasurement),(true)).
rule(is_a(volume,physicalMeasurement),(true)).
rule(is_a(sleepingCapacity,basic),(true)).
rule(is_a(usage,basic),(true)).
rule(attributes(basic,[]),(true)).
rule(attributes(product,[maker,price,netWeight,grossWeight]),(true)).
rule(attributes(company,[]),(true)).
rule(attributes(tent,[sleepingCapacity,usage,packedSize,floorArea,inter
iorSpace]),(true)).
rule(attributes(monetaryValue,[]),(true)).
rule(attributes(price,[]),(true)).
rule(attributes(physicalMeasurement,[]),(true)).
rule(attributes(weight,[]),(true)).
rule(attributes(length,[]),(true)).
rule(attributes(area,[length,width]),(true)).
rule(attributes(volume,[length,width,height]),(true)).
rule(attributes(sleepingCapacity,[]),(true)).
rule(attributes(usage,[]),(true)).
rule(modifiers(basic,[]),(true)).
rule(modifiers(product,[]),(true)).
rule(modifiers(company,[]),(true)).
rule(modifiers(tent,[]),(true)).
rule(modifiers(monetaryValue,[currency]),(true)).
rule(modifiers(price,[]),(true)).
rule(modifiers(physicalMeasurement,[]),(true)).
rule(modifiers(weight,[weightUnit]),(true)).
rule(modifiers(length,[lengthUnit]),(true)).
rule(modifiers(area,[]),(true)).
rule(modifiers(volume,[]),(true)).
rule(modifiers(sleepingCapacity,[numPersons]),(true)).
rule(modifiers(usage,[]),(true)).

%% Relations
rule(relation(oracle,dsr_tent_antarctica,ie,[['Name',string],['Net_weig
ht',string],['Gross_weight',string],['Interior',string],['Price',string
]],cap([[0,0,0,0,0]],[])),(true)).
rule(relation(oracle,dsr_tent_efunctional,ie,[['Maker',string],['Model'
,string],['Name',string],['Seasons',string],['Sleeps',string],['Minimum
_weight',string],['Floor_area',string],['Price',string]],cap([[0,0,0,0,
0,0,0,0]],[])),(true)).
rule(relation(oracle,dsr_tent_outback,ie,[['Maker',string],['Model',str
ing],['Season',string],['Capacity',string],['Trail_weight',string],['Pa
cked_weight',string],['Floor_area',string],['Price',string]],cap([[0,0,
0,0,0,0,0,0]],[])),(true)).
```

```
rule(relation(oracle,dsr_tent_terranova,ie,[['Model',string],['Usage',s
tring],['Persons',string],['Weight',string],['Packed_length',string],['
Packed_width',string],['Interior_length',string],['Interior_width',stri
ng],['Interior_height',string],['Price',string]],cap([[0,0,0,0,0,0,0,0,
0,0]],[])),(true)).
rule(relation(oracle,tent_usage_map,i,[['FromUsage',string],['ToUsage',
string]],cap([[0,0]],[])),(true)).

%% elevations
rule(dsr_tent_antarctica_p(skolem(product,Name,c_antarctica,1,dsr_tent_
antarctica(Name,Net_weight,Gross_weight,Interior,Price)),skolem(weight,
Net_weight,c_antarctica,2,dsr_tent_antarctica(Name,Net_weight,Gross_wei
ght,Interior,Price)),skolem(weight,Gross_weight,c_antarctica,3,dsr_tent
_antarctica(Name,Net_weight,Gross_weight,Interior,Price)),skolem(volume
,Interior,c_antarctica,4,dsr_tent_antarctica(Name,Net_weight,Gross_weig
ht,Interior,Price)),skolem(price,Price,c_antarctica,5,dsr_tent_antarcti
ca(Name,Net_weight,Gross_weight,Interior,Price))),(dsr_tent_antarctica(
Name,Net_weight,Gross_weight,Interior,Price))).

rule(dsr_tent_efunctional_p(skolem(company,Maker,c_efunctional,1,dsr_te
nt_efunctional(Maker,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_are
a,Price)),skolem(product,Model,c_efunctional,2,dsr_tent_efunctional(Mak
er,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,Price)),skolem(p
roduct,Name,c_efunctional,3,dsr_tent_efunctional(Maker,Model,Name,Seaso
ns,Sleeps,Minimum_weight,Floor_area,Price)),skolem(usage,Seasons,c_efun
ctional,4,dsr_tent_efunctional(Maker,Model,Name,Seasons,Sleeps,Minimum_
weight,Floor_area,Price)),skolem(sleepingCapacity,Sleeps,c_efunctional,
5,dsr_tent_efunctional(Maker,Model,Name,Seasons,Sleeps,Minimum_weight,F
loor_area,Price)),skolem(weight,Minimum_weight,c_efunctional,6,dsr_tent
_efunctional(Maker,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,
Price)),skolem(area,Floor_area,c_efunctional,7,dsr_tent_efunctional(Mak
er,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,Price)),skolem(p
rice,Price,c_efunctional,8,dsr_tent_efunctional(Maker,Model,Name,Season
s,Sleeps,Minimum_weight,Floor_area,Price))),(dsr_tent_efunctional(Maker
,Model,Name,Seasons,Sleeps,Minimum_weight,Floor_area,Price))).

rule(dsr_tent_outback_p(skolem(company,Maker,c_outback,1,dsr_tent_outba
ck(Maker,Model,Season,Capacity,Trail_weight,Packed_weight,Floor_area,Pr
ice)),skolem(product,Model,c_outback,2,dsr_tent_outback(Maker,Model,Sea
son,Capacity,Trail_weight,Packed_weight,Floor_area,Price)),skolem(usage
,Season,c_outback,3,dsr_tent_outback(Maker,Model,Season,Capacity,Trail_
weight,Packed_weight,Floor_area,Price)),skolem(sleepingCapacity,Capacit
y,c_outback,4,dsr_tent_outback(Maker,Model,Season,Capacity,Trail_weight
,Packed_weight,Floor_area,Price)),skolem(weight,Trail_weight,c_outback,
5,dsr_tent_outback(Maker,Model,Season,Capacity,Trail_weight,Packed_weig
ht,Floor_area,Price)),skolem(weight,Packed_weight,c_outback,6,dsr_tent_
outback(Maker,Model,Season,Capacity,Trail_weight,Packed_weight,Floor_ar
ea,Price)),skolem(area,Floor_area,c_outback,7,dsr_tent_outback(Maker,Mo
del,Season,Capacity,Trail_weight,Packed_weight,Floor_area,Price)),skole
m(price,Price,c_outback,8,dsr_tent_outback(Maker,Model,Season,Capacity,
Trail_weight,Packed_weight,Floor_area,Price))),(dsr_tent_outback(Maker,
Model,Season,Capacity,Trail_weight,Packed_weight,Floor_area,Price))).

rule(dsr_tent_terranova_p(skolem(product,Model,c_terranova,1,dsr_tent_t
erranova(Model,Usage,Persons,Weight,Packed_length,Packed_width,Interior
_length,Interior_width,Interior_height,Price)),skolem(usage,Usage,c_ter
ranova,2,dsr_tent_terranova(Model,Usage,Persons,Weight,Packed_length,Pa
```

```
cked_width,Interior_length,Interior_width,Interior_height,Price)),skole
m(sleepingCapacity,Persons,c_terranova,3,dsr_tent_terranova(Model,Usage
,Persons,Weight,Packed_length,Packed_width,Interior_length,Interior_wid
th,Interior_height,Price)),skolem(weight,Weight,c_terranova,4,dsr_tent_
terranova(Model,Usage,Persons,Weight,Packed_length,Packed_width,Interio
r_length,Interior_width,Interior_height,Price)),skolem(length,Packed_le
ngth,c_terranova,5,dsr_tent_terranova(Model,Usage,Persons,Weight,Packed
_length,Packed_width,Interior_length,Interior_width,Interior_height,Pri
ce)),skolem(length,Packed_width,c_terranova,6,dsr_tent_terranova(Model,
Usage,Persons,Weight,Packed_length,Packed_width,Interior_length,Interio
r_width,Interior_height,Price)),skolem(length,Interior_length,c_terrano
va,7,dsr_tent_terranova(Model,Usage,Persons,Weight,Packed_length,Packed
_width,Interior_length,Interior_width,Interior_height,Price)),skolem(le
ngth,Interior_width,c_terranova,8,dsr_tent_terranova(Model,Usage,Person
s,Weight,Packed_length,Packed_width,Interior_length,Interior_width,Inte
rior_height,Price)),skolem(length,Interior_height,c_terranova,9,dsr_ten
t_terranova(Model,Usage,Persons,Weight,Packed_length,Packed_width,Inter
ior_length,Interior_width,Interior_height,Price)),skolem(price,Price,c_
terranova,10,dsr_tent_terranova(Model,Usage,Persons,Weight,Packed_lengt
h,Packed_width,Interior_length,Interior_width,Interior_height,Price))),
(dsr_tent_terranova(Model,Usage,Persons,Weight,Packed_length,Packed_wid
th,Interior_length,Interior_width,Interior_height,Price))).

rule(tent_usage_map_p(skolem(basic,FromUsage,ctxt,1,tent_usage_map(From
Usage,ToUsage)),skolem(basic,ToUsage,ctxt,2,tent_usage_map(FromUsage,To
Usage))),(tent_usage_map(FromUsage,ToUsage))).

%% Context
rule(is_a(c_uk,basic),(true)).
rule(is_a(c_us,basic),(true)).
rule(is_a(c_cuba,basic),(true)).
rule(is_a(c_terranova,basic),(true)).
rule(is_a(c_efunctional,basic),(true)).
rule(is_a(c_antarctica,basic),(true)).
rule(is_a(c_outback,basic),(true)).
rule(is_a(ctxt,basic),(true)).

%% Modifier
rule(modifier(monetaryValue,_O,currency,c_antarctica,M),(cste(basic,M,c
_antarctica,GBP))).
rule(modifier(monetaryValue,_O,currency,c_efunctional,M),(cste(basic,M,
c_efunctional,USD))).
rule(modifier(monetaryValue,_O,currency,c_outback,M),(cste(basic,M,c_ou
tback,USD))).
rule(modifier(monetaryValue,_O,currency,c_terranova,M),(cste(basic,M,c_
terranova,GBP))).
rule(modifier(monetaryValue,_O,currency,c_us,M),(cste(basic,M,c_us,USD)
)).
rule(modifier(monetaryValue,_O,currency,c_cuba,M),(cste(basic,M,c_cuba,
CUP))).
rule(modifier(monetaryValue,_O,currency,c_uk,M),(cste(basic,M,c_uk,GBP)
)).
rule(modifier(weight,_O,weightUnit,c_antarctica,M),(cste(basic,M,c_anta
rctica,kg))).
rule(modifier(weight,_O,weightUnit,c_efunctional,M),(cste(basic,M,c_efu
nctional,lb))).
```

```
rule(modifier(weight,_O,weightUnit,c_outback,M),(cste(basic,M,c_outback
,lb))).
rule(modifier(weight,_O,weightUnit,c_terranova,M),(cste(basic,M,c_terra
nova,kg))).
rule(modifier(weight,_O,weightUnit,c_us,M),(cste(basic,M,c_us,lb))).
rule(modifier(weight,_O,weightUnit,c_cuba,M),(cste(basic,M,c_cuba,kg)))
.
rule(modifier(weight,_O,weightUnit,c_uk,M),(cste(basic,M,c_uk,kg))).
rule(modifier(length,_O,lengthUnit,c_antarctica,M),(cste(basic,M,c_anta
rctica,cm))).
rule(modifier(length,_O,lengthUnit,c_efunctional,M),(cste(basic,M,c_efu
nctional,in))).
rule(modifier(length,_O,lengthUnit,c_outback,M),(cste(basic,M,c_outback
,in))).
rule(modifier(length,_O,lengthUnit,c_terranova,M),(cste(basic,M,c_terra
nova,cm))).
rule(modifier(length,_O,lengthUnit,c_us,M),(cste(basic,M,c_us,in))).
rule(modifier(length,_O,lengthUnit,c_cuba,M),(cste(basic,M,c_cuba,cm)))
.
rule(modifier(length,_O,lengthUnit,c_uk,M),(cste(basic,M,c_uk,cm))).
rule(modifier(sleepingCapacity,_O,numPersons,c_antarctica,M),(cste(basi
c,M,c_antarctica,122.0x244.0cm))).
rule(modifier(sleepingCapacity,_O,numPersons,c_efunctional,M),(cste(bas
ic,M,c_efunctional,48.00x96.00in))).
rule(modifier(sleepingCapacity,_O,numPersons,c_outback,M),(cste(basic,M
,c_outback,48.00x96.00in))).
rule(modifier(sleepingCapacity,_O,numPersons,c_terranova,M),(cste(basic
,M,c_terranova,122.0x244.0cm))).
rule(modifier(sleepingCapacity,_O,numPersons,c_us,M),(cste(basic,M,c_us
,48.00x96.00in))).
rule(modifier(sleepingCapacity,_O,numPersons,c_cuba,M),(cste(basic,M,c_
cuba,91.00x213.0cm))).
rule(modifier(sleepingCapacity,_O,numPersons,c_uk,M),(cste(basic,M,c_uk
,122.0x244.0cm))).

%% Attribute
rule(attr(X,maker,Y),()).
rule(attr(X,price,Y),()).
rule(attr(X,netWeight,Y),()).
rule(attr(X,grossWeight,Y),()).
rule(attr(X,sleepingCapacity,Y),()).
rule(attr(X,usage,Y),()).
rule(attr(X,packedSize,Y),()).
rule(attr(X,floorArea,Y),()).
rule(attr(X,interiorSpace,Y),()).
rule(attr(X,length,Y),()).
rule(attr(X,width,Y),()).
rule(attr(X,length,Y),()).
rule(attr(X,width,Y),()).
rule(attr(X,height,Y),()).

%% Converison Function
%%
%% lib_currency
%%
rule(is_a(lib_currency_exchangeRate, basic), (true)).
rule(is_a(lib_currency_date, basic), (true)).
```

```
rule(is_a(lib_currency_currency, basic), (true)).
rule(relation(cameleon, lib_currency_olsen, i, [['Exchanged', string],
['Expressed', string], ['Rate', string], ['Date', string]],
cap([[0,0,0,0]], [])), (true)).
rule(lib_currency_olsen_p(
skolem(lib_currency_currency, Exch, Ctxt, 1, lib_currency_olsen(Exch,
Express, Rate, Date)),
skolem(lib_currency_currency, Express, Ctxt, 2,
lib_currency_olsen(Exch, Express, Rate, Date)),
skolem(lib_currency_exchangeRate, Rate, Ctxt, 3,
lib_currency_olsen(Exch, Express, Rate, Date)),
skolem(lib_currency_date, Date, Ctxt, 4, lib_currency_olsen(Exch,
Express, Rate, Date))),
(lib_currency_olsen(Exch, Express, Rate, Date))).
rule(lib_currency_currentDate_p(
skolem(lib_currency_date, V, Ctxt, 1, lib_currency_currentDate(V))),
(lib_currency_currentDate(V))).
rule(cvt(commutative, monetaryValue, Object, currency, Ctxt, Ms, Vs,
Mt, Vt),
(lib_currency_olsen_p(Fc, Tc, Rate, Date),value(Fc, Ctxt, Ms),value(Tc,
Ctxt, Mt),value(Rate, Ctxt,
Rv),lib_currency_currentDate_p(CurDate),value(CurDate, Ctxt,
DateValue),value(Date, Ctxt, DateValue),multiply(Vs, Rv, Vt))).
rule(lib_currency_currentDate(Date), ({date(D), substring(D, 5, 3,
Month), substring(D, 9, 2, Day), substring(D, 23, 2, Year)},
month(Month, NumMonth), {concat_string([NumMonth, /, Day, /, Year],
Date)})).
rule(month("Jan", 01), (true)).
rule(month("Feb", 02), (true)).
rule(month("Mar", 03), (true)).
rule(month("Apr", 04), (true)).
rule(month("May", 05), (true)).
rule(month("Jun", 06), (true)).
rule(month("Jul", 07), (true)).
rule(month("Aug", 08), (true)).
rule(month("Sep", 09), (true)).
rule(month("Oct", 10), (true)).
rule(month("Nov", 11), (true)).
rule(month("Dec", 12), (true)).
%%
%% lib_currency end
%%



%%
%% lib_physical_unit
%%
rule(relation(cameleon, lib_physical_unit, i,
[['FrmUnit',string],['ToUnit',string],['UnitFactor',string]],
cap([[0,0,0]],[])), (true)).
rule(lib_physical_unit_p(
skolem(basic, FrmUnit, Ctxt, 1, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor)),
skolem(basic, ToUnit, Ctxt, 2, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor)),
```

```
skolem(basic, UnitFactor, Ctxt, 3, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor))),
(lib_physical_unit(FrmUnit, ToUnit, UnitFactor))).
rule(cvt(commutative, weight, Object, weightUnit, Ctxt, Ms, Vs, Mt,
Vt),
(lib_physical_unit_p(Fu, Tu, Uf),value(Fu, Ctxt, Ms),value(Tu, Ctxt,
Mt),value(Uf, Ctxt, Ufv),multiply(Vs, Ufv, Vt))).
%%
%% lib_physical_unit end
%%



%%
%% lib_physical_unit
%%
rule(relation(cameleon, lib_physical_unit, i,
[['FrmUnit',string],['ToUnit',string],['UnitFactor',string]],
cap([[0,0,0]],[])), (true)).
rule(lib_physical_unit_p(
skolem(basic, FrmUnit, Ctxt, 1, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor)),
skolem(basic, ToUnit, Ctxt, 2, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor)),
skolem(basic, UnitFactor, Ctxt, 3, lib_physical_unit(FrmUnit, ToUnit,
UnitFactor))),
(lib_physical_unit(FrmUnit, ToUnit, UnitFactor))).
rule(cvt(commutative, length, Object, lengthUnit, Ctxt, Ms, Vs, Mt,
Vt),
(lib_physical_unit_p(Fu, Tu, Uf),value(Fu, Ctxt, Ms),value(Tu, Ctxt,
Mt),value(Uf, Ctxt, Ufv),multiply(Vs, Ufv, Vt))).
%%
%% lib_physical_unit end
%%


rule(cvt(commutative, sleepingCapacity, _0, numPersons, ctxt, Ms, Vs,
Mt, Vt),
      ({substring(Ms, 1, 5, Fv1), substring(Ms, 7, 5, Fv2),
substring(Ms, 12, 2, Fu), substring(Mt, 1, 5, Tv1), substring(Mt, 7, 5,
Tv2), substring(Mt, 12, 2, Tu)},
      unit_conv_p(Fu2, Tu2, Uf),
      value(Fu2, ctxt, Fu),
      value(Tu2, ctxt, Tu),
      value(Uf, ctxt, Ufv),
      multiply(Fv1, Ufv, Fv11),
      multiply(Fv2, Ufv, Fv21),
      multiply(Fv11, Fv21, Fv3),
      multiply(Vs, Fv3, Ttv),
      divide(Ttv, Tv1, Vt1),
      divide(Vt1, Tv2, Vt))).
```