International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:4, 2009

An Ontology Abstract Machine

Leong Lee, Jennifer Leopold, Julia Albath, and Alton Coalter

Abstract**—**As more people from non-technical backgrounds are becoming directly involved with large-scale ontology development, the focal point of ontology research has shifted from the more theoretical ontology issues to problems associated with the actual use of ontologies in real-world, large-scale collaborative applications. Recently the National Science Foundation funded a large collaborative ontology development project for which a new formal ontology model, the Ontology Abstract Machine (OAM), was developed to satisfy some unique functional and data representation requirements. This paper introduces the OAM model and the related algorithms that enable maintenance of an ontology that supports node-based user access. The successful software implementation of the OAM model and its subsequent acceptance by a large research community proves its validity and its real-world application value.

Keywords**—**Ontology, Abstract Machine, Ontology Editor, Web-based Ontology Management System.

I. 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; 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 use of ontologies in real-world, large-scale collaborative applications. For example, substantive research efforts have been spent on the development of formal languages that can be used to define ontologies and exchange information inherent to them. Open Knowledge Base Connectivity protocol [2] and Knowledge Interchange Format [3] are two of the most comprehensive languages that have been developed for these purposes.

Leong Lee, Jennifer Leopold, Julia Albath, Alton Coalter are affiliated with the Department of Computer Science, Missouri University of Science and Technology, Rolla, MO 65409 USA (e-mail: {llkr4, leopoldj, jgadkc, abcp7c}@mst.edu).

As the use of ontologies has become more pervasive in diverse domains, there also have been considerable efforts to facilitate ontology content development. A number of ontology editors have been developed, and their functionalities have been compared in various research papers (e.g., [4] compares Protege, Chimaera, OBO-Edit and OilEd, and [5] discusses Ontolingua, WebOnto, Protege, OntoSaurs, ODE and KADS22). Although most ontology editors are stand-alone file-based applications, there are also some ontology servers that take advantage of the World Wide Web to provide service to geographically distributed groups (e.g., the Ontolingua server [6] and the IBM ontology management system [7]).

In particular, ontology editor tools have been critical for supporting community-based efforts to create re-usable ontologies. For example, OBO Foundry is a project with the goal of creating a suite of orthogonal interoperable reference ontologies in the biomedical domain [8], [9]. As these community-based ontologies become larger and more complex, the issues of ontology merging and alignment have emerged as critical needs. Although a considerable amount of preliminary research has already been done in this area, there are still many open questions [10]. In fact, attempts have been made to classify different problems associated with combining ontologies; see [10], which also provides an overview of some of the unresolved issues.

As an example of a large ontology project that required the use of collaborative editing and a degree of modularity, in 2007 the National Science Foundation funded a project to build collaboratively an ontology of amphibian anatomy (*AmphibAnat* [11]). After determining that existing ontology editors/servers did not meet all of their needs, the biologists associated with the project requested the design and construction of a new Web-based ontology management system (*RDBOM* [12]).

To facilitate the implementation of that system, a novel theoretical ontology model, an Ontology Abstract Machine (OAM), was developed which meets the collaborative and modular needs of the *AmphibAnat* project. In this paper, the OAM model is introduced; the motivation for developing the new model is examined, and various algorithms and examples associated with the *AmphibAnat* project are presented.

II. Related Work

In [13], the authors discuss the need for the evolution of formal ontology languages to accommodate modular ontologies. The authors of [14] describe P-OWL, an extension of OWL that supports modular design, adaptation, and reuse of ontologies. P-OWL improves upon OWL by translating OWL entities and relationships into P-OWL

modules/packages. As with programming languages that support modularization via package and library constructs, P-OWL promotes decentralized development of ontologies.

Like databases, controlled user access to the various parts of an ontology also must be considered. In [15], the authors discuss the need for security in several applications owing to the private/sensitive information contained in the data. The authors point out that the need for controlled access is critical to the collaborative processes involved in the development and usage of ontologies; different collaborators play different roles, and different roles require different security access. The authors propose a semantics driven policy to enforce security.

In addition, many applications require selective sharing of information with users. The authors in [16] introduce the notion of a privacy-preserving reasoner that can be used to check whether a particular user can view certain information within an ontology (depending on the user's access rights) when querying information.

In developing the OAM model, both the nature of the modularity constructs that were developed for P-OWL, as well as the controlled access needs discussed in [15] and [16], were considered.

III. Motivation

A. The AmphibAnat Project

The Amphibian Anatomical Ontology (*AmphibAnat*) project is dedicated to semi-automatically constructing an amphibian anatomy ontology. In *AmphibAnat*, small ontologies are constructed manually by domain experts. Data-mining software is then used to mine electronic media for instances of concepts and properties to be added to the ontologies (using skeletons of the small ontologies as seed data) [17]. Draft semi-automatically constructed ontologies are made Web-accessible and are presented for community modification, enhancement, and curation.

Constructing an ontology of amphibian anatomy that is acceptable to the community requires multiple iterations and considerable effort. There are thousands of species of amphibians, each having a unique anatomical system. Numerous related publications, images, and specimens maintained by different researchers around the world must be considered and integrated into the process. Many of the researchers have developed their own small (formal or informal) ontologies related to their specialized research area that must be combined and resolved with the *AmphibAnat* ontology. Additionally, there are constant discussions and debates among amphibian researchers, with topics ranging from specific anatomical problems to the basic definition of an ontology. Traditionally, OBO-Edit and Protégé have been the two most common ontology editors used by these biologists. However, neither these tools nor any other available ontology maintenance tools fully met all of the centralized, community-curation needs for this project.

B. Relational Database Ontology Maintenance (RDBOM)

Based on the user requirements provided by the amphibian anatomy researchers, a collaborative ontology management system was developed. Relational Database Ontology Maintenance (*RDBOM*) is a Web-based software system that exploits the traditional features of a relational database management system in terms of concurrency control, security, and consistency checking in order to facilitate querying and updating an ontology. The details of the relational database design and the system architecture of RDBOM are discussed in [12]. The amphibian research community currently uses this system for their collaborative ontology construction effort, and the feedback gathered during the most recent yearly conference has been very positive.

C. Collaborative Ontology System

This new ontology system satisfies the following basic properties:

1) It is a Web-based, multi-user ontology system.
2) It provides user login and security features. This allows for user-level permissions and control of different ontology classes and instances. In a collaborative editing environment, only users with permission granted can edit the parts of the ontology for which they are responsible.
3) The basic ontology definition is customizable to satisfy the special requirements of the amphibian research community. This allows future evolutions of the amphibian anatomy ontology and provides the foundation to extend the system to other biology research communities.
4) It allows users to import any parts of the ontology from (and export to) other common ontology data formats such as OBO (used by OBO-Edit) [18] and OWL (used by Protégé) [19], [20].
5) It allows users to merge two ontologies, and to calculate the differences between two ontologies.
6) It allows users to swap in (and out) a subset of an ontology as a module. It should be noted that an ontology module normally has links to other parts of the ontology, and that those links should be preserved as much as possible during these swapping operations.

IV. Ontology Abstract Machine (OAM)

A. The Need for a Mathematical Model

In the initial design of the *RDBOM* system, one of the main challenges was the development of an appropriate formal model. An implementation-independent mathematical model was needed to represent the various ontology constructs (i.e., terms, relations, attributes, etc.), and to facilitate algorithms that would support multi-user access, node-based security, a customizable ontology definition, importing/exporting to other data formats, module swapping, and merging/calculating the difference of ontology modules. Hence, a mathematical model called an Ontology Abstract Machine (OAM) was designed, and it became the foundation for implementing *RDBOM.*

B. Special User Requirements

The main objective of building the *RDBOM* system (and hence designing the underlying mathematical model) was to serve the needs of a community of hundreds of amphibian biologists. The use of a directed acyclic graph $G = (V, E)$ to represent the ontology was considered, where V is a set of vertices (nodes / terms) and E is a set of edges (relationships), similar to the approach used in OBO-EDIT [18]. Although the definition of a directed graph can be modified to implement requirements like multi-user support and node-based security, a directed graph does not readily accommodate some of the other requirements for the *AmphibAnat* project.

Representation of the *AmphibAnat* data required two types of relationships. The first class of relationships forms the base structure or backbone of the ontology (e.g., *is_a* and *part_of* relations), and, for this application, should not allow cycles. But there is also a second class of extended relationships in this ontology wherein cycles should be allowed. This requirement affects the semantics of the ontology; the base relationships represent mainly the domain knowledge, whereas the extended relationships represent a mixture of domain knowledge (in terms of properties), data representation, and data integration. Effectively, this could be implemented using two different instantiations of a directed graph for the ontology. But (as will be seen later) the two directed graphs would need to be connected for some functions, and separated for other functions. These special requirements were the main motivations for developing a new model.

C. OAM Definition

The OAM model is defined in a format similar to that used for a finite state automaton. This format makes it very straightforward to accommodate the two classes of relationships and to implement functionality such as importing/exporting modules, swapping modules, and merging/calculating the difference of ontologies (the algorithms for which are presented in Section V).

Definition 1. Ontology Abstract Machine (OAM)

OAM is a 5-tuple representation of an ontology.

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

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, contains_image, is_from_image, ...}*

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.

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

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. It can be used as links (among classes and instances) to generate the main graph view of an ontology. This main graph view cannot have any cycles. Two of the most common base relationship types are *is_a* and *part_of*.

$\sum_E$ 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. This is in response to a special request from the *AmphibAnat* project coordinators to allow users to create a link between classes and instances (without affecting the main structure of the ontology).

Here an example of using the OAM to represent an ontology is presented.

Example 1: OAM representation of an ontology

OAM instance $M_1 = (Q_1, \sum_1, \delta_1, Q_{01}, F_1)$:

$Q_1 = \{Concepts, e, b, g, i, f\}$;

all nodes are classes in this case

$\sum_1 = \sum_{B1} \cup \sum_{E1} = \{is\_a, is\_contained\_in\_image, image\_contains\}$

$\sum_{B1} = \{is\_a\}$; $\sum_{E1} = \{is\_contained\_in\_image, image\_contains\}$

$Q_{01} = \{b, g, f\}$

$F_1 = \{Concepts\}$

$\delta_1 = \{(e, is\_a, Concepts), (b, is\_a, e), (g, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, g), (g, is\_contained\_in\_image, f)\}$

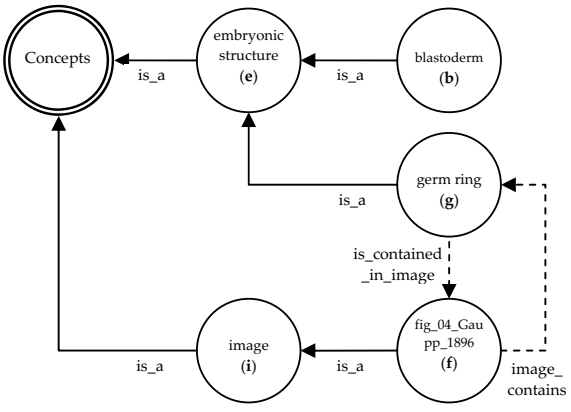

Fig. 1 Ontology M_1

Please refer to Fig. 1 for the graphical OAM representation

of this ontology. Note that this is a 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 (for a small ontology).

V. The Algorithms

To provide the functionality required by the *AmphibAnat* project to manipulate the ontology, the following algorithms were developed.

A. Multi-User Access

Multi-user access and node-based security access for the OAM model are addressed in Algorithms 1.1, 1.2 and 1.3. User access is controlled at the level of class nodes and instance nodes. In other words, user ids are only attached to elements of sets Q_c and Q_i. As a result, in the algorithms given below, parent/child navigation is only done within the main graph view (classes Q_c and instances Q_i linked by base relationship types, $\sum_B$).

Algorithm 1.1. Grant access to a node for a user.
Input: An OAM instance, a user id u_i, and a node q.
Output: An updated OAM instance.
Algorithm:

```
if q is an element of (Qc ∪ Qi) then proceed, else exit
result = check_up (q, ui)   // check_up is declared below
// determine if user permission already exists
if result is "found" then
   exit
else
begin
   attach ui to q as user access record
   check_down (q, ui)   // check_down is declared below
   //check if there is any lower level permission record
end

check_up (test_node, ui)
begin
   if ui is attached to test_node then
      return "found"
   else if test_node is a root then
      return "not_found"
   else
      for each parent of test_node
      begin
         result = check_up (parent_node, ui)
         if result is "found" then
            return "found"
      end for loop
      return "not_found"
   end if/else
end check_up

check_down (test_node, ui)
begin
   if there are children of test_node then
   begin
      for each child_node of test_node
      begin
         if ui is attached to child_node then
            remove access to child_node for ui
         else
            check_down (child_node, ui)
         end if/else
      end for loop
   end if
end check_down
```

Example 2: Grant permission to a node for a user
Please refer to Fig. 2 for the graphical OAM representation of this example.

Request A:
If permission is to be granted to user id u_1 for node "*b*", on ontology M_1, the following steps take place:
1. check_up (*b*), result is "not_found"
2. attach u_1 to *b* as user access
3. check_down (*b*), nothing is done

Request B:
If permission to be granted to user id u_1 for node "*Concepts*", on ontology M_1, the following steps take place:
1. check_up (*Concepts*), result is "not_found"
2. attach u_1 to *b* as user access
3. check_down (*Concepts*), remove access to *b* for u_1

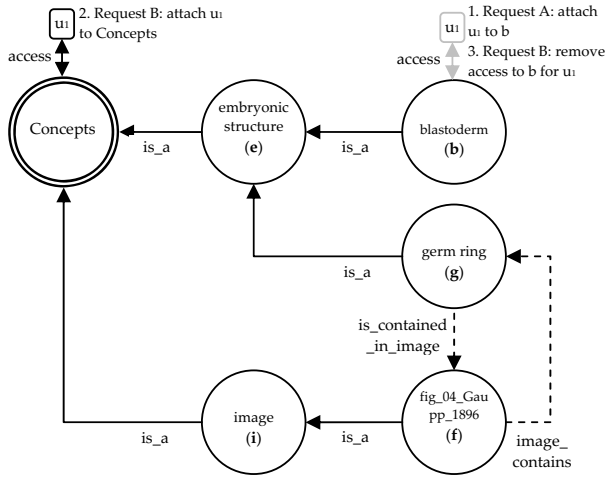

Fig. 2 Grant user permission to a node for M_1

Algorithm 1.2. Check if a user has access to a node.
Input: An OAM instance, a user id u_i, and a node q.
Output: If u_i have access to q, Yes/No.

Algorithm:
result = check_up (q, u_i) // it is declared in algorithm 1.1
if result is "found" then u_i has access to node q
else u_i does not have access to node q

Algorithm 1.3. Delete user access from a node.
Input: An OAM instance, a user id u_i, and a node q.
Output: An updated OAM instance.
Algorithm:
result = check_up (q, u_i) //it is declared in algorithm 1.1
if result is "found" then
  exit //operation not allowed
else
begin
  if u_i is attached to q as user access record then
    remove access to q for u_i
  else
    check_down (q, u_i) //it is declared in algorithm 1.1
end

To date, the association of access rights with ontologies is a research area that has not been thoroughly investigated. In [21], the authors propose a method for representing general role-based access control in OWL. The OAM-based approach, specifically provides access rights for users, allowing a much more detailed level of security.

B. Customize Ontology

During the *AmphibAnat* workshop, the participants requested the ability to customize the basic ontology hierarchy; that is, the users (with appropriate authority) wanted to be able to add/delete/modify the relationship types (both base and extended) during the ontology building process. This functionality is formalized for the OAM in Algorithms 2.1.1-2.1.3 and 2.2.1-2.2.3.

Algorithm 2.1.1. Add a base relationship type.
Input: An OAM instance, a new base relationship type b.
Output: An updated OAM instance.
Algorithm:
If b is not an element of $\sum_B$ then
  add b to set $\sum_B$
else
  return "b exists, nothing is done"
end if/else

Algorithm 2.1.2. Delete a base relationship type.
Input: An OAM instance, an existing base relationship type b.
Output: An updated OAM instance.
Algorithm:
if b is not an element of $\sum_B$ then
  return "b does not exist, nothing is done"
else
begin
  if b is used in any current edge then
    return "b is used in edge(s), nothing is done"
  else
    remove b from set $\sum_B$
end if/else

Algorithm 2.1.3. Modify a base relationship type.
Input: An OAM instance, an existing base relationship type b, a new base relationship type b'.
Output: An updated OAM instance.
Algorithm:
if b is not an element of $\sum_B$ then
  return "b does not exist, nothing is done"
else
begin
  change b to b' in set $\sum_B$
  update all existing edges using b (change b to b')
end if/else

Algorithm 2.2.1. Add an extended relationship type.
Input: An OAM instance, a new extended relationship type e.
Output: An updated OAM instance.
Algorithm:
If e is not an element of $\sum_e$ then
  add e to set $\sum_e$
else
  return "e exists, nothing is done"
end if/else

Algorithm 2.2.2. Delete an extended relationship type.
Input: An OAM instance, an existing extended relationship type e.
Output: An updated OAM instance.
Algorithm:
If e is not an element of $\sum_e$ then
  return "e does not exist, nothing is done"
else
begin
  if e is used in any current edge then
    return "e is used in edge(s), nothing is done"
  else
    remove e from set $\sum_e$
end if/else

Algorithm 2.2.3. Modify an extended relationship type.
Input: An OAM instance, an existing extended relationship type e, a new extended relationship type e'.
Output: An updated OAM instance.
Algorithm:
If e is not an element of $\sum_e$ then
  return "e does not exist, nothing is done"
else
begin
  change e to e' in set $\sum_e$
  update all existing edges using e (change e to e')
end if/else

C. Basic Ontology Construction

The basic functions that most users need to perform during ontology construction are to add/delete/modify nodes and to add/delete/modify edges. The algorithms for the construction of nodes and edges are intuitive. For example, to add a child node to an existing ontology just requires the creation of the node, and the creation of a new directed edge (child node, relationship type, existing parent node). The extra consideration here is checking to make sure that the user performing this operation has appropriate user access to the existing node(s). To check if a user id u_i has such access to the involved node(s), Algorithm 1.2 can be used (i.e., to check to see if a user has access to a particular node, recursively as necessary).

D. Import/Export an Ontology

An ontology is an explicit specification of a conceptualization of a domain [22]. Changes in the specification can occur when an ontology is translated from one language to another. For the *AmphibAnat* project, some parts of the ontology were initially developed in OWL by several distinct groups of biologists while other parts were developed in OBO. In order to facilitate the community-based, collaborative ontology construction effort, these OWL and OBO ontologies were converted to the OAM model. But OWL and OBO not only differ in their syntax, but also in their semantics and expressivity. As a result, the main focus of our translation task was to try to preserve the semantics of both representations.

As an example, synonyms are handled very differently in OWL and OBO. The OAM model adopts a representation similar to that used by OWL, but still preserves the more expressive synonym categorization utilized by OBO. In general, export from the OAM model to either the OWL or OBO languages tries to preserve the original semantics of the ontology as much as possible. In the algorithms presented in this section the import/export to OBO function is used as an example of the more general process of importing/exporting an ontology to another format such as OWL.

Algorithm 3.1. Import an ontology.
Input: An OBO file.
Output: An OAM instance.
Algorithm:

1. Place values corresponding to the default set of OBO relationships into $\sum_B$ and $\sum_E$.
2. Get all terms tagged as OBO names. Place those from "[Term]" stanzas in Q_c, those from "[Instance]" stanzas in Q_i, and those from "[Typedef]" stanzas in either $\sum_B$ or $\sum_E$ as appropriate.
3. Get all terms tagged as OBO ids and place them in Q_c.
4. Create edges between the terms created in step 2 above to the corresponding ID terms created in step 3 above using the appropriate value from $\sum_E$.
5. For each term get all tags corresponding to values from $\sum_B$ (base relationship types) and create edges between the term and its parent(s).
6. Add the term "Concepts" to Q_c.
7. Create an edge between each term in Q_c (other than "Concepts") that had no edge created in step 5 above (that is, those terms with no parent) and the term "Concepts" using the "*is_a*" relationship from $\sum_B$.
8. For all other tags previously unused, create an edge between the term and its associated value or term using the corresponding value from $\sum_B$ or $\sum_E$, placing that value into Q_v. All terms at this time will already exist in Q_c or Q_i as a result of previous steps.

Algorithm 3.2. Export an ontology.
Input: An OAM instance.
Output: An OBO file.
Algorithm:

1. Get all root nodes from F, and create OBO tags.
2. Get associated values (elements of Q_v) for all root nodes, using edges (value Q_v, extended relationship type $\sum_E$, root node F); create OBO tags for values and edges found.
3. Starting from root nodes, recursively find all nodes in the main graph view (classes Q_c and instances Q_i linked by base relationship types, $\sum_B$), and create OBO tags for nodes found. Elements of Q_c are related to "[Term]" stanzas in OBO, and elements of Q_i are related to "[Instance]" stanzas.
4. Use different OBO tags for OBO specific semantics exceptions (e.g. synonyms representation in OBO).
5. For each node found in step 3, get the associated edges linking classes and instances, (classes Q_c and instances Q_i linked by base relationship types, $\sum_B$; and classes Q_c and instances Q_i linked by extended relationship types, $\sum_E$), and create OBO tags for edges found.
6. For each node found in step 4, get the associated values, (elements of Q_v) using edges (value Q_v, extended relationship type $\sum_E$, node Q_c or Q_i), and create OBO tags for values and edges found.
7. Get all values from $\sum_B$ and from $\sum_E$, and create OBO tags. These tags will be related to OBO "[Typedef]" stanzas.

E. Merge Ontologies

Prior to the use of *RDBOM*, there were situations where two or more biologists working on the *AmphibAnat* project started building an ontology together, then traveled to different geographical areas, continuing to add extensions to their own copy of their original ontology. This was one example of the need to be able to compare and/or merge two ontologies for later consolidation. The algorithms presented below address this task for the OAM model.

Algorithm 4.1. Merge two ontologies.
Input: OAM instances M_1 and M_2.
$M_1 = (Q_1, \sum_1, \delta_1, Q_{01}, F_1)$; $\sum_1 = \sum_{B1} \cup \sum_{E1}$
$M_2 = (Q_2, \sum_2, \delta_2, Q_{02}, F_2)$; $\sum_2 = \sum_{B2} \cup \sum_{E2}$
Output: OAM instances M_3, $M_3 = M_1 \cup M_2$.

Algorithm:
$M_3 = (Q_3, \sum_3, \delta_3, Q_{03}, F_3)$
$Q_3 = Q_1 \cup Q_2$
$\sum_3 = \sum_{B3} \cup \sum_{E3}$
$\sum_{B3} = \sum_{B1} \cup \sum_{B2}$; $\sum_{E3} = \sum_{E1} \cup \sum_{E2}$
$\delta_3 = \delta_1 \cup \delta_2$
Q_{03} is to be determined from scanning δ_3.
$F_3 = F_1 \cup F_2$

Example 3: $M_3 = M_1 \cup M_2$
Please refer to Fig. 1 for ontology M_1, Fig. 3 for ontology M_2 and Fig. 4 for ontology M_3.
Ontology M_2:
$Q_2 = \{Concepts, a, e, b, i, f, d\}$;
$Q_{c2} = \{Concepts, a, e, b, i, f\}$; $Q_{i2} = \{\}$; $Q_{v2} = \{d\}$
"d" is a value node
$\sum_2 = \sum_{B2} \cup \sum_{E2} = \{is\_a, is\_contained\_in\_image, image\_contains, is\_definition\}$
$\sum_{B2} = \{is\_a\}$; $\sum_{E2} = \{is\_contained\_in\_image, image\_contains, is\_definition\}$
$Q_{02} = \{a, b, f\}$
$F_2 = \{Concepts\}$
$\delta_2 = \{(a, is\_a, Concepts), (e, is\_a, Concepts), (b, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, b), (b, is\_contained\_in\_image, f), (d, is\_definition, a)\}$

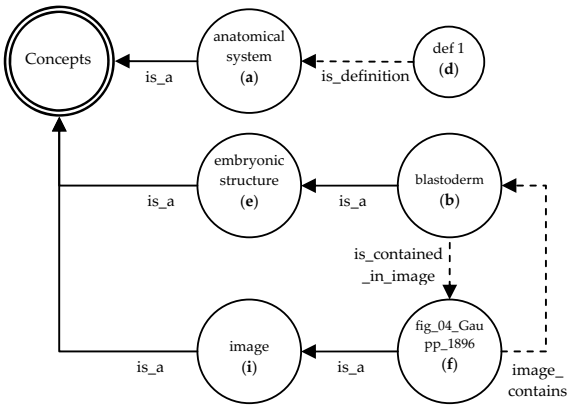

Fig. 3 Ontology M_2

Ontology M_3: $M_3 = (Q_3, \sum_3, \delta_3, Q_{03}, F_3)$
$Q_3 = Q_1 \cup Q_2$
$= \{Concepts, e, b, g, i, f\} \cup \{Concepts, a, e, b, i, f, d\}$
$= \{Concepts, a, e, b, g, i, f, d\}$
$Q_{c3} = \{Concepts, a, e, b, g, i, f\}$; $Q_{i3} = \{\}$; $Q_{v3} = \{d\}$
$\sum_3 = \sum_{B3} \cup \sum_{E3} = \{is\_a, is\_contained\_in\_image, image\_contains, is\_definition\}$
$\sum_{B3} = \sum_{B1} \cup \sum_{B2} = \{is\_a\}$
$\sum_{E3} = \sum_{E1} \cup \sum_{E2} = \{is\_contained\_in\_image, image\_contains, is\_definition\}$
$\delta_3 = \delta_1 \cup \delta_2 = \{(a, is\_a, Concepts), (e, is\_a, Concepts), (b, is\_a, e), (g, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, g), (f, image\_contains, b), (g, is\_contained\_in\_image, f), (b, is\_contained\_in\_image, f), (d, is\_definition, a)\}$
$Q_{03} = \{a, b, g, f\}$.
$F_3 = F_1 \cup F_2 = \{Concepts\}$

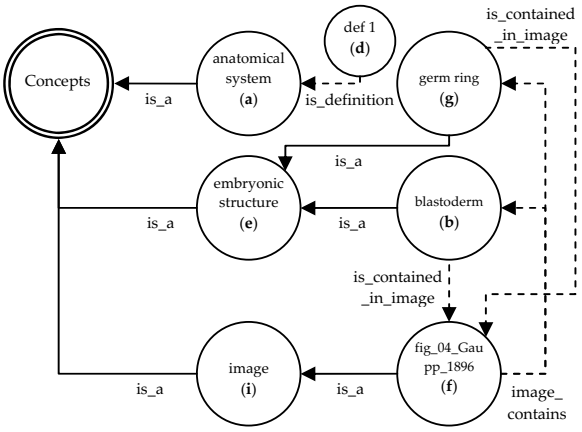

Fig. 4 Ontology M_3

Algorithm 4.2. Find the difference between two ontologies.
Input: OAM instances M_1 and M_2.
$M_1 = (Q_1, \sum_1, \delta_1, Q_{01}, F_1)$; $\sum_1 = \sum_{B1} \cup \sum_{E1}$
$M_2 = (Q_2, \sum_2, \delta_2, Q_{02}, F_2)$; $\sum_2 = \sum_{B2} \cup \sum_{E2}$
Output: OAM instances M_4, $M_4 = M_1 - M_2$.
$M_4 = (Q_4, \sum_4, \delta_4, Q_{04}, F_4)$
$\delta_4 = \delta_1 - \delta_2$
Q_4 is to be determined from analyzing δ_4;
$Q_{c4} = Q_4 \cap (Q_{c1} \cup Q_{c2})$;
$Q_{i4} = Q_4 \cap (Q_{i1} \cup Q_{i2})$;
$Q_{v4} = Q_4 \cap (Q_{v1} \cup Q_{v2})$.
$\sum_4$ is to be determined from analyzing δ_4;
$\sum_{B4} = \sum_4 \cap (\sum_{B1} \cup \sum_{B2})$;
$\sum_{E4} = \sum_4 \cap (\sum_{E1} \cup \sum_{E2})$.
Q_{04} is to be determined from analyzing δ_4.
F_4 is to be determined from analyzing δ_4.

Example 4: $M_4 = M_1 - M_2$
Please refer to Fig. 1 for M_1, Fig. 3 for M_2, and Fig. 5 for M_4.
$\delta_4 = \delta_1 - \delta_2$
$= \{(g, is\_a, e), (f, image\_contains, g), (g, is\_contained\_in\_image, f)\}$
$Q_4 = \{g, e, f\}$
$\sum_4 = \{is\_a, image\_contains, is\_contained\_in\_image\}$
$\sum_{B4} = \sum_4 \cap (\sum_{B1} \cup \sum_{B2}) = \{is\_a\}$
$\sum_{E4} = \sum_4 \cap (\sum_{E1} \cup \sum_{E2}) = \{image\_contains, is\_contained\_in\_image\}$
$Q_{04} = \{g\}$
$F_4 = \{e, f\}$

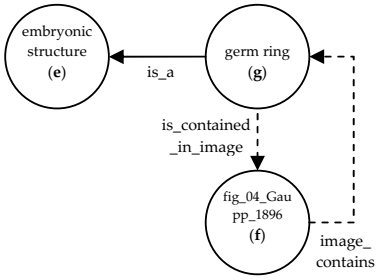

Fig. 5 Ontology M_4

The work in [23] describes several approaches to ontology merging and the calculation of differences. Regarding ontology merging, PROMPT and OntoMerge are discussed. PROMPT also includes an algorithm for difference calculation. Additionally, the authors of [23] introduce a practical approach for ontology merging and calculation of difference. *RDBOM* also provides for ontology merging and difference calculation. However, the *RDBOM* implementation is based on the OAM, while the work in [23] is based on description logic (DL).

F. Subset of an Ontology

Some of the more challenging requests that were received for the *AmphibAnat* project involved the ability to manipulate a subset of an ontology. For example, a user may want to import or export a subset of an ontology to a separate file, or to "swap" one part of an ontology for part of another ontology. The latter case is particularly tricky because presumably the user wants to keep all possible existing edges which link the subset ontology to/from the rest of the larger ontology. This required the definition of a subset of an OAM.

Definition 2. Subset of an OAM

Given an OAM, $M = (Q, \sum, \delta, Q_0, F)$; $Q = Q_c \cup Q_i \cup Q_v$; $\sum = \sum_B \cup \sum_E$, a subset of M is defined as a pair (M, a), where $a \in (Q_c \cup Q_i)$. "a" is provided by user as part of the subset request. Thus, $(M, a) = (Q', \sum', \delta', Q_0', F')$

- $Q' = Q_c' \cup Q_i' \cup Q_v'$ $(Q_c' \subset Q_c,\ Q_i' \subset Q_i,\ Q_v' \subset Q_v)$
- $(Q_c' \cup Q_i') = \{a\} \cup$ set of descendant nodes of "a" in M, elements of set Q_v' = elements of Q_v connected to elements of $(Q_c' \cup Q_i')$ by edges in M.
- δ' = set of relationships (n_1, r, n_2), where $n_1 \in Q'$, $n_2 \in Q'$, $r \in \sum$, and $(n_1, r, n_2) \in \delta$ in M.
- $\sum' = \sum_B' \cup \sum_E'$ $(\sum_B' \subset \sum_B,\ \sum_E' \subset \sum_E)$, elements in $\sum_B'$ and $\sum_E'$ are derived from δ'.
- Q_0' = set of source nodes which are derived from δ'.
- F' = set of root nodes which are derived from δ'.

Note:

- To find descendant nodes of "a", start from node "a", and recursively find all nodes in the main graph view (classes Q_c and instances Q_i linked by base relationship types, $\sum_B$).
- To form $\sum_B'$ and $\sum_E'$, for each relationship (n_1, r, n_2) of δ', if r is an element of $\sum_B$, add r to $\sum_B'$, else add r to $\sum_E'$.

Example 5: Subset of an ontology.

Given OAM instance M_5 (Fig. 6), find subset (M_5, g) where

$M_5 = (Q_5, \sum_5, \delta_5, Q_{05}, F_5)$

$Q_5 = \{Concepts, e, b, h, ep, g, i, f\}$;
all nodes are classes in this case

$\sum_5 = \sum_{B5} \cup \sum_{E5} = \{is\_a, part\_of, is\_contained\_in\_image, image\_contains\}$
$\sum_{B5} = \{is\_a, part\_of\}$; $\sum_{E5} = \{is\_contained\_in\_image, image\_contains\}$

$Q_{05} = \{b, h, ep, f\}$

$F_5 = \{Concepts\}$

$\delta_5 = \{(e, is\_a, Concepts), (b, is\_a, e), (h, is\_a, e), (h, part\_of, g), (ep, is\_a, e), (ep, part\_of, g), (g, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, g), (g, is\_contained\_in\_image, f)\}$

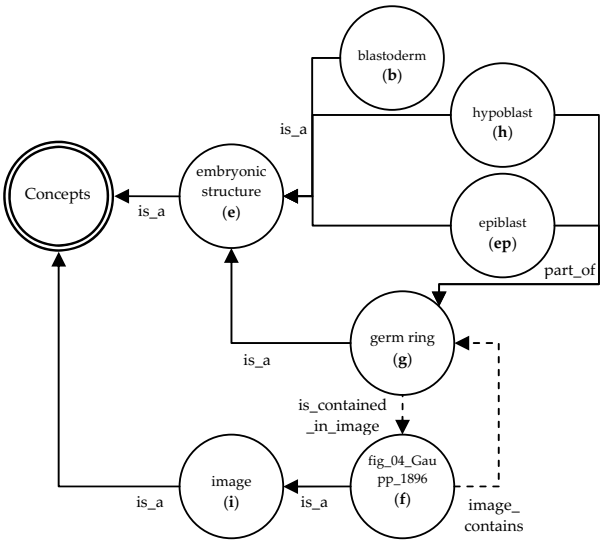

Fig. 6 Ontology M_5

subset $(M_5, g) = (Q_5', \sum_5', \delta_5', Q_{05}', F_5')$

$Q_5' = Q_{c5}' \cup Q_{i5}' \cup Q_{v5}' = \{g, h, ep\}$
$(Q_{c5}' \cup Q_{i5}') = \{g\} \cup$ *set of descendant nodes of "g"*
$(Q_{c5}' \cup Q_{i5}') = \{g, h, ep\}$; Q_{v5}' *is empty*

$\delta_5' = \{(h, part\_of, g), (ep, part\_of, g)\}$

$\sum_5' = \sum_{B5}' \cup \sum_{E5}' = \{part\_of\}$
$\sum_{B5}' = \{part\_of\}$; $\sum_{E5}'$ *is empty*

$Q_{05}' = \{h, ep\}$

$F_5' = \{g\}$

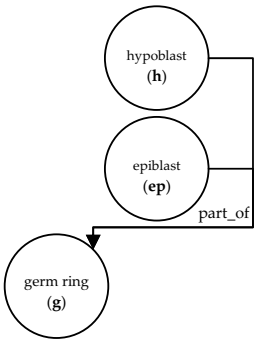

Fig. 7 subset (M_5, g)

It is important to note that the terms *module* and *subset* are semantically similar. In [24], the authors proposed a definition of a module for an ontology that is intended to be reused. Based on this definition, the authors showed that determining whether a portion of an ontology is a module for a given vocabulary is undecidable even for rather restricted sub-languages of OWL DL. However, the authors provided two approximation algorithms for finding small modules of ontologies.

The module definition in [24] is somewhat different from the subset definition given in this paper; a module is designed for knowledge reuse based on description logic, whereas the OAM subset definition is intended to facilitate particular ontology operations for a multi-user ontology management system based on the OAM model. However, the minimal module problem in [24] does provide interesting future research opportunities for the OAM model, and hence future possible extension for the *RDBOM* system.

Algorithms 5.1 and 5.2 use the definition of a subset OAM to perform export and import functions on an ontology.

Algorithm 5.1. Export a subset of an ontology.

Input: OAM instance M and a node "a", $a \in (Q_c \cup Q_i)$.

Output: OAM instance subset (M, a), as an output file (e.g. OBO file).

subset (M, a) = (Q', $\sum'$, δ', Q_0', F')

1. Find descendant nodes of "a". Start from node "a", recursively find all nodes in the main graph view (classes Q_c and instances Q_i linked by base relationship types, $\sum_B$). Put these nodes and node "a" in ($Q_c' \cup Q_i'$).
2. Find elements of Q_v connected to elements of ($Q_c' \cup Q_i'$) by edges in M. Put these elements in Q_v'.
3. $Q' = Q_c' \cup Q_i' \cup Q_v'$
4. for each element / relationship (n_1, r, n_2) of δ (from M)
   if ((n_1 is a member of Q') and (n_2 is a member of Q'))
   then put (n_1, r, n_2) in δ'
   end for
5. for each element / relationship (n_1, r, n_2) of δ'
   if r is an element of $\sum_B$ then add r to $\sum_B'$
   else add r to $\sum_E'$
   end for
6. $\sum' = \sum_B' \cup \sum_E'$
7. for each element / relationship (n_1, r, n_2) of δ'
   if n_1 has no incoming edge then add n_1 to Q_0'
8. for each element / relationship (n_1, r, n_2) of δ'
   if n_2 has no outgoing edge then add n_2 to F'
9. subset (M, a) is now a complete OAM instance
10. Use Algorithm 3.2. (export ontology) to complete the export process using subset (M, a) as input to Algorithm 3.2.

Importing an ontology as a subset is an easier task than exporting a subset. Basically an imported ontology is treated as a subset and is attached to a specified node.

Algorithm 5.2. Import an ontology as a subset.

Input: Existing OAM instance M and a node "a", $a \in (Q_c \cup Q_i)$.

An input file (e.g. OBO file) to be imported and attached to node "a" of M.

Output: Updated OAM instance M.

1. Use Algorithm 3.1. to complete the import process, use the input file as input to Algorithm 3.1. The output of Algorithm 3.1 is called OAM instance M'.
2. Since M' only has one root node "*Concepts*", attach M' to M by replacing node "*Concepts*" of M' by node "a" of M. Thus M' is attached to node "a" of M, and become part of M.

Swapping a subset of an ontology is different from combining the actions of exporting a subset of an ontology, deleting the subset (and related edges), and importing a new ontology as a new subset. If the old subset is simply deleted, all the edges linking the subset and the rest of the ontology will be lost. The new imported subset will not have these edges linked to the rest of the ontology. Hence, a new algorithm, Algorithm 5.3, was developed to swap a subset of an ontology.

Algorithm 5.3. Swap a subset of an ontology.

Input: OAM instance M and a node "a", $a \in (Q_c \cup Q_i)$; $M = (Q, \sum, \delta, Q_0, F)$. An input file (e.g. OBO file) is to be imported and replace subset (M, a).

Output: Updated OAM instance M, called M'. $M' = (Q', \sum', \delta', Q_0', F')$.

1. Use Algorithm 3.1. to perform the import process, and use the resulting file as input to Algorithm 3.1. The output of Algorithm 3.1 is called OAM instance N. N is the OAM instance to be used to replace subset (M, a).
2. Initialize M' to be an empty OAM, $M' = (Q', \sum', \delta', Q_0', F')$.
3. Add nodes to M', $Q' = Q$ - nodes of subset (M, a) $\cup \{a\}$
4. Add edges to M'.
   for all edges (n_1, r, n_2) in δ of M
   if both n_1, n_2 are in Q' of M' then add (n_1, r, n_2) to δ'
   end for

5. Use Algorithm 5.2. to attach N to M'.
6. Add edges to M'
   for all edges (n_1, r, n_2) in δ of M (that has not been added to δ' in step 4 above)
   if ((n_1 is not node "a") and (n_2 is not node "a")) then
   if ((n_1 is a node in M and n_2 is a node in N) then
   add (n_1, r, n_2) to δ'
   elseif (n_1 is a node in N and n_2 is a node in M) then
   add (n_1, r, n_2) to δ'
   end if
   end if
   end for
7. $\sum'$ is determined by analyzing δ' and comparing with $\sum$.
8. Q_0', and F' are to be determined from analyzing δ'.

Example 6: Swap a subset of an ontology.
Given OAM instance M_5 (Fig. 6), to swap subset (M_5, g). subset (M_5, g) is to be replaced by OAM instance N. The output of this example is M_6.
$M_5 = (Q_5, \sum_5, \delta_5, Q_{05}, F_5)$; it is shown in Example 5 and Fig. 6.
Use Algorithm 5.3, step 1 to get N.
$N = (Q, \sum, \delta, Q_0, F)$, which is shown in Fig. 8.
$Q = \{Concepts, h, s, ax\}$;
all nodes are classes in this case
$\sum = \sum_B \cup \sum_E = \{part\_of\}$
$\sum_B = \{part\_of\}$; $\sum_E$ is empty
$Q_0 = \{ax, s\}$
$F = \{Concepts\}$
$\delta = \{(s, part\_of, Concepts), (h, part\_of, Concepts), (ax, part\_of, h)\}$

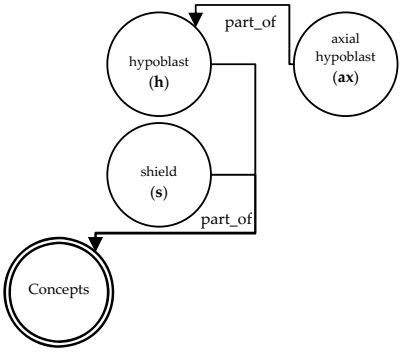

Fig. 8 Ontology N

Use Algorithm 5.3, step 2 to initialize M_6 to be an empty OAM, $M_6 = (Q_6, \sum_6, \delta_6, Q_{06}, F_6)$. Use Algorithm 5.3, step 3 to add nodes to M_6.
$Q_6 = Q_5$ - nodes of subset (M_5, g) $\cup \{g\}$
Nodes of subset (M_5, g) = $\{g, h, ep\}$
$Q_6 = \{Concepts, e, b, i, f, g\}$
Use Algorithm 5.3, step 4 to add edges to M_6.
$\delta_6 = \{\{(e, is\_a, Concepts), (b, is\_a, e), (g, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, g), (g, is\_contained\_in\_image, f)\}$
At this point M_6 is shown in Fig. 9.

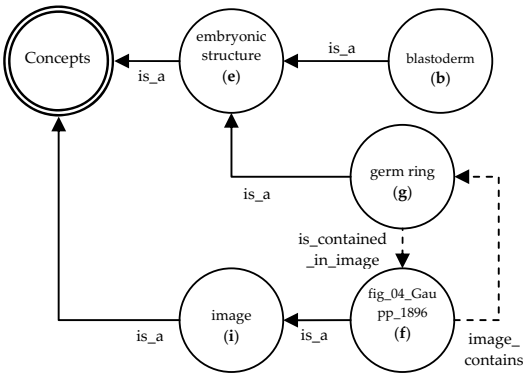

Fig. 9 Updated Ontology M_6 (subset swap, step 4)

Use Algorithm 5.3, step 5 to attach N to M_6. Now M_6 has the following nodes, Q_6 and edges, δ_6. At this intermediate stage M_6 is shown in Fig. 10.
$Q_6 = \{Concepts, e, b, i, f, g, s, h, ax\}$
$\delta_6 = \delta_6 \cup \{(s, part\_of, g), (h, part\_of, g), (ax, part\_of, h)\} = \{(e, is\_a, Concepts), (b, is\_a, e), (g, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, g), (g, is\_contained\_in\_image, f), (s, part\_of, g), (h, part\_of, g), (ax, part\_of, h)\}$

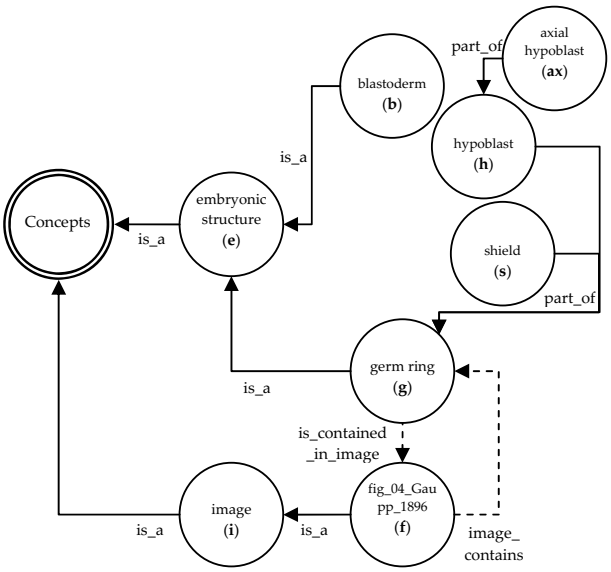

Fig. 10 Updated Ontology M_6 (subset swap, step 5)

Use Algorithm 5.3, step 6 to add edges to M_6.
$\delta_6 = \delta_6 \cup \{(h, is\_a, e)\}$
Now determine $\sum_6$, Q_{06}, and F_6.
As shown in Fig. 11, M_6 is the completed OAM instance after the subset swap operation.
$M_6 = (Q_6, \sum_6, \delta_6, Q_{06}, F_6)$
$Q_6 = \{Concepts, e, b, i, f, g, s, h, ax\}$
$\delta_6 = \{(e, is\_a, Concepts), (b, is\_a, e), (g, is\_a, e), (i, is\_a, Concepts), (f, is\_a, i), (f, image\_contains, g), (g, is\_contained\_in\_image, f), (s, part\_of, g), (h, part\_of, g), (ax,$

part_of, h), (h, is_a, e)}}
$\sum_6 = \sum_{B6} \cup \sum_{E6}$ = *{is_a, part_of, is_contained_in_image, image_contains}*
$\sum_{B6}$ = *{is_a, part_of}*; $\sum_{E6}$ = *{is_contained_in_image, image_contains}*
Q_{06} = *{b, ax, s, f}*
F_6 = *{Concepts}*

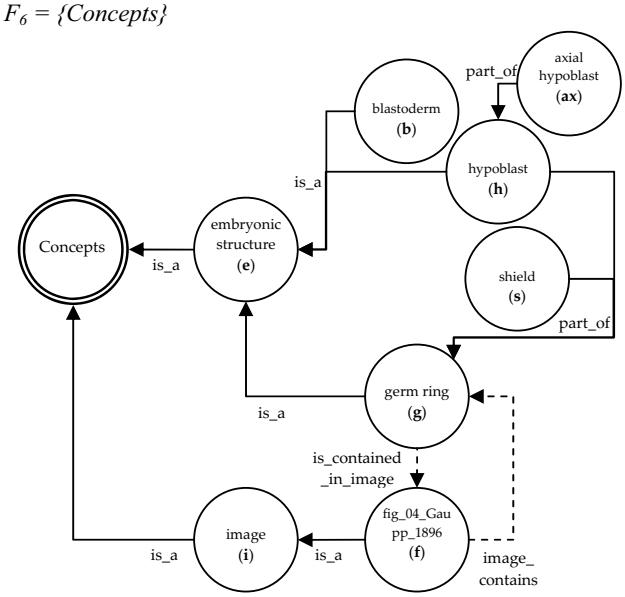

Fig. 11 Completed ontology M_6 (after subset swap)

Although Algorithm 5.3. satisfies the basic requirements of the *AmphibAnat* project, it also presents some interesting algorithm-related issues and future research opportunities.

As illustrated in Example 7, Algorithm 5.3 is efficient for replacing a subset of an ontology, and re-establishing edges (relationships) between the swapped subset and the original ontology. This is done under the assumption that the nodes (elements of Q) and relationships types (elements of $\sum$) are not updated (renamed) during the subset swap operation. Although in the *AmphibAnat* project such update operations are extremely rare during subset swaps, in general that assumption should not be made for the OAM algorithms.

There are basically two common types of possible updates (case 1 and case 2 below) during subset swaps. The cases are described below with reference to Algorithm 5.3 and Example 6.

Case 1:
Some elements of $\sum$ in N are the results of updated elements of $\sum$ in M. For example, $\sum_5$ of M_5 = *{part_of, ...}*; $\sum$ of N = *{is_part_of}*; *part_of* and *is_part_of* have the same meaning for the updated ontology.

Case 2:
Some elements of Q in N are the results of updated elements of Q in M. For example, Q_5 of M_5 = *{s, ...}*; Q of N = *{s'...}*; *s'* is the updated *s* in the ontology.

For case 1, a swap table is used to map the original relationship types to the updated relationship types when the relationship types of M and N do not exactly match. The user also has a choice of choosing one of the two relationship types for the updated ontology. The construction of the swap table and the final relationship type decision must be done before the execution of Algorithm 5.3. This is a simple operation (that can be done manually) because the size of $\sum$ is typically very small; for the *AmphibAnat* ontology modules, $|\sum_B|$ has a value of *2* to *,4* and $|\sum_E|$ has a value of *20* to *40*.

In Example 6, if the user chooses *is_part_of* as the new relationship type for M_6, the final value for $\sum_6 = \sum_{B6} \cup \sum_{E6}$ = *{is_a, is_part_of, is_contained_in_image, image_contains}*; $\sum_{B6}$ = *{is_a, is_part_of}*. The rest of M_6 is thus updated accordingly.

For case 2, different techniques could be used. The OAM-based initial approach was to use a swap table to store the original node names, updated node names, and the final node names. The user also can augment the swap table with a list of synonymous names. This approach is easy to implement technically, but requires the user's time to thoughtfully consider what synonyms should be added for the affected terms. The *AmphibAnat* ontology modules each have $|Q|$ of around *5,000* to *10,000*, so the future need for automating the synonym selection task is important. Another approach that could be used to automate the construction of an enhanced swap table is to use existing natural language processing algorithms to identify similar elements by matching Q of M and Q of N. User feedback will likely still be needed to validate the swap table, but this technique should significantly reduce the users' time for the overall task.

VI. SUMMARY

In this paper an ontology model called an Ontology Abstract Machine (OAM) and its related algorithms for various ontology maintenance tasks have been presented. The OAM model is not intended as a replacement for any ontology language; it is an abstract model, defined in a format similar to that for a finite state automaton. The objective of developing this model was to provide a framework for constructing a collaborative, Web-based ontology management system that required modularity and distinct relationship classifications. The successful implementation of the *RDBOM* software system (and its subsequent acceptance by a large research community) validates the OAM model and its real-world value.

The similarity between the OAM model and a nondeterministic finite automaton (NFA) [25] provides many interesting future research opportunities. NFA and regular language research is a well-studied computer science research area. Can those established theories and algorithms help answer unsolved ontology questions or identify new ontology challenges? A much more detailed analysis and comparison (between OAM and NFA) will be conducted in the future in the hopes of further enhancing the usefulness and usability of ontologies.

ACKNOWLEDGEMENTS

This work was supported by NSF under award DBI-0640053.

REFERENCES

[1] Buffalo Ontology Site: http://ontology.buffalo.edu/
[2] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, J. P. Rice, "OKBC: A programmatic foundation for knowledge base interoperability," in Proc. *15th national conference on artificial intelligence (AAAI-98)*, 1998.
[3] M. R. Genesereth, R. E. Fikes, Knowledge Interchange Format Version 3.0 Reference Manual: http://logic.stanford.edu/kif/Hypertext/kif-manual.html
[4] P. Lambrix, M. Habbouche, M. Pérez, "Evaluation of ontology development tools for bioinformatics," *Bioinformatics*, vol. 19, no. 12, August 2003, pp. 1564–1571.
[5] A. J. Duineveld, R. Stoter, M. R. Weiden, B. Kenepa, V. R. Benjamins, "WonderTools? A comparative study of ontological engineering tools," *International Journal of Human-Computer Studies*, vol. 52, no. 6, June 2000, pp. 1111–1133.
[6] A. Farquhar, R. Fikes, J. Rice, "The Ontolingua Server: a tool for collaborative ontology construction", *International Journal of Human-Computer Studies (1997)*, pp. 707–727.
[7] 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.
[8] B. Smith, et al., "The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration," *Nature biotechnology*, vol. 25, iss.11, pp.1251, 2007.
[9] Open Biological Ontologies: http://www.obofoundry.org/
[10] M. Klein, "Combining and relating ontologies: an analysis of problems and solutions," in Proc. *IJCAI workshop on Ontologies (IJCAI-2001)*.
[11] AmphibAnat: http://www.amphibanat.org/
[12] 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*, in review.
[13] J. Bao and V. Honava., "Divide and conquer semantic web with modular ontologies - a brief review of modular ontology language formalisms," *in Proc. ISWC2006 International Workshop on Modular Ontologies (WoMo2006)*.
[14] J. Bao and V. Honavar, "Ontology Language Extensions to Support Localized Semantics, Modular Reasoning, and Collaborative Ontology Design and Ontology Reuse," Technical Report, Department of Computer Science, Iowa State University, 2004.
[15] P. Kodeswaran, S. B. Kodeswaran, A. Joshi, T. Finin, "Enforcing security in semantics driven policy based networks," *ICDE Workshops 2008*, pp. 490–497.
[16] J. Bao, G. Swlutzki, V. Honavar, "Privacy-Preserving Reasoning on the Semantic Web," *IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 791–797, 2007.
[17] H. Luong, S. Gauch, and Q. Wang, "Ontology Learning Through Focused Crawling and Information Extraction," *International Conference on Information, Process, and Knowledge Management*, Cancun, Mexico, Feb. 1-7, 2009.
[18] OBO Edit: http://www.oboedit.org/
[19] OWL Web Ontology Language Guide: http://www.w3.org/TR/2004/REC-owl-guide-20040210/
[20] Stanford medical informatics Home page: http://protege.stanford.edu
[21] T. Finin, A. Joshi, L. Kagal, J. Niu, W. H. Winsborough, B. thuraisingham, "ROWLBAC – Representing Role Based Access Control in OWL", *in Proc. SACMAT'08,* 2008.
[22] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl .Acquis*, vol. 5, no. 2, pp.199–220, June 1993.
[23] J. de Bruijn, M. Ehrig, C. Feier, F. Martíns-Recuerda, F. Scharffe, and M. Weiten, "Ontology Mediation, Merging, and Aligning", in *Semantic Web Technologies*, July 2006, pp. 95-113.
[24] B. C. Grau, I. Horrocks, Y. Kazakov, and U. Sattler, "Just the right amount: extracting modules from ontologies", *in Proc. 16th International Conference on World Wide Web*, ACM, New York, NY, pp. 717-726.
[25] P. Linz, "An Introduction to Formal Languages and Automata", Jones & Bartlett, 4th edition, 2006, pp. 49–54.
[26] J. D. Richter, M. A. A. Harris, M. Haendel, S. Lewis, "OBO-Edit—an ontology editor for biologists", *Bioinformatics*, vol. 23, no. 16, 2007, pp. 2198–2200.