Fall 2010

Representation and validation of domain and range restrictions in a relational database driven ontology maintenance system

Patrick Garrett. Edgett

Department:

REPRESENTATION AND VALIDATION OF DOMAIN AND RANGE RESTRICTIONS IN A RELATIONAL DATABASE DRIVEN ONTOLOGY MAINTENANCE SYSTEM

by

PATRICK GARRETT EDGETT

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2010

Approved by

Dr. Jennifer L. Leopold, Advisor
Dr. Dan Lin
Dr. Ronald L. Frank

ABSTRACT

An ontology can be used to represent and organize the objects, properties, events, processes, and relations that embody an area of reality [1]. These knowledge bases may be created manually (by individuals or groups), and/or automatically using software tools, such as those developed for information retrieval and data mining. Recently, the National Science Foundation funded a large collaborative development project for the semi-automated construction of an ontology of amphibian anatomy (AmphibAnat [2]). To satisfy the extensive community curation requirements of that project, a generic, Web-based, multi-user, relational database ontology management system (RDBOM [3]) was constructed, based upon a novel theoretical ontology model called an Ontology Abstract Machine (OAM [4]). The need to support concurrent data entry by multiple users with different levels of access privileges (as determined and assigned by the administrators), made it critical to ensure that the entered data were semantically correct. In particular, the ability to define and enforce restrictions on property characteristics such as the domain and range of a relation provide several advantages. It helps to identify inconsistencies in the ontology, maintain a higher level of overall integrity, and avoid erroneous conclusions that could be made by automated reasoners. In this thesis a modified OAM model is presented that includes definitions for property characteristics and the associated validation algorithms. As proof of concept, it is shown how this modified abstract model has been implemented for domain and range restrictions in RDBOM.

ACKNOWLEDGMENTS

I would like to thank Dr. Jennifer Leopold for getting me involved in research as an undergraduate, and encouraging me to obtain a Master of Science Degree along the same line of work. Additionally I want to thank Dr. Dan Lin and Dr. Ronald Frank for serving as members of my committee. This work was supported by the National Science Foundation under award DBI-0640053, and I am also extremely grateful for the Chancellor's Fellowship provided by Missouri S&T. I also want to thank my cat Horace and my sister's dog Floyd for serving as excellent examples and particularly Floyd for not eating Horace. Lastly I want to thank Leong Lee and Alton Coalter for their support throughout my work on RDBOM and designing the domain and range restrictions.

TABLE OF CONTENTS

Page

ABSTRACT iii
ACKNOWLEDGMENTS iv
LIST OF ALGORITHMS vii
LIST OF EXAMPLES viii
LIST OF ILLUSTRATIONS ix
LIST OF SCHEMAS x
NOMENCLATURE xi
SECTION
1. INTRODUCTION 1
2. BACKGROUND AND MOTIVATION 3
2.1. PROPERTY CHARACTERISTICS 3
2.2. DOMAIN AND RANGE 4
2.3. ONTOLOGY ABSTRACT MACHINE 7
2.4. RESTRICTION CAPABILITIES OF ONTOLOGIES REPRESENTED AS RELATIONAL DATABASES 10
2.5. SUMMARY 11
3. MODIFIED ONTOLOGY ABSTRACT MACHINE 12
3.1. MODIFIED OAM MODEL 12
3.2. MODIFIED OAM MODEL EXAMPLES 13
3.3. ALGORITHMS TO VALIDATE RELATIONSHIP EDGES 15
3.4. SUMMARY 22
4. RDBOM'S STRUCTURE 23
4.1. REPRESENTATION OF THE ONTOLOGY 23
4.2. ONTOLOGY MODIFICATION 25
4.3. USER AUTHORIZATION 26
4.4. SUMMARY 27
5. PROPOSED SOLUTION 28
5.1. PROPERTY CHARACTERISTICS 28
5.2. VALIDATING RESTRICTED PROPERTY CHARACTERISTIC USE 29

5.3. VIOLATING RESTRICTED PROPERTY CHARACTERISTICS 32
5.4. SUMMARY 35
6. IMPLEMENTATION 37
6.1. PROPERTY CHARACTERISTICS 37
6.2. VALIDATING RESTRICTED PROPERTY CHARACTERISTICS 38
6.2.1. Linking a Node to Another Node 38
6.2.2. Moving Nodes/Branches 38
6.2.3. Linking to Additional Parents 40
6.3. RESTRICTED PROPERTY CHARACTERISTIC VIOLATIONS 41
6.3.1. Node to Node Link Operations 42
6.3.2. Move/Cut and Link to Additional Parents 43
6.4. ENTIRE ONTOLOGY VALIDATION 44
6.5. INFORMATIONAL PAGES 46
6.6. SUMMARY 49
7. FUTURE WORK 50
8. SUMMARY 51
BIBLIOGRAPHY 52
VITA 54

LIST OF ALGORITHMS

Item Page

Algorithm 3.1. Create a New Domain 16

Algorithm 3.2. Create a New Range 17

Algorithm 3.3. Perform a Validation Check Before Creating a New Edge 19

LIST OF EXAMPLES

Item		Page
Example 2.1.	OAM Representation of the Ontology Presented In Figure 2.1	8
Example 3.1.	Example of the Modified OAM Representation of an Ontology	14
Example 3.2.	Create a New Range *(is defined by, literature)*	18
Example 3.3.	Perform a Validation Check (In Terms of Violation Detection) Before Creating a New Edge	20
Example 3.4.	Perform a Validation Check (In Terms of Acceptance) Before Creating a New Edge	21

LIST OF ILLUSTRATIONS

Item		Page
Figure 2.1.	Example of a Domain and Range Restriction in OWL	6
Figure 2.2.	OAM Diagram of the Ontology in Example 2.1	9
Figure 3.1.	OAM Diagram of the Ontology in Example 3.1	15
Figure 3.2.	OAM Diagram of the Ontology in Example 3.4	22
Figure 4.1.	RDBOM Update Page for *Horace*	25
Figure 5.1.	Move/Cut Node Operation in RDBOM	31
Figure 5.2.	Link to Additional Parent Operation in RDBOM	31
Figure 6.1.	RDBOM Detection of Invalid Edge from Figure 2.1	39
Figure 6.2.	RDBOM Attempting a Link Node to Additional Parent Operation	41
Figure 6.3.	RDBOM Fixing a Violation via the Move Operation	45
Figure 6.4.	AmphibAnat Restricted Properties	46
Figure 6.5.	RDBOM Term Information Page for *Floyd*	47
Figure 6.6.	RDBOM Violations Overview Page	48
Figure 6.7.	RDBOM Violations Overview Page for AmphibAnat	49

LIST OF SCHEMAS

Item Page

Schema 5.1. Restriction Violations Table Schema..... 34

NOMENCLATURE

Symbol	Description
M	Ontology Abstract Machine (OAM), $(Q, \sum, \delta, Q_0, F)$
Q	Set of nodes, $Q_c \cup Q_i \cup Q_v$
Q_c	Set of class terms in an ontology
Q_i	Set of instance terms in an ontology
Q_v	Set of values in an ontology
$\sum$	Set of relationship types, $\sum_B \cup \sum_E$
$\sum_B$	Set of base relationship types
$\sum_E$	Set of extended relationship types
δ	Set of triples representing relationships (edges) among nodes
Q_0	Set of source nodes (no incoming $\sum_B$ edge, leaves)
F	Set of root nodes
U	Set of individual user IDs
δ_{domain}	Set of domain restriction tuples
δ_{range}	Set of range restrictions tuples

1. INTRODUCTION

Philosopher Barry Smith has defined an ontology as "…the science of what is, of the kinds and structures of objects, properties, events, processes, and relations in every area of reality. For an information system, an ontology is a representation of some pre-existing domain of reality which: 1) reflects the properties of the objects within its domain in such a way that there obtains a systematic correlation between reality and the representation itself; 2) is intelligible to a domain expert; and 3) is formalized in a way that allows it to support automatic information processing" [1].

A perusal of ontology-related literature in computer science clearly reflects that the research focus and problems relating to ontologies in information systems have changed over the years; the focal point has shifted from the more theoretical ontology issues to problems associated with the actual development and use of ontologies in real-world, large-scale, collaborative applications. As an example of a large ontology project that required the use of modularity and collaborative editing, in 2007 the National Science Foundation funded a project to build a comprehensive ontology of amphibian anatomy (AmphibAnat [2]). After determining that existing ontology editors/servers did not meet all of their needs, the research team designed and constructed a Web-based, multi-user, relational database ontology management system (RDBOM [3]) based on a novel theoretical ontology model called an Ontology Abstract Machine (OAM) [4].

However, the need for multi-user, collaborative ontology development tools is only part of the problem. The *extent* of user participation in developing an ontology (of any size) also is an important consideration. Typically, ontology curation is performed manually by a small number of users using a single-user-based ontology editor such as OBO-Edit [5], Protégé [6], SWOOP [7], or COBrA [8], and the ontology files are maintained with a version control system. When a very limited number of people are modifying an ontology, it may be feasible for the entire process to be overseen by those who are "in charge"; they can exercise some degree of control over the data entry, and make sure that the data adhere to an agreed upon design.

Much larger community-curated ontology projects require automated control over the data entry process, particularly in terms of data validation. As an example of such a

project, the AmphibAnat ontology of amphibian ontology contains over 22,000 terms and approximately 44 relationship types, and is curated by a community of 21 active users from various specific areas of anatomical expertise. Initially, the administrators created a skeleton for the ontology hierarchy, and defined the relationships to be used to link terms. Access to branches of the ontology was assigned to various members of the community to allow for content to be added and updated. It made sense to utilize a database management system to address several multi-user issues, including concurrency control and the restriction of data access privileges.

The success of an extensive project like AmphibAnat was dependent upon the participation of a fairly large number of users. Although the administrators were responsible for major design decisions, they needed to have the ability to enforce such decisions in the open development environment. One such requirement was the ability to restrict the use of term relationships based on characteristics defined with respect to the relationship, in particular domain and range. That would help prevent erroneous inputs, both accidental and malicious, and thereby improve the overall integrity of the ontology. Additionally, it would help clarify the relationship definitions.

Herein is presented a modified abstract ontology model that includes property restrictions and the associated validation algorithms. As proof of concept, it is described how this model has been implemented for domain and range restrictions in the RDBOM relational database driven ontology maintenance system. The modified ontology abstract machine model and its related algorithms are discussed in Section 3. An understanding of the internal structure of RDBOM is covered in Section 4. In Section 5, discussion is focused on introducing restricted property characteristics to RDBOM as well as the need in some situations for the restrictions to be ignored and thus violated. Finally, Section 6 pertains to the actual implementation of restricted properties in RDBOM.

2. BACKGROUND AND MOTIVATION

There are several important concepts regarding ontologies that need to be discussed before presenting the OAM and RDBOM modifications to support property restrictions. The first such subject is what exactly property characteristics are, and how they can be used to check if a property is being used correctly (and consistently) throughout an ontology, as well as how additional information about terms can be inferred through property characteristics. The second subject is how the concepts of data constraints such as domain and range apply to an ontology; specifically, the validation capabilities of ontology languages such as the Web Ontology Language (OWL) and the Open Biological and Biomedical Ontology language (OBO) are compared to the desired functionality of RDBOM's domain and range validation. Lastly, a formal definition and several examples of the original Ontology Abstract Machine are given.

2.1. PROPERTY CHARACTERISTICS

There must be a clear understanding of how the concepts of domain and range apply to ontologies. In an ontology, properties are used to describe the relationships among terms. The relationship itself is regarded as a property of a term. A property characteristic describes additional information about a specific property. OWL facilitates the designation of several property characteristics including transitive and inverse relations [10]. For example, the transitive property characteristic can be used to indicate that if a transitive property relates term A to term B, and relates term B to term C, then term A is related to term C.

A property characteristic such as transitivity can be used to infer additional information about terms in the ontology. Often this is referred to as *reasoning* over information from an ontology. Other types of property characteristics can be used to restrict the values for a particular property. For example, restricting the minimum or maximum numerical values of a property *has_legs* would ensure that every use of the *has_legs* property lies within a certain range. In doing this, *consistency checking* of the property's use ensures that the target value meets the restriction set in place on the

property. In the discussion that follows in this section, it is shown how property characteristics such as domain and range can be used to perform both consistency checking of an ontology as well as how they can be used for reasoning by OWL.

It should be noted that the objective of this work is to implement the *restricted* form of property characteristics, a much more focused type of a property characteristic. It requires additional functionality to account for the validation of property use that has a restricted property characteristic applied to it. As will be discussed in subsequent sections, this work also accommodates possible violations of the restricted property characteristic in order to track and eliminate issues in existing ontologies. Implementing *non-restricted* property characteristics in a relational database driven ontology maintenance system such as RDBOM is not nearly as difficult, and does not satisfy the functionality desired by the end users of RDBOM for the AmphibAnat project; hence, this is the reason for focusing on restricted property characteristics. Additionally, the mechanisms required for restricting a property characteristic can potentially be used to infer additional information on non-restricted property characteristics, as will be discussed in Section 7.

2.2. DOMAIN AND RANGE

Domain and range are both features of popular ontology languages such as OWL (via RDFS) [9] and OBO [5]. Both of those languages use virtually the same definition for domain and range. From [10], *domain* is defined as "if a property P has domain D, then any term T that has a relationship of type P to another term is a subclass of D." This states that "any term that has a relationship of type P to another term is *by definition* a subclass of D." Also from [10], *range* is defined as "if a property P has range R, then any term T that is the target of a relationship of type P is a subclass of R." Equivalently, "any term that is the target of a relationship of type P is *by definition* a subclass of R." Other relationships within the set of properties (such as disjointness) provide additional restrictions that a reasoner could use to determine violations of the integrity of the data.

An example of using domain and range within OWL to restrict the values of a property can be seen in Figure 2.1. The ontology consists of classes for *Animals* and

Food. *Dog* and *Cat* are both subclasses of *Animals*. An object property *eats* is defined with the domain of *Animals* and the range of *Food*. An instance of *Cat* is created with the name of *Horace*, and an instance of *Dog* is created with the name of *Floyd*. *Floyd* is then assigned the relationship *eats Horace*. Since the *Animals* and *Food* classes are disjoint, a reasoner could detect an error about the use of *Horace* as the range of the *eats* relationship for *Floyd*. This implies the use of *eats* is inconsistent with its definition.

If the two terms, *Animals* and *Food* are not disjoint then the behavior is different. For terms used as the domain of property *eats*, one could infer additional information about the term, stating that it is a subclass of *Animals* which is the specified domain of *eats*. Likewise, terms used as the range term of *eats* would be inferred to be subclasses of *Food*. Applying this to the *Floyd eats Horace* relationship in Example 1 without the disjointWith statements, *Floyd* is being used as the domain and is already a subclass of *Animals*, so no additional information is inferred. However, *Horace* is only a subclass of *Animals*. By using *Horace* as the range of a use of the *eats* property an entry could automatically be inferred, stating that *Horace* is also a subclass of *Food*, the range term for *eats*.

```
<owl:ObjectProperty rdf:about="#eats">
   <rdfs:domain rdf:resource="#Animals"/>
   <rdfs:range rdf:resource="#Food"/>
</owl:ObjectProperty>

<owl:Class rdf:about="#Animals">
   <rdfs:subClassOf rdf:resource="&owl;Thing"/>
   <owl:disjointWith rdf:resource="#Food"/>
</owl:Class>

<owl:Class rdf:about="#Cat">
   <rdfs:subClassOf rdf:resource="#Animals"/>
</owl:Class>

<owl:Class rdf:about="#Dog">
   <rdfs:subClassOf rdf:resource="#Animals"/>
</owl:Class>

<owl:Class rdf:about="#Food"/>
```

```
<owl:Class rdf:about="&owl;Thing"/>

<owl:Thing rdf:about="#Horace">
   <rdf:type rdf:resource="#Cat"/>
</owl:Thing>

<owl:Thing rdf:about="#Floyd">
   <rdf:type rdf:resource="#Dog"/>
   <eats rdf:resource="#Horace"/>
</owl:Thing>
```

Figure 2.1. Example of a Domain and Range Restriction in OWL

In a multi-user environment, it can be a challenging task to enforce constraints such as the domain and range restrictions on the values of relationships. In this case, the domain and range concepts provide the *restrictions* to be placed on the data, unlike their specification in OWL or OBO, where they instead are used to *define* the data. Using similar data to that from Figure 2.1, if someone tries to assert that "Floyd eats Horace", instead of defining *Horace* to be a subclass of *Food* (as would be the case for OWL or OBO), the system must detect a violation of the range of *eats* (since *Horace* is not already a subclass of *Food*). Note the distinct differences between OBO/OWL and this philosophy. For OBO/OWL, the object or relation is added to the existing set of data (e.g., *Horace* becomes a subclass of *Food)* and is considered inference of additional information for *Horace*. In a multi-user curation environment the data must already exist (e.g., *Horace* must be defined as a subclass of *Food*) so that the specification of the domain or range causes a validation to occur, and does not cause a new relation to be added.

The functionality desired from the OAM and RDBOM modifications is to accommodate checking for consistent property use throughout the ontology. The ability to detect the invalid property use of *eats* as with OWL in Figure 2.1 should be able to be performed without the explicit disjointWith statements existing in the class definitions in RDBOM.

2.3. ONTOLOGY ABSTRACT MACHINE

As previously mentioned, specification and (automated) enforcement of domain and range restrictions are necessary to ensure the data integrity of large community-curated ontologies. One such ontology, the AmphibAnat project, is based on the OAM model; hence restrictions such as domain and range need to be defined formally as part of that model. The motivations for developing the OAM model, the model definition, and the various algorithms associated with the model are discussed in detail in [4]. Here is provided a brief overview of the basic OAM model before the incorporation of property restrictions is addressed. The formal definition of the OAM is given as follows:

An Ontology Abstract Machine (OAM) is a 5-tuple representation of an ontology, $M = (Q, \sum, \delta, Q_0, F)$, where:

Q: set of nodes; $Q = Q_c \cup Q_i \cup Q_v$

Q_c = set of classes

Q_i = set of instances

Q_v = set of values

$\sum$: set of relationship types

$\sum = \sum_B \cup \sum_E$

$\sum_B$ = set of base relationship types, e.g. *{is_a, part_of}*

$\sum_E$ = set of extended relationship types, e.g. *{is_from_literature, image_ contains, is_from_image, …}*

δ: set of relationships in the form of edges (node, relationship type, node), $Q \times \sum \rightarrow Q$; hence each element is a child node, a relationship type, or a parent node.

Q_0: set of source nodes: These are nodes with no incoming $\sum_B$ edge. This set can be identified from δ. Q_0 is a subset of $(Q_c \cup Q_i)$. Source nodes can only be elements of the set of classes or elements of the set of instances.

F: set of root nodes, i.e. nodes with no outgoing $\sum_B$ edge. e.g. F = *{Concepts}*, F is a subset of Q_c.

Another set $U = \{u_1, u_2, \ldots u_i \ldots u_n\}$ is used to represent individual user ids in

order to implement security features. Any elements of U can be associated with any node in Q.

$\sum_B$ is a set of base relationship types. Members of this set can be used as links (among classes and instances) to generate the main graph view of an ontology; two of the most common base relationship types used for this purpose are *is_a* and *part_of*. Here it is assumed that the main graph view is acyclic.

$\sum_E$ is a set of extended relationship types. The member elements can be used as links between values and classes, or as links between values and instances. In other words, they are used to link attributes to classes and instances. Another required function is the ability to link classes and instances (without affecting the main structure of the ontology).

To better understand the OAM model, an OAM will be developed based off the ontology presented in Figure 2.1. The OAM is described in Example 2.1 and depicted in Figure 2.2.

OAM instance $M = (Q, \sum, \delta, Q_0, F)$:

$Q = Q_c \cup Q_i \cup Q_v = \{Thing, Food, Animals, Dog, Cat, Floyd, Horace\}$;

$Q_c = \{Thing, Food, Animals, Dog, Cat\}$; $Q_i = \{Floyd, Horace\}$;

$Q_v = \{\}$

$\sum = \sum_B \cup \sum_E = \{is\_a, eats\}$

$\sum_B = \{is\_a\}$; $\sum_E = \{eats\}$

$\delta = \{(Food, is\_a, Thing), (Animals, is\_a, Thing), (Dog, is\_a, Animals),$
$(Floyd, is\_a, Dog), (Cat, is\_a, Animals), (Horace, is\_a, Cat)\}$

$Q_0 = \{Food, Floyd, Horace\}$

$F = \{Thing\}$

Example 2.1. OAM Representation of the Ontology Presented In Figure 2.1

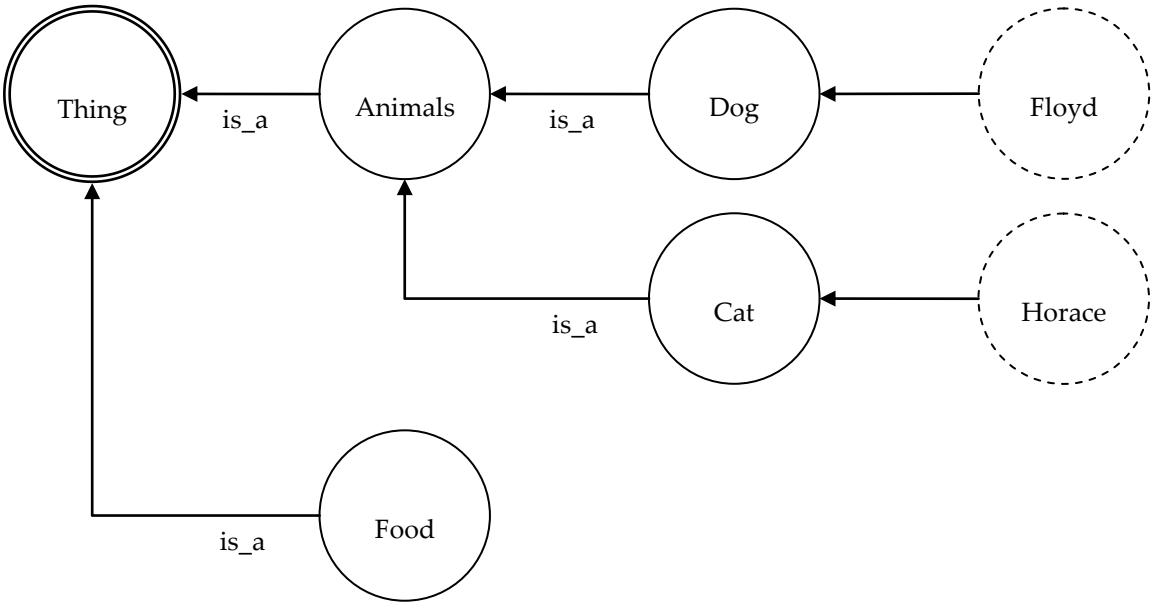

Figure 2.2. OAM Diagram of the Ontology in Example 2.1

As discussed in [4], the OAM model is a generic theoretical model developed to provide a framework for constructing a collaborative, Web-based ontology management system that supports modularity and distinct relationship classifications. For collaboration purposes, it is useful to assign user access rights to a subset of the ontology; for example *(M, Animals)* denotes a subset of *M* [4] (or informally a branch below the *Animals* node). The RDBOM relational database driven ontology management system uses the OAM model to implement the multi-user access control capability [3] [4]. For example, administers can attach a user u_i to the node *Animals*, thereby granting user u_i both access and update rights to the subset *(M, Animals)*, but not to other parts of the ontology *M*.

The original OAM model did not include the capability to identify the domain and range of a relationship, and thereby restrict the classes that can be used with a relationship. For example, referring to Example 2.1, the administrators should be able to control the domain of "$\sum_E$ = *{eats}*" and the range of "$\sum_E$ = *{eats}*"; specifically, the domain of *eats* should be nodes from *(M, Animals)*, and the range of *eats* should be nodes

from *(M, Food)*. If a user tries to create an edge *(Floyd eats Horace)*, the OAM model, and hence the RDBOM system, should give a domain/range violation warning.

2.4. RESTRICTION CAPABILITIES OF ONTOLOGIES REPRESENTED AS RELATIONAL DATABASES

In [11], the importance of formally stating rules regarding relations is emphasized. In an open, multi-user setting, rules (or restrictions) are needed to ensure that the community follows the standards and designs that have been decided upon for the particular ontology. The widely used Protégé application [6] user interface supports consistency checking through FaCT [12], which can be used on the entire ontology or only on a selected part of the ontology. There are also plugins for other reasoners such as Pellet [13].

Ontology development project teams have started to recognize how valuable it is to be able to check the consistency of an ontology. Members of the Gene Ontology project [14] stated that as the size of their ontology increased, the curation of it became much more challenging. As a result, they turned to the use of Protégé, in part for its consistency checking features, to ensure that the knowledge model was being used correctly by its classes and instances.

For some applications, it is useful to convert an ontology into a relational database, as demonstrated in [15]. There exist systems such as Minerva [16] that support persistent storage and inference of OWL ontologies in a relational database. Likewise, DBOWL [17] consists of an OWL relational database storage system, and an OWL reasoning system. Thus, it is possible to store an ontology as a relational database, and to specify OWL-compatible domain and range restrictions, such as those illustrated in Figure 2.1. However, these systems are designed for *persistent storage* of ontologies, and not *dynamic management* of the data contained therein; if changes are made to the data, the database must be reloaded and the consistency checker must be re-executed.

2.5. SUMMARY

This section has provided some background information about the differences between property characteristics, restricted property characteristics, and different ways they can be used within an ontology to perform consistency checking of properties used in relationships as well as inference of additional term data. The OAM model and the RDBOM implementation aim to incorporate restricted property characteristics such as domain and range. This provides a means for checking the consistency of properties and how they are used throughout the ontology, which has shown to be valuable in RDBOM's multi-user curation environment. The original OAM model was introduced and applied to Figure 2.1, as can be found in Example 2.1 and Figure 2.2, in order to provide a basic understanding of how it represents an ontology. In the next section the OAM model will be extended to support restrictions, using domain and range as a proof of concept.

3. MODIFIED ONTOLOGY ABSTRACT MACHINE

The original definition of the OAM model presented in the previous section must be modified to take into consideration restrictions such as domain and range. Those modifications, together with the supporting algorithms for maintenance of that information, are provided in this section. Additionally, several examples are given using a very small section of the Amphibanat ontology to enforce these concepts. Lastly, an algorithm for validating a new edge (relationship) introduced to the OAM is provided along with an example.

3.1. MODIFIED OAM MODEL

Introduced to the model are two new sets, δ_{domain} and δ_{range}. Both of these sets contain tuples of a relationship type, and a node which represents the property to restrict as well as the node to which to restrict the values. That is, all domain and range property restrictions are specified in δ_{domain} and δ_{range}, respectively.

The (modified) Ontology Abstract Machine is a 7-tuple representation of an ontology, $M = (Q, \sum, \delta, Q_0, F, \delta_{domain}, \delta_{range})$, where:

Q: set of nodes; $Q = Q_c \cup Q_i \cup Q_v$

Q_c = set of classes

Q_i = set of instances

Q_v = set of values

$\sum$: set of relationship types

$\sum = \sum_B \cup \sum_E$

$\sum_B$ = set of base relationship types, e.g. *{is_a, part_of}*

$\sum_E$ = set of extended relationship types, e.g. *{is_from_literature, image_ contains, is_from_image, ...}*

δ: set of relationships in the form of edges (node, relationship type, node),

$Q \times \sum \rightarrow Q$; hence each element is a child node, a relationship type, or a parent node.

Q_0: set of source nodes: These are nodes with no incoming $\sum_B$ edge. This set can be identified from δ. Q_0 is a subset of $(Q_c \cup Q_i)$. Source nodes can only be elements of the set of classes or elements of the set of instances.

F: set of root nodes, i.e. nodes with no outgoing $\sum_B$ edge. F is a subset of Q_c.

δ_{domain}: set of meta data in the form of (*relationship type, node*), $\sum \times Q$. This is used to define the domain of a relationship type; it means that the domain of this relationship type can only be nodes from subset *(M, node)*. Normally the relationship type is an element of $\sum_E$, but it also can be an element of $\sum_B$.

δ_{range}: set of meta data in the form of (*relationship type, node*), $\sum \times Q$. This is used to describe the range of a relationship type; it means that the range of this relationship type can only be nodes from subset *(M, node)*. Normally the relationship type is an element of $\sum_E$, but it also can be an element of $\sum_B$.

The descriptions of $U = \{u_1, u_2, \ldots u_i \ldots u_n\}$, $\sum_B$, and $\sum_E$ are the same as their descriptions described in Section 2.3, the original OAM definition.

3.2. MODIFIED OAM MODEL EXAMPLES

Several examples will be given to demonstrate the modifications made to the original OAM model. Example 3.1 illustrates how the (modified) OAM can be used to represent domain and range restrictions.

$Q = Q_c \cup Q_i \cup Q_v$

Q_c = *{Concepts, image, Gaupp_1896_Fig_11, synonym, sternum inferius, literature, Maglia et al. 2007, amphibian anatomical entity, sternum};*

$Q_i = \{\}$; $Q_v = \{\}$

$\sum = \sum_B \cup \sum_E$ = *{is_a, is defined by, has related synonym, image contains}*

$\sum_B$ = *{is_a}*; $\sum_E$ = *{has related synonym, image contains}*

δ = *{(image, is_a, Concepts), (Gaupp_1896_Fig_11, is_a, image),*

(synonym, is_a, Concepts), (sternum inferius, is_a, synonym),
(literature, is_a, Concepts), (Maglia et al. 2007, is_a, literature),
(amphibian anatomical entity, is_a, Concepts),
(sternum, is_a, amphibian anatomical entity),
(Gaupp_1896_Fig_11, image contains, sternum),
(sternum, has related synonym, sternum inferius)}

Q_0 = *{Gaupp_1896_Fig_11, sternum inferius, Maglia et al. 2007, sternum}*

F = {Concepts}

δ_{domain} = *{(image contains, image)}*

δ_{range} = *{(has related synonym, synonym)}*

Example 3.1. Example of the Modified OAM Representation of an Ontology

In Example 3.1, the domain specification of the relationship *has_related_synonym* and range specification for the relationship *image_contains* were not included for brevity. It is assumed that the *image_contains* relationship can be used with any term in the ontology being a valid range term. Likewise, the *has_related_synonym* relationship can correctly be used with any term in the ontology as its range.

Shown in Figure 3.1 is the graphical OAM representation of the ontology in Example 3.1. Note that this is a very simple example with only a few nodes. In the AmphibAnat project the OAM is used to represent ontology modules which each contain thousands of nodes. For the RDBOM implementation, the OAM data are stored in a relational database. However, the OAM model is implementation independent, and could be stored as a flat file or simply stored in system memory.

In Example 3.1 it can be seen that δ_{domain} = *{(image contains, image)}*, which means the domain of relationship type *image contains* can only be nodes from subset *(M, image)*. The edge *(Gaupp_1896_Fig_11, image contains, sternum)* satisfies this requirement because the node *Gaupp_1896_Fig_11* is under *(M, image).*

Similarly, δ_{range} = *{(has related synonym, synonym)}*, means the range of relationship type *has related synonym* can only be nodes from subset *(M, synonym.)* The

edge *(sternum, has related synonym, sternum inferius)* satisfies this requirement because *sternum inferius* is under *(M, synonym)*.

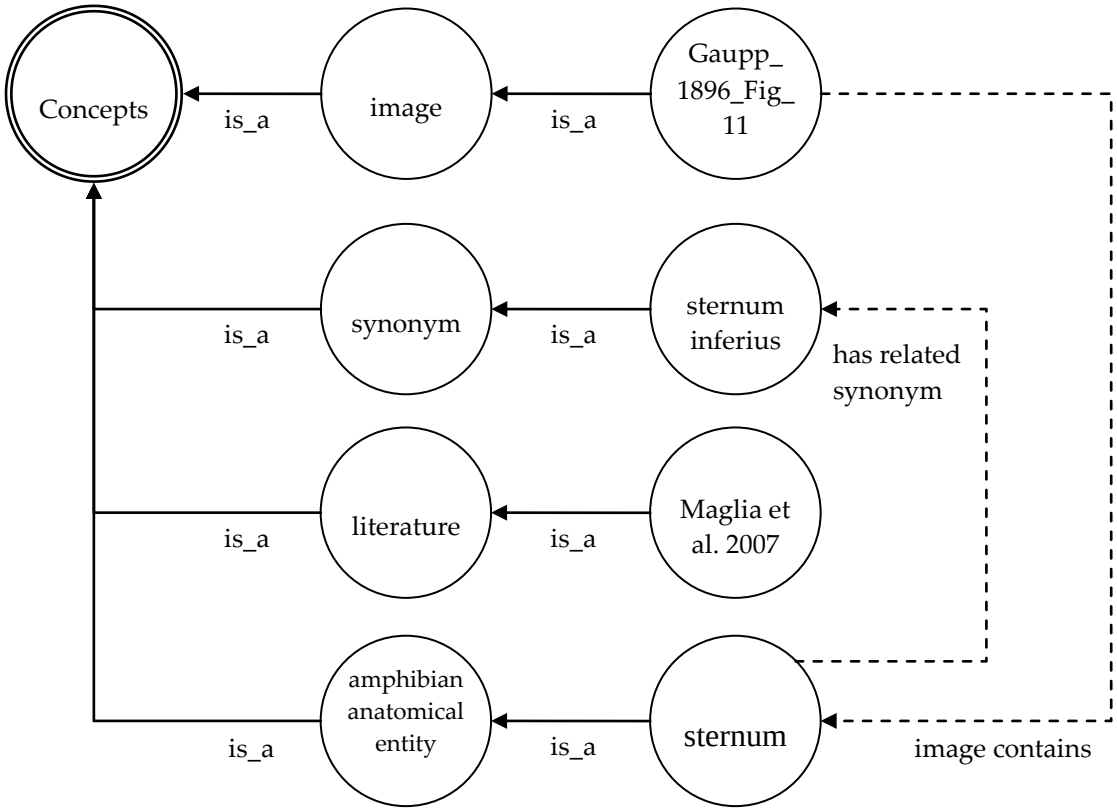

Figure 3.1. OAM Diagram of the Ontology in Example 3.1

3.3. ALGORITHMS TO VALIDATE RELATIONSHIP EDGES

Because the OAM model has been modified to store domain and range property restrictions, an algorithm needs to be provided to enable the addition and enforcement of the restrictions when adding new edges (relationships) to the OAM. To provide the functionality required to specify domain and range restrictions, and perform validation checks, Algorithm 3.1, Algorithm 3.2 and Algorithm 3.3 were developed. An application of Algorithm 3.2 is given in Example 3.2 where a new range restriction *"is defined by* has range *literature"* is created.

Input: An OAM instance, and a new domain definition in the form of *(r, n)*, where *r* is a relationship type and *n* is a class node.

Output: An updated OAM instance.

Algorithm:

if *n* is not an element of Q_c then

    exit and output the error message "class node *n* does not exist"

end if

if *r* is not an element of $\sum$ then

    if *r* should be of base relation type then

        add *r* to set $\sum_B$

    elseif *r* should be of extended relationship type then

        add *r* to set $\sum_E$

    end if/elseif

else

    $\delta_{domain} = \delta_{domain} \cup \{(r, n)\}$

end if/else

Algorithm 3.1. Create a New Domain

Input: An OAM instance, and a new range definition in the form of *(r, n)*, where *r* is a relationship type and *n* is a class node.

Output: An updated OAM instance.

Algorithm:

if *n* is not an element of Q_c then

    exit and output the error message "class node *n* does not exist"

end if

if *r* is not an element of $\sum$ then

    if *r* should be of base relation type then

        add *r* to set $\sum_B$

    elseif *r* should be of extended relationship type then

        add *r* to set $\sum_E$

end if/elseif

end if

$\delta_{range} = \delta_{range} \cup \{(r, n)\}$

Algorithm 3.2. Create a New Range

Input: The OAM instance in Example 3.1, and a new range definition in the form of *(is defined by, literature)*, where *is defined by* is an extended relationship type, and *literature* is a class node.

Steps:

Since *literature* $\in Q_c$; continue

is defined by $\notin \sum$; so add *is defined by* to $\sum_E$ (*is defined by* is an extended relationship type)

now $\sum_E$ = *{has related synonym, image contains, is defined by}*

$\delta_{range} = \delta_{range} \cup \{(is\ defined\ by, literature)\}$

now δ_{range} = *{(has related synonym, synonym), (is defined by, literature)}*

Output: An updated OAM instance as shown below.

OAM instance $M = (Q, \sum, \delta, Q_0, F, \delta_{domain}, \delta_{range})$:

$Q = Q_c \cup Q_i \cup Q_v$

Q_c = *{Concepts, image, Gaupp_1896_Fig_11, synonym, sternum inferius, literature, Maglia et al. 2007, amphibian anatomical entity, sternum};*

$Q_i = \{\}$; $Q_v = \{\}$

$\sum = \sum_B \cup \sum_E$ = *{is_a, is defined by, has related synonym, image contains}*

$\sum_B$ = *{is_a};*

$\sum_E$ = *{has related synonym, image contains, is defined by}*

δ = *{(image, is_a, Concepts), (Gaupp_1896_Fig_11, is_a, image), (synonym, is_a, Concepts), (sternum inferius, is_a, synonym), (literature, is_a, Concepts), (Maglia et al. 2007, is_a, literature), (amphibian anatomical entity, is_a, Concepts), (sternum, is_a, amphibian anatomical entity), (Gaupp_1896_Fig_11, image contains,*

sternum), (sternum, has related synonym, sternum inferius)}

Q_0 *= {Gaupp_1896_Fig_11, sternum inferius, Maglia et al. 2007, sternum}*

F = {Concepts}

δ_{domain} *= {(image contains, image)}*

δ_{range} *= {(has related synonym, synonym), (is defined by, literature)}*

Example 3.2. Create a New Range *(is defined by, literature)*

Note that the only differences between the OAM instances in Example 3.1 and Example 3.2 are the sets $\sum_E$ and δ_{range}. It does not affect the OAM diagram, although the meta-data concerning the range of relationships are changed. As a result, Example 3.2 has the same OAM diagram as Example 3.1, shown in Figure 3.1.

Algorithm 3.3 can be used to validate the domain and range definitions. Here it is assumed that such validation checks normally would be performed before an edge is added to the ontology. Example 3.3 illustrates that the algorithm performs the validation check and detects that this is a domain/range violation; consequently, the edge creation request is rejected. Example 3.4 demonstrates that the algorithm allows the edge creation request, as the request satisfies all domain/range definitions.

Input: An OAM instance, and a new edge in the form of (n_d, r, n_r), where r is a relationship type, and n_d and n_r are nodes.

Output: An updated OAM instance.

Algorithm:

if ($n_d \notin Q$) or ($n_r \notin Q$) or ($r \notin \sum$) then

exit and output error message “class node or relationship type does not exist”

end if

for each element *(r’, n)* in set δ_{domain} do

if ($r' = r$) then

if (check_up (n_d, n) = “not_found”) then //check_up is declared below

exit and output the error message “domain *(r’, n)* violation”

```
            end if
        end if
    end for
    for each element (r', n) in set δ_range do
        if (r' = r) then
            if (check_up (n_r, n) = "not_found") then
                exit and output the error message "range (r', n) violation"
            end if
        end if
    end for
    δ = δ ∪ {(n_d, r, n_r)}  //add edge to OAM

    check_up(test_node, node)
    begin
        for each ancestor path p of test_node:
            if node not in p then
                return "not_found"
            end if
        end for loop

        return "found"
    end check_up
```

Algorithm 3.3. Perform a Validation Check Before Creating a New Edge

The check_up routine included in Algorithm 3.3 is used to validate any domain or range restrictions that apply to a new edge. In OAM it is possible for nodes to have multiple parents. When checking to see if a node can be used in a property with domain or range restrictions, the node must be a descendent of the node to which the property is restricted. Depending on the location of the node being validated, the restricted node could be an ancestor in one path but not another. The restriction violations are used to

perform consistency checking of the relationships using properties, so it is important to ensure that their use is correct in every path of the node.

Input: The OAM instance in Example 3.2, and a new edge *(sternum, image contains, Maglia et al. 2007)* to be added, where *image contains* is a relationship type, and *sternum* and *Maglia et al. 2007* are nodes.

Steps:

(*sternum* ∈ *Q*), (*Maglia et al. 2007* ∈ *Q*), and (*image contains* ∈ ∑)

for element *(image contains, image)* in set δ_{domain}

since (*r' = image contains = r*)

since (check_up (sternum, image) = "not found")

exit and output error message "domain *(image contains, image)* violation"

Output: Since there is a violation, the OAM instance remains unchanged.

Example 3.3. Perform a Validation Check (In Terms of Violation Detection) Before Creating a New Edge

Input: The OAM instance in Example 3.2, and a new edge *(sternum, is defined by, Maglia et al. 2007)* to be added, where *is defined by* is a relationship type, and *sternum* and *Maglia et al. 2007* are nodes.

Steps:

(*sternum* ∈ *Q*) and (*Maglia et al. 2007* ∈ *Q*) and (*is defined by* ∈ ∑)

no element (*r', n*) in set δ_{domain} satisfies (*r' = is defined by*), so it passes the domain check

for element *(is defined by, literature)* in set δ_{range}

since (*r' = is defined by = r*)

(check_up (*Maglia et al. 2007, literature*) = "found"), so it passes range check

$\delta = \delta \cup \{$*(sternum, is defined by, Maglia et al. 2007)*$\}$ //add edge to the OAM instance

$\delta =$ *{(image, is_a, Concepts), (Gaupp_1896_Fig_11, is_a, image), (synonym, is_a, Concepts), (sternum inferius, is_a, synonym), (literature, is_a, Concepts), (Maglia et al. 2007, is_a, literature), (amphibian anatomical entity, is_a, Concepts), (sternum,*

is_a, amphibian anatomical entity), (Gaupp_1896_Fig_11, image contains, sternum), (sternum, has related synonym, sternum inferius), {(sternum, is defined by, Maglia et al. 2007)}

Output: An updated OAM instance shown below. Please refer to Figure 3.2 for the graphical OAM representation of this ontology instance.

OAM instance $M = (Q, \sum, \delta, Q_0, F, \delta_{domain}, \delta_{range})$:

$Q = Q_c \cup Q_i \cup Q_v$

Q_c = *{Concepts, image, Gaupp_1896_Fig_11, synonym, sternum inferius, literature, Maglia et al. 2007, amphibian anatomical entity, sternum};* $Q_i = \{\}$; $Q_v = \{\}$

$\sum = \sum_B \cup \sum_E$ = *{is_a, is defined by, has related synonym, image contains}*

$\sum_B$ = *{is_a};* $\sum_E$ = *{ has related synonym, image contains, is defined by}*

δ = *{(image, is_a, Concepts), (Gaupp_1896_Fig_11, is_a, image), (synonym, is_a, Concepts), (sternum inferius, is_a, synonym), (literature, is_a, Concepts), (Maglia et al. 2007, is_a, literature), (amphibian anatomical entity, is_a, Concepts), (sternum, is_a, amphibian anatomical entity), (Gaupp_1896_Fig_11, image contains, sternum), (sternum, has related synonym, sternum inferius), {(sternum, is defined by, Maglia et al. 2007)}*

Q_0 = *{Gaupp_1896_Fig_11, sternum inferius, Maglia et al. 2007, sternum}*

F = *{Concepts}*

δ_{domain} = *{(image contains, image)}*

δ_{range} = *{(has related synonym, synonym), (is defined by, literature)}*

Example 3.4. Perform a Validation Check (In Terms of Acceptance) Before Creating a New Edge

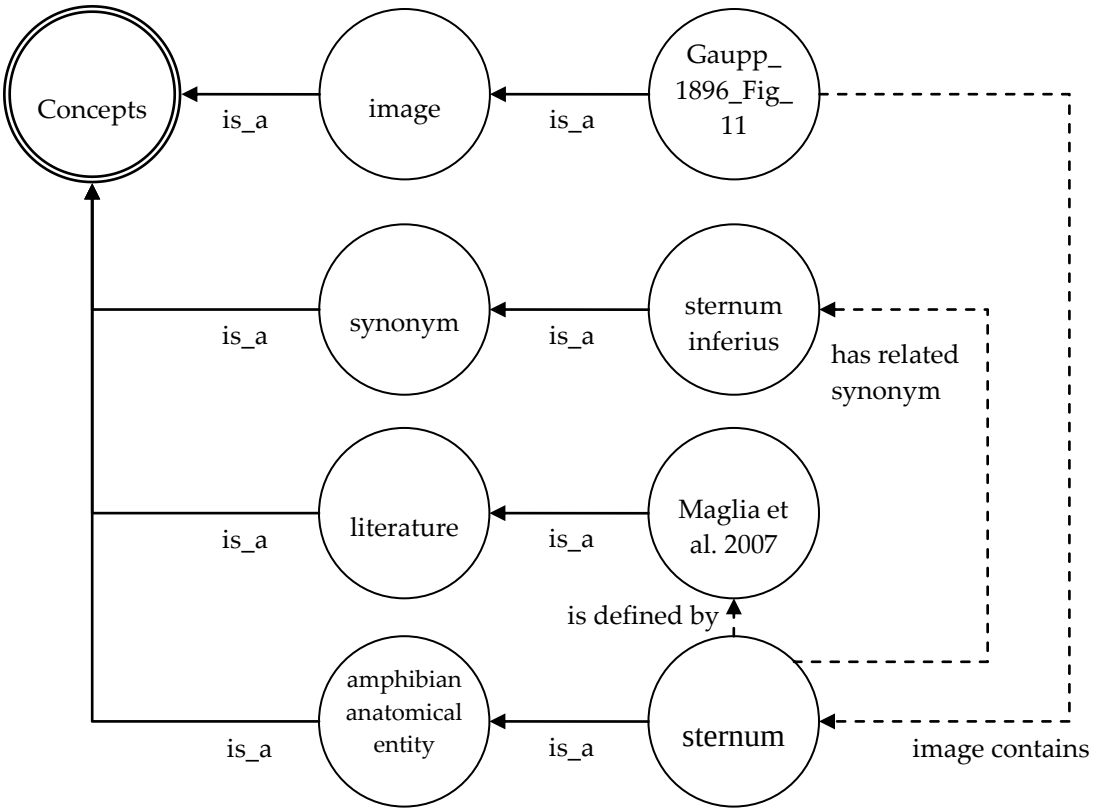

Figure 3.2. OAM Diagram of the Ontology in Example 3.4

3.4. SUMMARY

This section has introduced the necessary modifications to the OAM model to accommodate property restrictions. Domain and range restrictions were used as a proof of concept. However, it is intended that the modified OAM model definition, and restriction maintenance and validation algorithms could be applied to other types of restrictions as well.

In the next section, RDBOM, an implementation of the (original) OAM is presented. Section 5 explains how the OAM modifications for property restrictions were incorporated into RDBOM.

4. RDBOM'S STRUCTURE

RDBOM is a relational database driven ontology maintenance system based on the OAM. In order to efficiently implement the OAM modifications to support property restrictions that were discussed in the previous section, the existing RDBOM relation schemas had to be considered carefully. In this section, a brief overview of the RDBOM system is provided with a focus on: (1) the storage features that needed to be expanded upon to accommodate restricted property characteristics, and (2) the role of user authorization, which plays a part in the enforcement of restrictions. Also briefly covered are the various operations that can be performed to modify the ontology data in RDBOM, and how property restrictions are involved in those functions.

4.1. REPRESENTATION OF THE ONTOLOGY

At the center of RDBOM's data storage is the terms table. The terms table assigns a unique integer identifier to a piece of text. Every piece of information in an ontology is stored as a term in RDBOM. It does not matter what the type of information is, whether it is the name of a class, the name of a property, a definition of a term, the path to an image representing a term, etc. Additionally, the terms table entry also provides the identifier of the ontology with which it is associated.

The term types table gives definitions and identifiers for every type of term that is used in RDBOM. The 'class' term type is used as in OWL, and can be seen in Figure 2.1 to represent the *Food* and *Animal* classes in the ontology. 'Value' term types identify terms being used as class definitions or unique identifiers for classes within the ontology so they can be referenced easily. Instances of classes are denoted with the 'instance' term type (e.g., *Horace* and *Floyd* from Figure 2.1). Any properties (e.g., *eats* from Figure 2.1) are specified with the 'property' term type.

In order to express how a term should be used in the ontology, one or more entries are created for it in the term usages table. The term usages table uses foreign keys to reference a terms table entry and a term types entry. Additionally, for terms defined to be of type 'property' via the term usages table, a corresponding entry in the properties table

is made. The properties table was originally intended to store property characteristics for every property. The table includes an indicator for whether or not the property is considered a 'tree' property, (i.e., *is a* and *part of*), which subsequently determines if that term gets displayed as a node in the hierarchical display of the ontology in the user interface browsing mode. There are other attributes to denote if it a property is transitive, reflexive, symmetric, antisymmetric, or cyclic, as well as an attribute to provide the identifier for an inverse property term (if one exists).

In the original RDBOM schema, the property characteristics (attributes) are nothing more than flags to indicate the characteristic. However, this does support the more generic approach to maintaining property characteristics (namely, those that require some other term to be the property characteristic's value). For example, if the domain and range restricted property characteristics were to be implemented in the same manner, then two more columns would have to be added to the *properties* table to indicate the terms to which they are restricted. Herein, domain and range are being used as proof of concept for a generic implementation of property characteristics. This means for any other property characteristics that would be added to RDBOM in the future, the schema for the *properties* table would need to be modified in the same manner (i.e., adding additional attributes to the RDBOM tables). Additionally, it would not allow for the same property characteristic to be applied to multiple terms for a single property. The user could not restrict a property to multiple classes in the ontology since the *properties* table maintains a 1-to-1 relation between property terms and the property characteristic information.

Another consideration in the incorporation of property restrictions into RDBOM is the existing term2terms table which allows properties to be used to link two terms together in the form of "term1 relation_term term2." In Figure 2.1, the relationship of *Floyd eats Horace* would be represented as term1 referring to *Floyd*, relation_term being *eats* and term2 referencing *Horace*. The use of this table and additional term types entries provides the ability to generically store both restricted and unrestricted property characteristics in the RDBOM schema.

4.2. ONTOLOGY MODIFICATION

The ontology data stored in the RDBOM database is modified through a series of Web pages. All of the commands are initiated by first selecting a class term to modify. From there the user is presented with the list of commands that can be applied to the selected term. The commands are split into two categories, with the first category encompassing operations for modifying the ontology's structure. Figure 4.1 shows the ontology from Figure 2.1 represented in RDBOM with the update page for the term *Horace* selected.

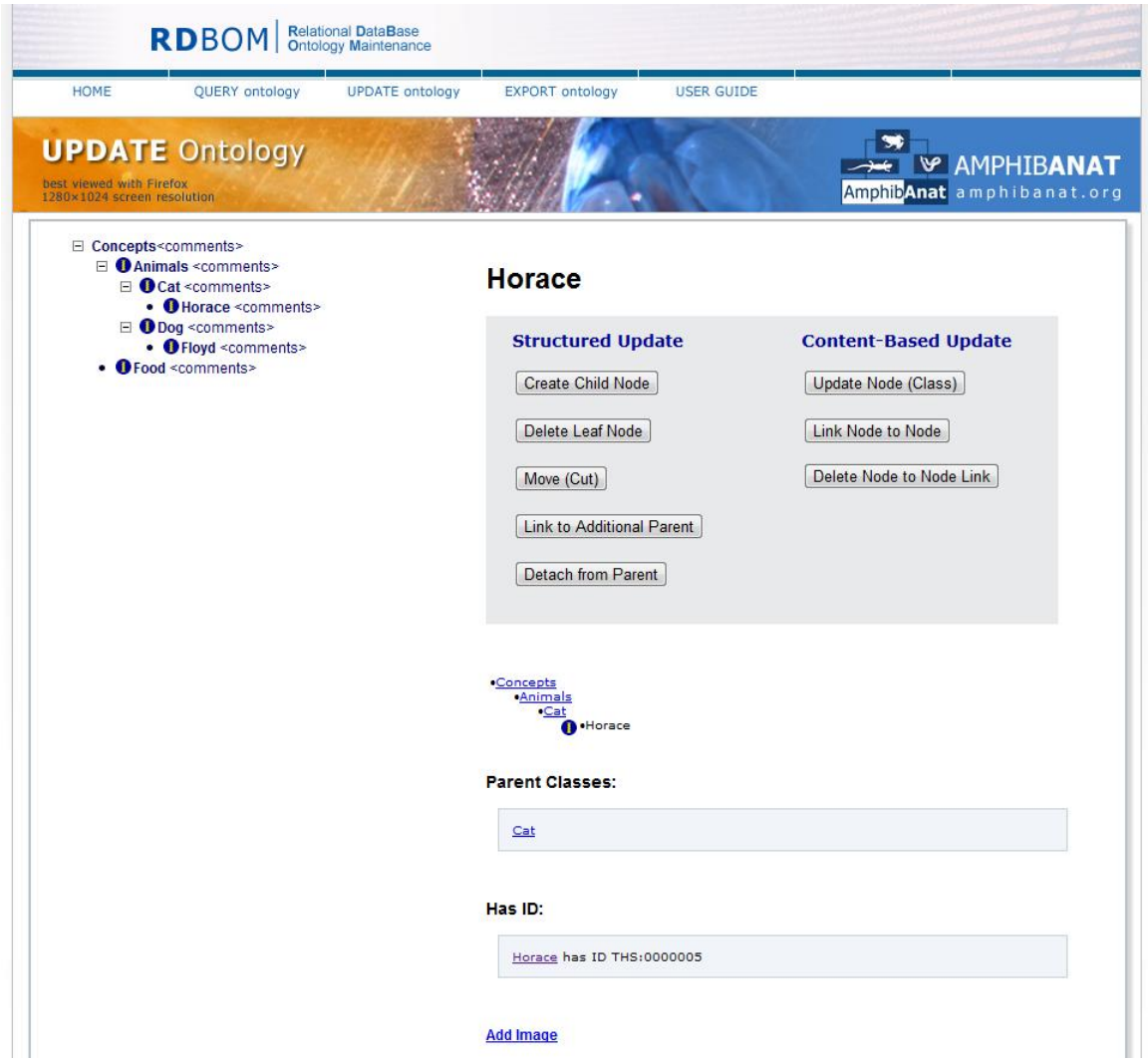

Figure 4.1. RDBOM Update Page for *Horace*

Structure modifications allow for the creation and deletion of child terms, moving (cutting and pasting) a term to a different parent, linking the term to an additional parent, and also detaching the term from a parent. The move operation can be applied to both leaf and non-leaf terms. When used on a non-leaf term, it serves the function of moving a branch of the ontology. Linking a term to an additional parent causes multiple ancestry paths for a term. This is important to note since domain and range restrictions of any property applying to the term must be valid in each path.

The second category of modifications deals with the content of the term. Definitions can be created for the term, and changes can be made to the term's name. Relationships can be created to link two terms together by a property. Every creation of a relationship begins with the domain term. From this term, the property is selected followed by the range term. It also allows users to delete relationships. RDBOM contains an administrative module that allows administrators to create and delete properties in the ontology, grant specific users access to the update pages, and import existing ontologies into RDBOM's database. These descriptions of the various structural and content operations will be used to identify locations requiring changes to accommodate the validation of domain and range restrictions in the next section.

4.3. USER AUTHORIZATION

User authorization is important to understand since ultimately some users will be granted permission to violate restrictions. There are already existing mechanisms allowing users to authenticate with RDBOM by logging in with a user name and password. The only authorization performed prior to this work was used for controlling access to ontology modifications. As previously discussed, ontology data in RDBOM are maintained through a series of update operations.

User accounts and their information are stored in the users table. The authorization levels table provides the various tasks for which the system must authorize users. The 'update' authorization allows users to gain access to the data update operations of an ontology. However, the user might not have access to the entire ontology. Administrators use RDBOM's administrative module to grant the update permission to

users on a case by case basis, giving a user access to update only the parts (or "branches") of the ontology for which the user is regarded as an expert. This is done by granting access to the some class term in the ontology, and allowing update operations to be performed on any class that has the specified term as an ancestor (i.e., so that the user has permission to modify a branch of the ontology).

The permissions granted are stored in the authorizations table which uses foreign key references to identify a user from the users table, a particular permission from the authorization levels table, and a term from the terms table. An entry in the authorizations table can be used to specify a user, the permission the user has, and a branch of the ontology to which the authorization levels entry applies. Multiple authorizations entries are used to identify distinct branches in an ontology that a user has permission to update.

4.4. SUMMARY

The interactions between the main tables used for capturing the information in an ontology in RDBOM have been identified in this section, as well as the mechanisms involved with granting users permission to update various branches of the ontology. Additionally, the operations used to modify the ontology data were identified. These concepts are necessary to understand the design of the modifications to RDBOM that were required in order to support domain and range validation, the topic of the next section.

5. PROPOSED SOLUTION

This section discusses the design of the proposed solution and highlights the parts of RDBOM that need to be modified in order to support restricted property characteristics, specifically for domain and range as proof of concept of the more general case. Details that are specific to each change in RDBOM are explained, including database schema and update operation changes. First the changes required to generically represent any restricted property characteristic are identified. Next discussed are the steps required to validate property use in the ontology when there is a restricted property characteristic specified. Finally, consideration is given for when restricted property characteristics can knowingly be violated by users, and how to deal with recording such violations. The actual implementation of these modifications in RDBOM will then be covered in the following section.

5.1. PROPERTY CHARACTERISTICS

To accommodate restrictions such as domain and range, changes needed to be made to the way property characteristics are stored in RDBOM. As identified in Section 4.1, property characteristics are tightly coupled with the properties table. It restricts the number of property characteristics that can be applied to any given property to a single term or value, and also requires the schema to be altered for any addition of property characteristics. To allow for any arbitrary property characteristics to be represented in RDBOM, two new term types of 'property characteristic' and 'restricted property characteristic' were introduced to the term types table. Property characteristics could then be applied to a property via the term2terms table.

The RDBOM user interface already has full support for defining new relationships, and using them to link classes and instances in an ontology. This functionality was exploited to define the domain and range properties of relationship types. For simplicity, a class, instance, or relationship type in RDBOM will be referred to as a term. Recall that in RDBOM, the relationships between terms (δ in the OAM) are inserted into the term2terms table in the form of (*term1, relationship type, term2*), which

represents the relationship 'term1 has relationship type term2.' Referring to Example 3.1, the domain description (*image contains, image*) is represented in RDBOM by the term2terms entry (*image contains, has domain, image*). Similarly, the range description (*has related synonym, synonym*) can be expressed with the term2terms entry (*has related synonym, has range, synonym*). By doing this, the required information is available to RDBOM to allow Algorithm 3.3 to be applied for validating a relationship. It also should be noted that this can be used to represent other relationship characteristics, both restrictive and non-restrictive, such as cardinalities, inverse of, and symmetric.

The addition of property characteristics necessitates a few modifications to the RDBOM user interface. The update pages requiring validation will be discussed shortly. But now the user must be informed of any property characteristics involved in the domain and range restrictions. Also, the administrative module needs to now also include management of the ontologies restricted property characteristics, domain, and range. This requires providing an interface for administrators to enter any domain or range restrictions that should be placed upon existing properties.

5.2. VALIDATING RESTRICTED PROPERTY CHARACTERISTIC USE

Once there was a way to store restricted property characteristics in the database, it needed to determine which existing parts of RDBOM should be changed to accommodate enforcing the domain and range restrictions. Several of the update pages mentioned in the previous section would require validation. Validation must also occur in the administrative module when a restricted property characteristic is created or an ontology is imported into RDBOM**.** The method in which validation should occur needed to be determined as well, with consideration of any mechanisms already provided by RDBOM's relational database management system that could assist in the process.

One approach that was considered for doing this was the use of SQL triggers. Triggers can be specified for relational database tables to control the actions taken whenever an update, insert, or delete action occurs [18]. For a generic relational database ontology implementation, triggers could be predefined to check transactions that occur upon insertion of relationships between classes, thus performing validation of the

relationship usage. Additionally, they could be used upon updating the domain or range of existing relationship types to make sure that the modified relationship types remain valid. However, this could be a complicated and difficult task to perform on a non-generic database implementation of an ontology.

Another problem in using triggers in this manner arises from the potential need to be able to violate restrictions under certain circumstances. The trigger would become much more complex to specify since it could not simply refuse the transaction, but instead would need to *manage* the violations, with differing actions under differing conditions.

Further, if the domain and range of relationship types are the only restrictions being used in the ontology, not every insert or update transaction on the term relationships may need to be checked. For example, actions such as creating child classes for terms and updating term definitions do not involve domain and range restrictions, but still might activate triggers on the associated tables.

An alternative to database triggers is to perform these tasks in the ontology management system's application logic. In so doing, the actions for a particular validation could more easily be controlled. Validation could still be checked as the classes are accessed or updated, as well as upon demand for the entire ontology. This would eliminate much of the automated management required by triggers (since there is no longer a single point responsible for handling all validation cases). Furthermore, being able to check the entire ontology also would allow for the validation of the domain and range of relationships in imported ontologies.

The main disadvantage to implementing validation in the application logic is that every location where data in an ontology can be modified in some way must be assessed to determine if any validation needs to be performed. Several update operations accessible through the user interface were identified as requiring modification. The first was the Link Node to Node page which allows users to tie two classes together with a property. The second was the Move/Cut Node page that can be used to change the location of a single node or branch in the ontology. Figure 5.1 shows Figure 2.1 loaded into RDBOM before a move operation, the dialog prompted to the user for moving *Horace* to be located under the node *Food*, and then a view of the ontology after the

operation. The red boxes highlight changes in the ontology's structure where *Cat* no longer contains *Horace* as a subclass and *Food* does.

Figure 5.1. Move/Cut Node Operation in RDBOM

The last operation requiring validation was the Link to Additional Parent. When a term is linked to an additional parent a new ancestry path is created for the term since it is present in a new location of the ontology. If the term is involved in any relationships using properties with restricted domains or ranges, it may no longer be considered valid in this new location. The new set of ancestors created by the path to the new parent might not include the terms to which the properties are restricted. In this case, the new path is invalid and should not be allowed. An example of a Link to Additional Parent operation performed in RDBOM without the property restrictions in place can be seen in Figure 5.2 where *Horace* is linked to an additional parent of *Food*. The red box highlights the change to the ontology's structure in which *Horace* is now located under the *Food* term as well as *Cats*.

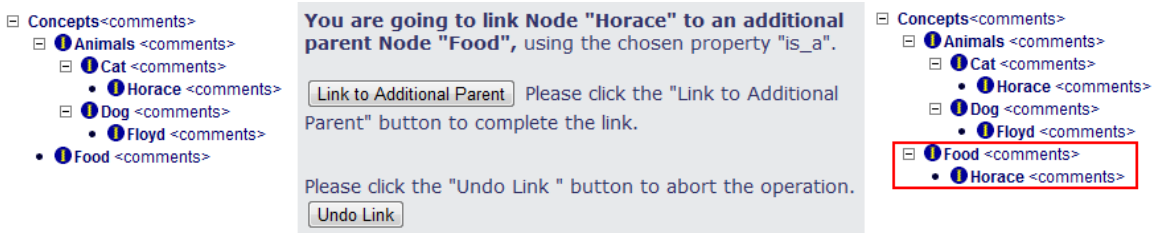

Figure 5.2. Link to Additional Parent Operation in RDBOM

5.3. VIOLATING RESTRICTED PROPERTY CHARACTERISTICS

One issue with restricting the domain and range of a relation lies in giving the users the ability to breach such restrictions. Any action that would violate what is defined as the domain or range for a given relationship type should be considered carefully to determine whether the current ontology design is adequate, especially if the action was not inadvertent. If it is a violation that is intended to be persistent, this suggests that there might be another solution for representing the information; either the ontology design needs to be changed, or the violated relationship needs to be modified to include other domain and/or range classes. Consider Figure 2.1 and 2.2, in which the range of the relationship type *eats* is defined as *Food*. Introducing the relationship (*Floyd eats Horace*) causes a violation of the specified domain for *eats*. Review of this violation suggests that perhaps dogs can in fact eat cats, and that this was not a careless mistake made by the user. By addressing this issue, and perhaps specifying an additional range class of *Cat* to the relation *eats*, or changing the relationship type *eats* to be *eats nonliving*, the clarity of the intended semantics of the ontology would be improved.

Some administrative tasks require the ability to identify all of the violations of the restricted property characteristics. A motivating factor is the application of restriction violations to existing properties that are actively being used for curation in the ontology. There are ontologies already being developed in the RDBOM system, and other existing ontologies easily can be imported into a RDBOM ontology from the OBO ontology file format. Without a way to identify existing violations when a new domain or range restriction is created, there is no easy way for administrators to create the restrictions. If the restrictions were added, then the existing invalid property uses would not be captured, or else the task of correcting all the errors would be entirely the responsibility of the administrators. RDBOM was developed to facilitate community curation of ontologies. Therefore, there needed to be a way for the violations to be presented to the users. This, in turn, would help the administrators in adopting the use of restrictions such as the domain and range since they could rely on the community to see invalid relationships and resolve them; the task would not be solely the responsibility of the administrators.

There also are scenarios in which a restriction *temporarily* must be broken, such as moving entire branches within an ontology; intermediate states may be in violation of

the rules, even though the final state would not contain those violations. An example of such a situation would be restructuring items that are subclasses of *Food* in Examples 2.1 and 2.2. Suppose that a user wants to add new classes under *Food* such as *Meat*, *Fruit*, and *Vegetable*. The relocation of the current classes and instances that have a parent of *Food* to a more specific parent would result in their restriction to *Food* to be broken temporarily.

In some cases, the user community must be able to perform these operations. And perhaps there may be a situation in which an administrator should have the option of intentionally breaking a relationship restricted by domain or range. By making administrators the only users able to violate these restrictions, it allows them to screen the actions of other users; that is, other users would be required to contact an administrator to make any changes that violate a domain or range restriction. Administrators could then either make the change if desired, or alternatively could make decisions on how better to represent the information involved in the violation.

It is possible that some members of the community will be trusted enough by the administrators to make these types of changes. In the current OAM model, users must be assigned rights to a branch of the ontology before they are able to modify it. One reasonable solution for giving a user the ability to violate a domain or range restriction is to add another permission setting (in addition to the branch permission) that would specify whether the user would be allowed to violate a restriction. Administrators then could control not only the branches that a user is able to update, but also the ability of the user to violate restrictions that have been set in place (perhaps based on the user's experience and expertise).

To avoid constantly re-running the validation routines on terms in the ontology each time they are accessed, a new table was introduced to persist the violated restriction information. This reduced the number of times that the validation algorithm has to be executed and improves the performance of the RDBOM system. Violated relationships are initially determined based on Algorithm 3.3, and are then recorded in a dedicated table. After an initial check to verify the data, this provides quicker access to invalid relationship use. That is, upon visiting a term simply for display purposes, any domain or range violations already will have been determined, and recorded in a table.

The schema for the table used to maintain violations is as follows:

```
TABLE restriction_violations (
    id int IDENTITY(1,1),
    violated_term2terms_id int,
    violated_term_usage_id int,
    restricted_term2terms_id int,
    )
```

Schema 5.1. Restriction Violations Table Schema

Here *id* is a unique identifier for the given violation, and *term2terms_id* refers to the unique identifier of the term2terms entry that violates the specified domain or range of a relationship type (where a 'term2terms' entry is in the form of (*term1, relationship type, term2)*). The *violated_term_usage_id* refers to the unique identifier for a term that is stored in the 'term2terms' record. The *term_usage_id* is the identifier used to reference a particular term, in this case *term1* or *term2* from a term2terms entry, in the ontology. Thus, the *violated_term_usage_id* value of the restriction violations entry can be used to determine whether it was the domain of the relationship type that is violated (in which case it would be found in the *term1* location of the term2terms entry), or the range of the relationship type (in which case it would be found in the *term2* location).

Also included in the table is a foreign key reference to the term2term entry that contains the definition of the restricted property characteristic being violated, *restricted_term2terms_id*. Relying solely upon *violated_term2terms_id* and *violated_term_usage_id* provides enough information to determine if the violation occurred due to a domain or range violation and allows for the retrieval of the violating property usage; however, it does not provide enough information to say why the use is invalid without re-running the validation algorithms. It would also make it difficult to uniquely identify multiple domain or range violations that resulted from a single property

use. Including a way to immediately look up the violated restriction gives a quick way to inform the user why the relationship is used incorrectly.

The foreign key constraints of RDBOM's relational database management system are leveraged to take care of many of the situations where an invalid property use becomes correct. The schema can be set so that any deletions of the referenced *violated_term2terms_id* or *violated_term_usage_id* cascades to the restriction violations table. As a result, all violated entries referencing the id of either a deleted term usage id (which happens when a term is deleted) or the id of a term2term entry that is deleted are automatically removed from the restriction violations table. Because of this, the only situations in which entries must be manually deleted are when the violations are corrected by restricting properties to additional terms (which could make them valid) and when a user moves the class within the ontology.

The cascading deletion cannot apply to the *restricted_term2terms_id* column because it references the term2terms table, as is the case with the *violated_term2terms_id* column. The database engine being used for RDBOM, MS SQL Server, will not allow cascade actions to be performed by two individual references to the same table since it can potentially introduce cycles or multiple cascade paths. Because of this particular implementation limitation, when an administrator deletes domain and range restrictions, the corresponding restriction violations entries also will need to be removed.

5.4. SUMMARY

In this section a design was proposed for representing, checking, and allowing the violation one type of property restriction (namely, domain and range) in RDBOM. In summary, the new term types entry 'restricted property characteristic' was introduced to record the restricted property characteristics domain and range. This would allow for two terms entries, *has domain* and *has range,* to be created and identified as restricted property characteristics. To specify the domain or range of a property *p* to be some term *t*, a term2terms entry of (*p, has domain/range, t)* was created. The locations requiring validation to be performed in RDBOM's update pages were identified, and it was decided that the best way to implement the validation was through RDBOM's application logic.

A new table, restriction violations, was also proposed to record any violations that occur, allowing quick access to the details of the violation.

In the next section, the actual implementation of this design in RDBOM is discussed.

6. IMPLEMENTATION

The actual implementation of the domain and range restricted property characteristics in RDBOM is discussed in this section. Discussion is focused on some of the more interesting details involved in the implementation, as well as the algorithms developed to handle the modification of various parts of RDBOM functionality based on the discussion in the preceding section. First discussed is how the administrative module was modified so domain and range could be applied to restrict existing properties. Then changes to the various update pages are addressed to account for validation of property use. Finally, the intricacies of allowing restrictions to be violated are covered, as well as methods for validating the entire ontology and locations in the RDBOM user interface where the violation information is displayed to the user. Figure 2.1 presented in Section 2 will be used to demonstrate the various implementations in RDBOM.

6.1. PROPERTY CHARACTERISTICS

Implementation of the property characteristics mainly dealt with modifying some administrative pages and creating the two new term types, 'restricted property characteristic' and 'property characteristic'. The existing property administration page was updated to allow administrators to create and delete restricted property characteristics, and apply them to existing properties and remove property characteristics. When creating the required SQL queries, several inconsistencies in the RDBOM data were identified. Seven terms being used as properties were lacking term usage entries to identify the term's type as a property. This had not been noticed yet during the RDBOM development because queries relating to properties performed joins between the terms table and properties table (which is sufficient since each property is required to have a properties table entry). For the sake of consistency with the RDBOM data and to help avoid future development problems that could arise due to looking for properties by a term's usage, corresponding term usage entries were created for the seven missing terms.

6.2. VALIDATING RESTRICTED PROPERTY CHARACTERISTICS

The core validation routines occur in the link node to node operations since only a single relationship is being validated. This routine was developed incrementally by others, starting with a way to validate a relationship, then validation of a term based on its relationships that use restricted properties, and finally the application to branches which validate all of the constituent terms. The following subsections describe how the validation was performed for various RDBOM operations.

6.2.1. Linking a Node to Another Node. Using a property to relate two terms takes part in several steps. First, the user selects the domain term and is presented with a list of properties. The next step involves selecting a property. Upon selecting a property, the user should now be presented with a list of the range restrictions in place on that property as well as any error messages due to the selection of the domain term. After selecting the range term, a confirmation page is displayed summarizing the relationship to be created. On this page it is first determined if the range of the property is valid in the context of any domain or range restrictions in place on the property, and, if not, display a meaningful error message to the user. Otherwise, the user can authorize the creation of the relationship. Implementing a routine validate_relationship was very straightforward and followed Algorithm 3.3 with no difficulties. Recall that in Figure 2.1 the edge *Floyd eats Horace* was attempted to be created. Figure 6.1 demonstrates how RDBOM utilizes Algorithm 3.3 to detect that was an invalid edge and prevents the user from creating it.

6.2.2. Moving Nodes/Branches. The move operation allows the user to relocate an individual term or the root term of entire branch in the ontology. If the selected term is a leaf node, then the validation process needs to evaluate the new ancestry path that would be created upon moving the term. Every relationship the term is involved with that contains restricted properties must be validated. For terms selected to be moved that are actually a root of a branch, this validation must be done for every term in the branch. In order to allow a user to relocate nodes in the ontology, the validity of the properties with domain and range restrictions must be upheld in the new location.

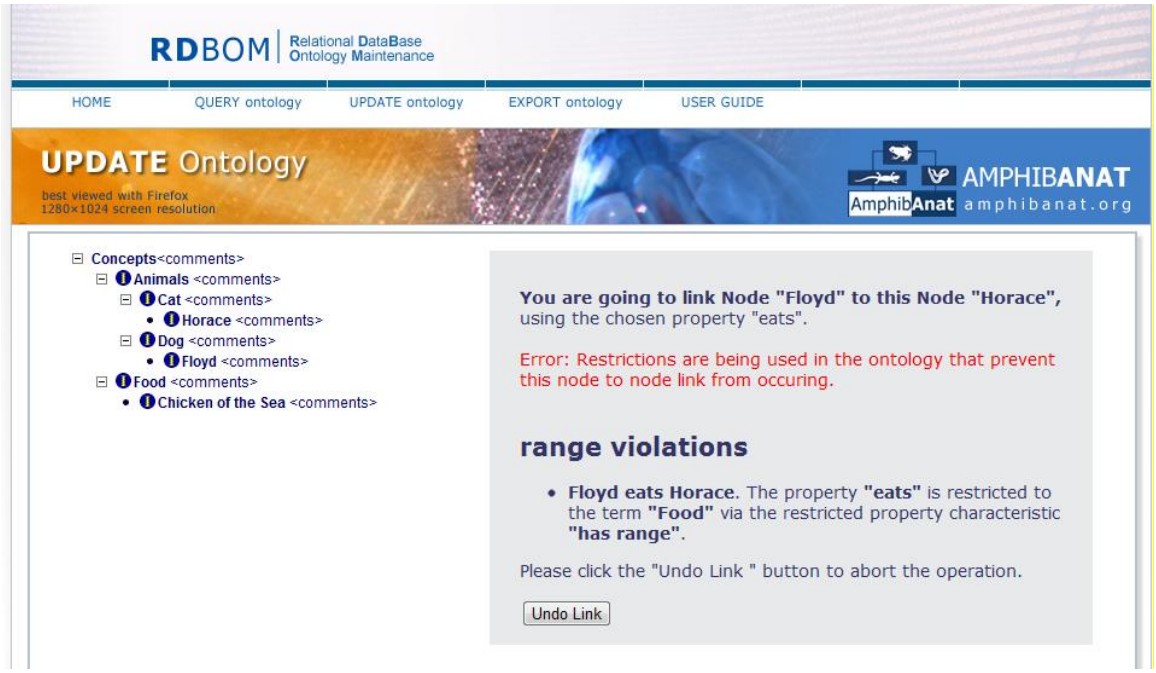

Figure 6.1. RDBOM Detection of Invalid Edge from Figure 2.1

Two methods were considered for performing the validation. The first involved carrying out the move by changing the data in the database. Immediately after this, a validation routine would be executed on every term involved with the move which would validate the domain and range of every property applied to every involved term. If any invalid property use occurred in the new location then the root term could just be moved back to its old location. However, this produces an intermediate state in RDBOM's data where it is not known if the relationships are being used correctly or not. RDBOM's main purpose is to facilitate community-based curation, so it is possible that other users could be working with the data during this event. The advantage is that this method would not require any changes to the way check_up is performed to validate restrictions from Algorithm 3.3 since the data will already be present in the location requiring validation.

The other option, which was ultimately chosen, involves leaving the selected term in place, not moving it, but instead using a modified version of Algorithm 3.3 to validate domain and range terms of relationship edges that already exist. In the modified algorithm, the check_up routine is changed to take information about the current root

term and the new root term. The current root term corresponds to the term selected to be moved in the ontology, and the new root term refers to the new parent the selected term will have. When performing the check_up on each term in the branch, check_up will search through each ancestor path of the term that is being tested until it reaches the current root term. Upon testing the current root term, if the relationship is still invalid it will switch to checking the ancestry of the new root term. It can be thought of as creating a temporary ancestry list corresponding to the term's new location that consists of the term's parents up through the root term of the branch being moved and then appended to the parents of the new root term. This allows any invalid domain and range restrictions to be identified without temporarily moving data in the RDBOM database.

To demonstrate this, imagine moving the branch *Animals* in Figure 2.1 to be a child of the term *Food*. Each term inside the branch (*Cat, Dog, Horace, Floyd)* must be checked in the new location to make sure that properties making use of these terms are still valid. When performing any check_up routines for *Horace,* it begins searching through the ancestry path checking *Horace, Cat,* and then *Animals*. The routine stops searching this ancestry path upon reaching *Animals* since this is the root of the branch being moved. However, it continues searching by starting at the end of *Food*'s ancestry path since this is the location to which the branch will be moved.

6.2.3. Linking to Additional Parents. Validation for linking a node to an additional parent can be done using the same technique as moving a branch. Since only valid paths are allowed, there is no need to validate them again, only the new path introduced by the new parent. The algorithm for checking a branch before it is moved is used with the current root term corresponding to the term being linked to another parent and the new root term being the parent to which to link the term. This will, in turn, validate any necessary relationships of the selected term in the context of the new parent's ancestry path.

Recall in Figure 2.1 how a new edge, *Floyd eats Horace,* was attempted to be introduced to the ontology. It failed because *Horace* is not a subclass of *Food*. Suppose it is determined by the administrators that the ontology really needs to support this relationship. Since RDBOM supports linking nodes to multiple parents, a quick potential solution to address this issue could be to have *Horace* link to an additional parent, *Food*.

Figure 6.2 shows the results. The property *eats* has a domain restriction of *Animals* and the relationship *Horace eats Chicken of the Sea* exists in the ontology. Therefore, the new path to *Horace* that would be introduced (*Concepts->Food->Horace*) contains a domain violation on the *eats* property and is not allowed. This is a good demonstration of how the consistency checking shows that the current class structure of the ontology does not provide the ability to express particular relationships that the administrators feel it should include. It should suggest to the administrators that a new structure be designed to accommodate this type of relationship.

Figure 6.2. RDBOM Attempting a Link Node to Additional Parent Operation

6.3. RESTRICTED PROPERTY CHARACTERISTIC VIOLATIONS

As identified in the previous section, there is a need in some situations for users to violate the restricted domain and range property characteristics that have been created on properties. Aside from the creation of the restriction violations table discussed in Section 5.3, the only other database work involved creating an authorization permission to

identify users capable of violating restrictions. By referring to the layout of RDBOM's authorization section (in Section 4.3), it is easy to see how this is accomplished by creating an authorization levels entry 'violate restrictions'. The subsections presented regarding various update pages only apply to users that have this authorization level.

The most considerable change to the existing validation routines involved the confirmation a user must provide before RDBOM commits the user's action to the database. For example, when linking two terms together, the term2terms entry is not created until the user's actions are reviewed, which takes place after the validation has been performed. Any violations of the domain and range restrictions that occur due to some action by a user cannot be stored immediately in the restriction violations table because the invalid data has not been created yet. This disrupted the flow of many pages in the Web-based RDBOM user interface that had followed a rigid, step by step process for carrying out actions, and resulted in the need to incorporate bookkeeping throughout every validation process. For every violation, the terms used as both the domain and range needs to be recorded as well as the restricted property and the violated restriction's term2terms entry. This provides enough information to uniquely identify a relationship stored in RDBOM's term2terms table as well as generate a detailed error message for the user.

The validation routine for the link node to node operation only needed to identify new domain and range violations that occurred due to a newly defined relationship. On the other hand, the Move/Cut update page is capable of not only introducing domain and range violations, but also fixing existing ones. Because of this, some validation routines have to not only check for new violations, but also check all the relationships that are validated to see if they are the cause of existing entries in the restriction violations table.

6.3.1. Node to Node Link Operations. All of the domain and range violations that occurred due to a new relationship need to be recorded. The implemented version of Algorithm 3.3 required some modification to keep lists of the specific properties that were violated and which specific property restriction caused the violation to occur. It also required information about the user to determine if they were able to violate restrictions or not, and thus, if their recording was necessary. After checking the relationship, any resulting errors get presented to the user. If the user chooses to go

ahead and make the relationship regardless of the errors, the term2term entry is created and there is now enough information to record each violation in the restriction violations table. The RDBOM interface would look identical to that shown in Figure 6.1, except there would be an additional button allowing the user to create the relationship regardless of the violations.

Nothing else needed to be implemented for the RDBOM operations affecting single relationships which involved deleting leaf nodes and deleting node to node links. Any violations caused by a leaf node were removed automatically from the restriction violations table due to the foreign key constraints specified on the *violated_term2term_id* column. Deleting a leaf deletes its relationships and thus, cascades to the violation entries. Similarly, a node to node link that gets deleted which also causes a violation which cascades through the *restricted_term2terms_id* of the restriction violations table.

6.3.2. Move/Cut and Link to Additional Parents. As mentioned earlier, the move term operation is capable of introducing new violations as well as fixing existing violations. The validation of a branch relies on validating each term of the branch with a new ancestry path. Individual relationship validation is capable of recording domain and range violations, so the term validation process can leverage this by retrieving a list of restricted properties a specific term is involved with and running each of these relationships through the already existing validation routine.

Moving a term to a new location in the tree could cause new violations to occur by either that term itself or its children. However, this action also is capable of fixing existing violations. The validate relationship routine only records the violated domain and range restrictions. In order to identify new violations that would result from a move and violations that are corrected, the branch validation routine is executed twice. The first time it is used to generate all of the current violations that exist in the branch. The second time it generates the violations that would occur by placing the branch under a new root, as described in Section 6.2.1. Comparing the two sets of violations produces the new violations which are not present in the first violation set, and the fixed violations which are those found in the first, but not second, violation set.

This process of determining the validity of a branch of the ontology before and after an operation is used in two other places besides the move/cut operation. Linking a

term to an additional parent can introduce new violations which can be detected in this manner, with the first violation set representing the violations of the term with its current ancestry paths and the second violation set representing only the new ancestry path of the term after the operation. Similarly, detaching a node from one of its parents can compare the first set of its current violations with the second set (which is generated after the node is detached from its parent). The second set contains the validations for all of the paths except for the deleted one so the comparison is used to identify any violations that are now valid (since the extra path has been deleted).

To demonstrate this, consider the modified Figure 2.1 that was presented at the beginning of this section. Assume those data had been imported into RDBOM, including the specification of the term *Chicken of the Sea* as a subclass of *Animal* instead of *Food.* This would be identified as a range violation of the property *eats* in the relationship *Horace eats Chicken of the Sea*. Figure 6.3 demonstrates how the move operation can be used to correct this violation.

6.4. ENTIRE ONTOLOGY VALIDATION

Performing validation of the entire ontology is necessary for reasons covered in Section 5, including checking the consistency of property use in imported ontologies, and defining new domain and range restrictions on properties. Thus, it could be the case that validation of an entire ontology is being performed which fully populates the restriction violations table, or that new property restriction is being added to an existing ontology. The implementation is similar to that for detaching a node from a parent (previously discussed in Section 6.3.3). First, a list of all the restriction violations entries are retrieved for the ontology. Next, the ontology is traversed, starting with the root term of the ontology and making a list of terms that have been visited. As each term is visited, it is validated in the same manner as terms are validated when checking branches. For each violation identified, the corresponding restriction violations entry is also identified and mark is as having been processed. If no entry is found, an appropriate entry in the table is created for it. After every term has been visited, the restriction violations entries that were not found are deleted and the user is informed that the violations no longer exist.

Figure 6.3. RDBOM Fixing a Violation via the Move Operation

Having the ability to validate an entire ontology allowed for the consistency checking of the AmphibAnat ontology. The AmphibAnat [2] ontology is being developed in RDBOM with 9,790 class terms and 44 properties. Excluding the basic relationships types (is a, part of) as well as properties used to describe unique RDBOM identifiers there are 15,145 relationships. The structure of the ontology is larger than the numbers portray due to many class terms having multiple parents which results in their entire branch being included in several places throughout the ontology. Several domain and range restrictions were put in place on the ontology's properties, as show in Figure 6.4. These restrictions mostly consist of properties involving terms located under the "Administrator's Node" which contains dbxrefs, images and synonyms. Other relationships involve citing pieces of literature. From these 12 property restrictions, 28 domain violations were discovered along with 58 range violations.

Reviewing the violations shows some improper use of several properties. For example, the *is discussed by* property is meant to relate a term to a piece of literature that contains a discussion of the term. The piece of literature should be the range of any *is*

Current Property Restrictions

Property	Restriction	Restricted to
is synonym of	has domain	synonym
image contains	has domain	image
discussed	has domain	literature
defined	has domain	literature
has dbxref	has range	dbxrefs
has rank	has range	rank
is defined by	has range	literature
is contained in image	has range	image
is discussed by	has range	literature
has related synonym	has range	synonym
has synonym	has range	synonym
has broad synonym	has range	synonym

Figure 6.4. AmphibAnat Restricted Properties

discussed by relationships. However, when a user was annotating one journal article term, *Dunlap 1960*, the journal article was improperly used as the domain of *is discussed by* relationships stating the journal article *is discussed by* several mussels. The intended property to use in the relationships was *discussed*. It identifies an inconsistent use of the *is discussed by* property. Figure 6.7 shows the report of violations detected.

6.5. INFORMATIONAL PAGES

A considerable amount of work went into managing violations of domain and range restrictions throughout the RDBOM system. But another aspect of this project concerned the display of relevant restriction violation information to the user. This affected two locations in the RDBOM user interface. Every class term has an information page that is displayed to the user which contains: relationships the term has with others, a term definition, and paths to the term in the ontology. Checking a relationship of a term to see if it is invalid when rendering this page to the user can be done easily since the violations are stored in a separate table. The invalid term in the

relationship (i.e., domain or range) is displayed in red and the restriction causing the entry is placed next to it. This provides an easy way for users to tell if a property has been applied correctly when browsing a term's information. As an example, Figure 6.5 shows RDBOM's information page for *Floyd* with the invalid relationship *Floyd eats Horace*. In the listing for *eats* relationships, *Horace* can be seen in red with the statement "range restricted to Food" following it.

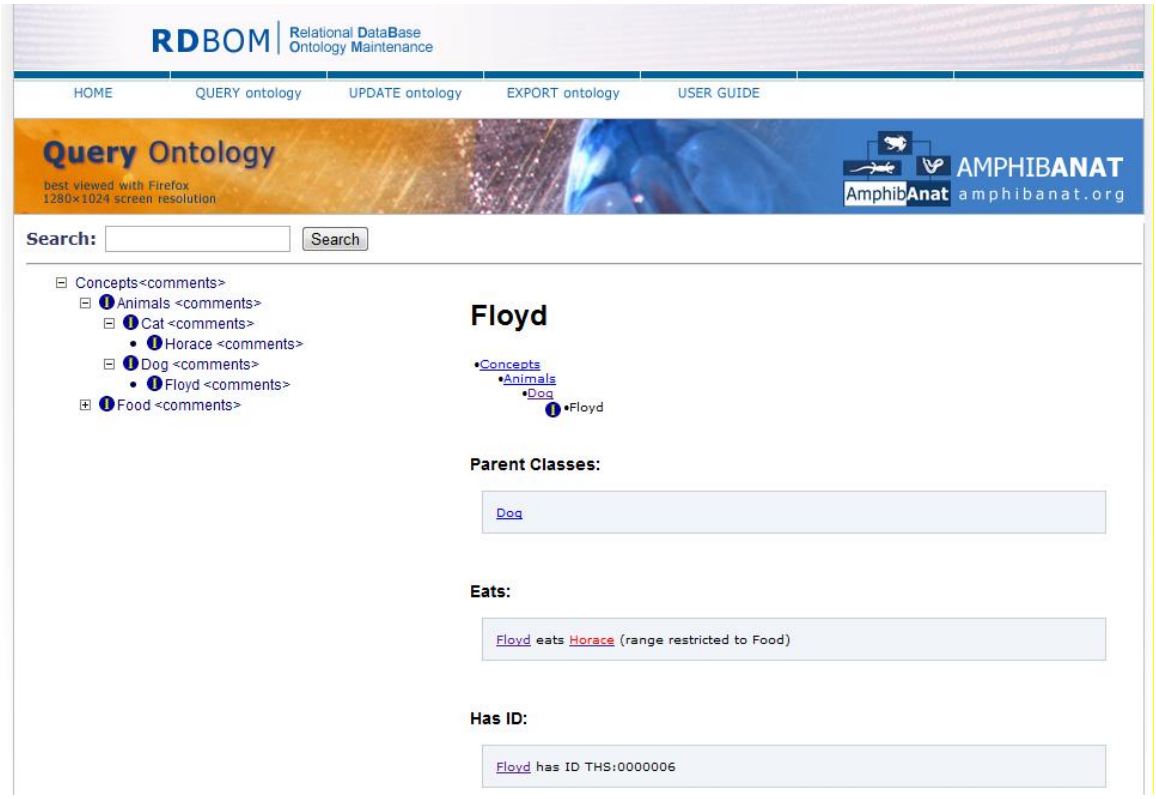

Figure 6.5. RDBOM Term Information Page for *Floyd*

Particularly in a large ontology such as AmphibAnat, it is not very practical for a user to check every single term in the ontology for invalid property use. Because of this, a page containing all of the relationships violating property restrictions was created. The page first shows every property restriction that exists in the ontology. It presents the user with a list of domain violations and a list of range violations. Links are provided to allow the user to view the terms' information pages. Additionally, if the user has the ability to

update the term to which the relationship applies, then a link is provided that allows him/her to delete node to node relationships for the term. An example of this functionality is shown in Figure 6.6, which is the violations overview page for Figure 2.1 (that has been used throughout this section). A listing of some of the violations found from the ontology validation performed on AmphibAnat in 6.4.1 can be seen in Figure 6.7.

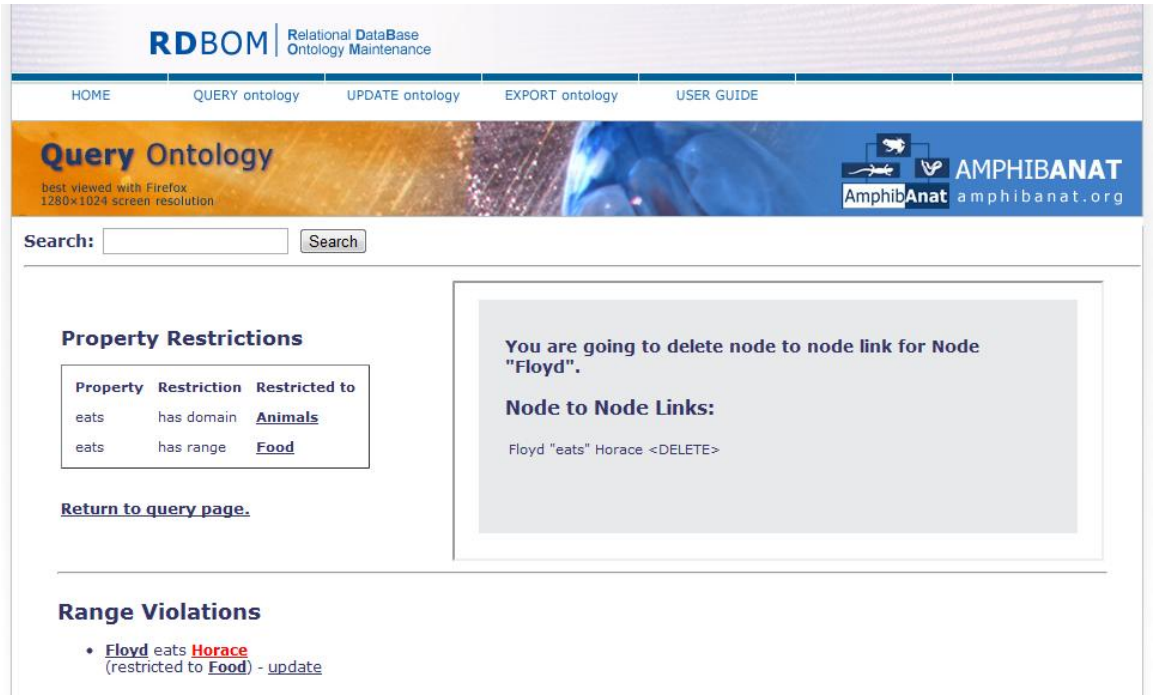

Figure 6.6. RDBOM Violations Overview Page

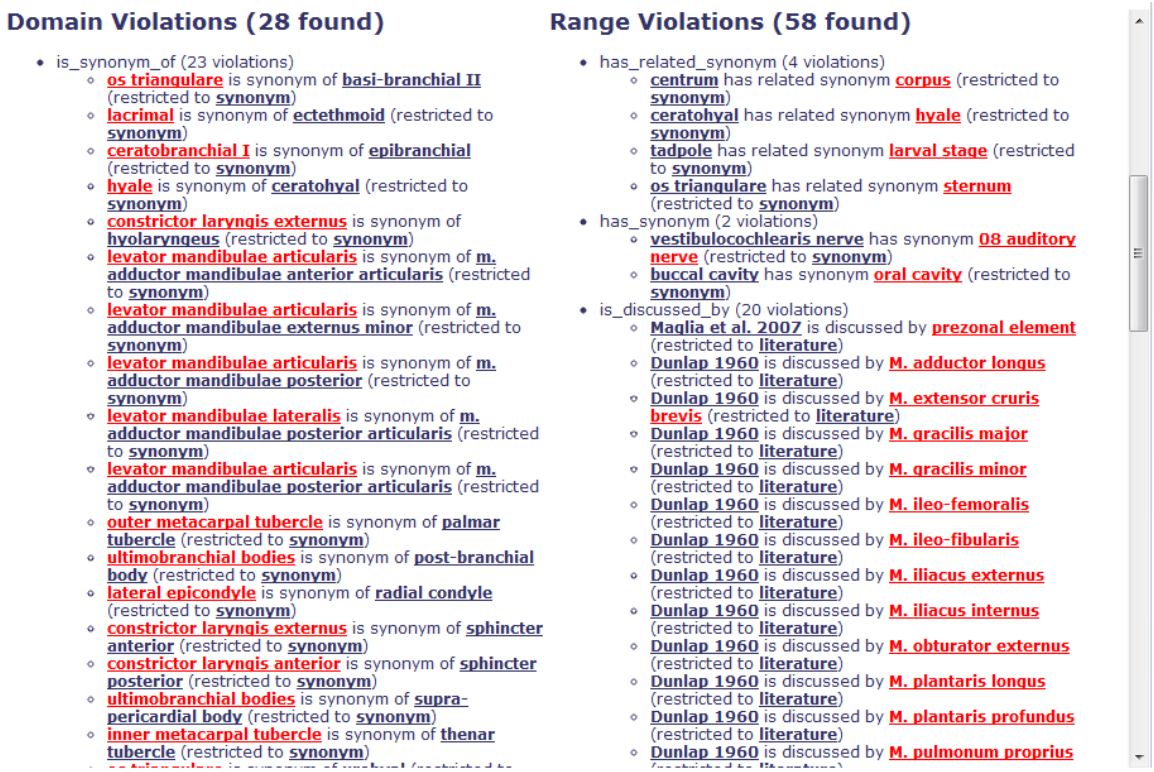

Figure 6.7. RDBOM Violations Overview Page for AmphibAnat

6.6. SUMMARY

This section addressed various implementation details associated with the domain and range restricted property characteristics in RDBOM. This discussion included functionality that allows administrators to define domain and range restrictions of properties, the detection and enforcement of the restrictions throughout the operations of RDBOM that allow modification of the ontology data, and the routines for performing the validation of the restrictions. Also discussed were the mechanisms that would allow users to violate restrictions under certain conditions, and a display of information about violated restrictions.

7. FUTURE WORK

The incorporation of property restrictions into the OAM, and the subsequent design and implementation of domain and range restrictions in RDBOM, suggested some interesting ideas for further work in this area.

As an example, Section 2 discussed how this implementation was focused on the need to check the consistency of property use throughout ontologies. But, property characteristics could also be used to *reason* over the information associated with terms. One extension to this work would be to turn the restriction violations table introduced in Section 5 into a table filled with facts inferred from reasoning. To demonstrate how this would work, recall the discussion in Section 2 regarding the difference in how RDBOM was to handle domain and range by restricting the properties to only those values, versus the OWL implementation which used domain and range to infer information about classes. OWL uses domain and range to infer that *Horace* is a subclass of *Food* based on the relationship of *Floyd eats Horace* and *eats has range Food*.

Now consider the violations stored in RDBOM's restriction violations as stored facts. Expressing this example in RDBOM, an entry would be created stating that *Floyd eats Horace* is invalid because of the range restriction on *eats* to the term *Food*. This means there is an entry in the restriction violations table stating *Floyd eats Horace* is invalid because of the restriction "*eats* has range *Food*." If range was considered to be a non-restricted property characteristic in RDBOM, this record could be used instead to state the relationship *Floyd eats Horace* is not invalid because of the restriction; rather it can be inferred that, since *Horace* is not explicitly stated to be a subclass of *Food* in the ontology, the relationship *Horace is_a Food* exists because of the use of *Horace* as the range term of the *eats* property.

In summary, future work on this project could focus more on using property characteristics to reason over the data, rather than check the consistency of the data. The ability to perform consistency checking is an excellent precursor to reasoning as it ensures valid facts would be inferred. In general, this is an area of research that currently is lacking for all ontologies, not just those based on the OAM, those implemented as a relational database, or those designed for multi-user curation.

8. SUMMARY

The research focus and problems relating to ontologies in information systems have changed over the years; namely, the focal point has shifted from the more theoretical ontology issues to problems associated with the actual development and use of ontologies in real-world, large-scale, collaborative applications. One step in this direction is the ability to specify and automatically enforce restrictions such as domain and range on relations and terms in the ontology. This would aid in the identification of inconsistencies in the ontology and ultimately help to maintain a higher level of data integrity.

OAM is a generic, abstract model that provides a framework for constructing a collaborative, Web-based ontology management system. In this paper, a modified Ontology Abstract Machine (OAM) model and its related algorithms for various domain and range maintenance tasks were presented. A modified OAM was proposed and implemented for the domain and range property restrictions. Detailed discussion of the RDBOM implementation was given for the validation of individual relationships, terms, and branches, as well as the techniques for managing information about relationships that violated the restrictions. It should be noted that domain and range were selected as a proof of concept; the proposed solution could be applied to other property restrictions as well.

It is hoped that future work on this project will explore the use of property characteristic restrictions to reason over the data, rather than just simply check the consistency of the data. In general, this is an area of research that currently is lacking for all ontologies, not just those based on the OAM, those implemented as a relational database, or those designed for multi-user curation. Realization of this goal ultimately will contribute significantly to the value of the Semantic Web.

BIBLIOGRAPHY

[1] http://ontology.buffalo.edu/. Buffalo Ontology Site, October 2010.

[2] http://www.amphibanat.org/. AmphibAnat, October 2010.

[3] J. Leopold, A. Coalter, and L. Lee, "A Generic, Functionally Comprehensive Approach to Maintaining an Ontology as a Relational Database," *ICOSE 2009: International Conference on Ontological and Semantic Engineering*, Rome, Italy, 2009.

[4] L. Lee**,** J. Leopold, J. Albath, and A. Coalter, "An Ontology Abstract Machine," *ICOSE 2009: International Conference on Ontological and Semantic Engineering*, Rome, Italy, 2009.

[5] http://www.oboedit.org/. OBO Edit, October 2010.

[6] http://protege.stanford.edu/. Protégé, October 2010.

[7] http://code.google.com/p/swoop/. SWOOP, October 2010.

[8] http://www.xspan.org/applications/cobra/. COBrA, October 2010.

[9] http://www.w3.org/TR/2004/REC-owl-guide-20040210/. OWL Web Ontology Language Guide, October 2010.

[10] http://www.geneontology.org/GO.format.obo-1_2.shtml. The OBO Flat File Format Specification, October 2010.

[11] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications," *The Semantic Web - ISWC 2004*, pp. 229-243, 2004.

[12] http://www.cs.man.ac.uk/~horrocks/FaCT/. FaCT, October 2010.

[13] http://clarkparsia.com/pellet. Pellet, October 2010.

[14] I. Yeh, P. D. Karp, N. F. Noy, and R. B. Altman, "Knowledge acquisition, consistency checking and concurrency control for Gene Ontology (GO)," *Bioinformatics*, Jan 22;19(2):241-8, 2003.

[15] I. Astrova, N. Korda, and A. Kalja, "Storing OWL Ontologies in SQL Relational Databases," *Proceedings of World Academy of Science, Engineering and Technology*, Vol 29, pp. 167-172, 2007.

[16] J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, and Y. Pan, "Minerva A Scalable OWL Ontology Storage and Inference System," *Asian Semantic Web Conference (ASWC)*, pp. 429-443, 2006.

[17] M. del Roldan-Garcia and J. F. Aldana-Montes, "DBOWL: Towards a Scalable and Persistent OWL Reasoner," *Internet and Web Applications and Services (ICIW '08)*, 2008.

[18] J. Lee, R. Goodwin, "Ontology Management for Large-Scale Enterprise Systems," *Electronic Commerce Research and Applications*, vol. 5, Iss. 1, Spring 2006, pp. 2–15.

VITA

Patrick Garrett Edgett was born and raised in Clinton, Missouri. After graduating from Clinton High School in May 2005 he started college the following August at Missouri University of Science and Technology in Rolla, Missouri. In May of 2009 he received the degree of Bachelor of Science in Computer Science. He continued his education at Missouri S&T and in December of 2010, completed the requirements for his Master of Science Degree in Computer Science.