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

A Generic, Functionally Comprehensive Approach to Maintaining an Ontology as a Relational Database

Jennifer Leopold, Alton Coalter, and Leong Lee

Abstract**—**An ontology is a data model that represents a set of concepts in a given field and the relationships among those concepts. As the emphasis on achieving a semantic web continues to escalate, ontologies for all types of domains increasingly will be developed. These ontologies may become large and complex, and as their size and complexity grows, so will the need for multi-user interfaces for ontology curation. Herein a functionally comprehensive, generic approach to maintaining an ontology as a relational database is presented. Unlike many other ontology editors that utilize a database, this approach is entirely domain-generic and fully supports Web-based, collaborative editing including the designation of different levels of authorization for users.

Keywords**—**Ontology Editor, Relational Database, Collaborative Curation.

I. INTRODUCTION

AN ontology is a data model that represents a set of concepts in a given field and the relationships among those concepts. As stated in [1], "ontologies are becoming popular largely due to what they promise: a shared and common understanding of a domain that can be communicated between people and application systems." These hierarchically organized, standardized vocabularies facilitate the discovery, sharing, and integration of information, and thereby will serve an invaluable role in realizing a semantic web. Yet one should not lose sight of the fact that an ontology is still fundamentally a collection of related data, and should be maintained in such a way as to ensure consistency of the information, while transparently providing concurrent, authorized access to the data. These requirements can most easily be addressed by utilizing a database management system to maintain the ontology.

Recently there have been various efforts to represent ontologies as relational databases [2]. However, for the most part, these efforts simply have been for the purpose of querying the data contained in the ontology, not for the daily management of the information by multiple users. Instead, updates to ontologies typically require the use of ontology editor tools (which are not necessarily built upon relational database management systems), most of which do not support multiple users, collaborative editing, or the designation of different levels of authorization for given users. Furthermore, many of the systems that do utilize a relational database tightly couple the schema to a particular ontology domain (e.g., there may be a relational table for each concept defined in the ontology), resulting in the need for customized software applications to access and maintain the particular ontology.

Herein a functionally comprehensive, generic approach to maintaining an ontology as a relational database is presented. Implemented as a Web-based software system called *RDBOM* (Relational Database Ontology Maintenance), this approach exploits: (1) 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 of the ontology, and (2) the features of a Web-based application to facilitate true community curation of the ontology.

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

II. BACKGROUND AND RELATED WORK

The traditional data model for an ontology is a directed graph that represents a set of triples, each of the form *(subject, predicate, object)*. The nodes of the graph are the *subject*s and *object*s, both of which are commonly referred to as *concept*s or *class*es. As in the object-oriented programming paradigm, an *instance* of a *concept* also can be defined. Each edge in the graph denotes the *predicate* (also known as a *property* or *relationship*) that relates a *subject* to an *object*. The most commonly used relationships in ontologies are *is_a* and *part_of*; however, other relationships such as *synonym_of* are often defined by the user. Ontology graphs are typically acyclic, and may or may not be rooted.

Because of the important role of relationships in ontologies, a relational database would seem to be an obvious choice for their implementation. In recent years there has been a considerable amount of work done in this area, with the following (practical) motivations:

- to enhance querying capabilities of the ontology,
- to utilize data types not normally supported in some ontology data formats,
- to provide a translation to and from an XML-based representation (e.g., OWL, DAML, OIL, RDF), and/or
- to facilitate Web-based access to the ontology data (e.g., via PHP or ASP).

In those systems that provide a translation from an XML-based representation, a dominant paradigm has been to map each concept in the ontology to a distinct relational database table (e.g., [3] and [4]), a strategy that would seem better suited for an object-oriented database. This tightly couples the ontology to a particular domain, resulting in the need for customized queries and maintenance applications. A partial exception to this domain-specific approach is the Gene Ontology (GO) [5] which, in addition to having tables specifically designed to contain information about genes, also includes more generic tables to represent the ontology as a directed graph and to maintain pre-computed transitive closures of commonly referenced relations within the ontology.

Regardless of whether or not an ontology editor is implemented with a relational database, the following features should be supported in order to provide end-users with a functionally comprehensive, collaborative query and maintenance system for their ontology:

- Queries: users should be able to perform both *structural* queries (e.g., finding ancestors and descendents of a concept, finding all defined relations between two concepts, etc.), and *content-based* queries (e.g., finding a term that contains a particular phrase).
- Updates: users should be able to perform both *structural* updates (e.g., moving a concept to a different location within the graph structure of the ontology, deleting or inserting a term, etc.), and *content-based* updates (e.g., modifying the value of an existing concept or property).
- Collaboration: the system should facilitate concurrent access by multiple users, with identification of who made what changes and when.
- Version control: some mechanism should be in place for logging updates to the data so that previous versions of the ontology can be recalled later.
- Security: the system must provide security in terms of user authorization for queries and updates; preferably, update authorization should be node-based in the sense that a particular user can be restricted to accessing only certain "branches" of the ontology.

TABLE. I SUMMARY OF FEATURES OF SOME DOMAIN-GENERIC, MULTI-USER ONTOLOGY EDITORS

Functional Comparisons:	RDBOM	Chimaera	KADS22	KAON	OBOedit	ODE	OilEd	Ontolingua	OntoSaurus	OpenCyc	PC Pack 5	Protégé	Vysniauskas	WebOnto
Structural queries	✓	–	✓	✓	✓	✓	–	–	×	–	–	–	×	–
Content-based queries	✓	–	✓	✓	✓	✓	–	–	✓	–	–	–	–	–
Structural updating	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	✓	×	–
Content-based updating	✓	✓	✓	✓	✓	✓	✓	–	✓	–	–	✓	–	–
Collaborative commenting	✓	×	×	–	×	×	×	×	×	×	–	–	×	–
Security, node-based per user	✓	×	×	×	×	×	×	×	×	×	×	×	×	×
Version control, logged user transactions	✓	×	n/a	–	✓	n/a	×	×	×	×	×	×	×	×
Web-based access	✓	✓	×	×	×	×	×	×	✓	✓	✓	×	×	✓
If RDB, is each class mapped to a table	N	n/a	×	N	×	n/a	×	×	×	n/a	n/a	x	Y	n/a
Commercial	N	N	N	N	N	N	N	N	N	Y	Y	N	N	N
✓ = functionality exists - = limited functionality exists × = functionality does not exist														

There are several ontology maintenance systems available; see [6] for an extensive list. However, many are not domain-generic and/or are not multi-user systems. Table 1 summarizes which of the above features are supported by fourteen ontology editor systems (including the *RDBOM* system introduced in this paper) that are domain-generic and support some degree of concurrent access.

Almost all of the ontology maintenance systems listed in Table 1 support some degree of structural and content-based querying and updating. However, despite providing multi-user access, many of these systems lack important features necessary to facilitate collaborative development of the ontology; namely, Web-based access, node-based user access control, transaction-based version control, and a node-based

forum for user discussion. In contrast, the *RDBOM* system does support all of these features.

The implementation of *RDBOM* in terms of the underlying data representation and the end-user functionality will be discussed next. The actual use of this system in terms of workflow, including the incorporation of ontology language translators, will then be addressed.

III. Data Representation

In order to facilitate data exchange with ontologies in OWL [7] and OBO [11] formats, the *RDBOM* relational database schema was designed to accomodate the expressivity of both of those languages. The relational table schemas used by *RDBOM* are as follows:

```
-- Databases distinguish separate ontologies
TABLE databases (
  id int IDENTITY(1,1),
  name varchar(64)
  )

-- Term types distinguish the usage of terms within
-- an ontology (e.g., class, instance, property,
-- restriction, etc.)
TABLE term_types (
  id int IDENTITY(1,1),
  name varchar(800)
  )

-- A term can be a class, instance, property, attribute,
-- restriction, or any other semantic unit of the ontology
TABLE terms (
  id int IDENTITY(1,1),
  name varchar(800),
  db_id int
  )

-- Relationship between a term and a specific type
-- of usage for the term.
TABLE term_usages (
  id int,
  term_id int,
  term_type_id int
  )

-- Relationship between two terms, specified along with
-- the edge joining them
TABLE term2terms (
  id int IDENTITY(1,1),
  term1_usage_id int,
  relation_term_id int,
  term2_usage_id int
  )


-- Unnamed concepts such as enumerations, unions,
-- and intersections.
TABLE restrictions (
  id int IDENTITY(1,1),
  term_id int,
  value_relation_id int,
  value_term_id int,
  property_term_usage_id int,
  union_term_id int
  )

-- An unnamed concept formed by the union of
-- other named classes
TABLE unions (
  id int IDENTITY(1,1),
  class_term_id int
  )

-- Values used in translating RDBOM terms to and
-- from terms used by other ontology formats (for
-- import and export)
TABLE translations (
  term varchar(256),
  translation varchar(256),
  format char(4)
  )

-- Data related to user discussions
TABLE forum (
  term_id int,
  User_ID int,
  DateTime datetime,
  comments varchar(2000)
  )

-- Metadata concerning relational properties
TABLE properties (
  id int IDENTITY(1,1),
  tree char(1),
  transitive char(1),
  cyclic char(1),
  reflexive char(1),
  symmetric char(1),
  antisymmetric char(1),
  metadata char(1),
  inverse_id int,
  term_id int,
  indicator varchar(100)
  )
```

In addition, the following schemas are used to maintain information on user accounts and node-based access permissions:

```
-- All users of the system must be registered for any
-- access beyond querying
TABLE users (
  ID int IDENTITY(1,1),
  Login varchar(128),
  Password varchar(50),
  Title varchar(5),
  Firstname varchar(128),
```

```
    Lastname  varchar(128),
    Affiliation  varchar(256),
    Researchinterests  varchar(512)
    )

--    Administrative users can perform tasks beyond
--    that of other users, such as granting access to
--    other users
TABLE admin_users (
    ID  int  IDENTITY(1,1),
    Login  varchar(128),
    Password  varchar(50),
    Title  varchar(5),
    Firstname  varchar(128),
    Lastname  varchar(128),
    Affiliation  varchar(256)
    )

--    Authorization levels for users identify query and
--    update privileges
TABLE authorization_levels (
    id  int  IDENTITY(1,1),
    description  varchar(128)
    )

--    Authorization levels are assigned at the topmost
--    node(s) that can be accessed by the user
TABLE authorizations (
    user_id  int,
    authorization_id  int,
    term_id  int
    )
```

It is important to note that the entire *RDBOM* schema is generic in the sense that it could be used to represent an ontology for any domain. There are also additional relational tables in *RDBOM* that support the storage of multimedia data such as image and sound files, and their subsequent association with ontology terms.

As an example of how some of the database tables would be populated, consider the portion of an ontology for anatomy [8] that is shown in Fig. 1, where the "I" icon represents an *is_a* relation. Examples of other available icons (not shown in Fig. 1) include a "P" icon to represent a *part_of* relation and a "D" icon to represent a *develops_from* relation.

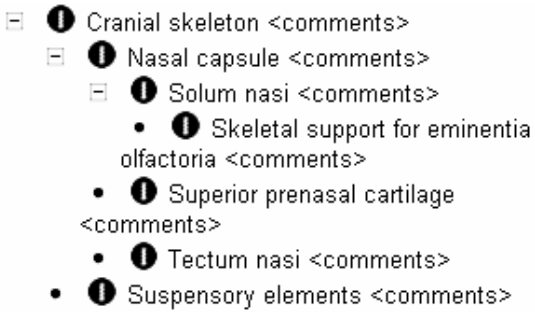

Fig. 1 Portion of an ontology of amphibian anatomy

The corresponding OWL representation for the ontology shown in Fig. 1 is as follows:

```
<owl:Class rdf:about="#Cranial_skeleton"/>
 <owl:Class rdf:about="#Superior_prenasal_cartilage">
  <owl:disjointWith rdf:resource="#Solum_nasi"/>
  <rdfs:subClassOf rdf:resource="#Nasal_capsule"/>
  <owl:disjointWith rdf:resource="#Tectum_nasi"/>
 </owl:Class>
 <owl:Class rdf:about="#Tectum_nasi">
  <owl:disjointWith rdf:resource="#Solum_nasi"/>
  <owl:disjointWith rdf:resource"#Superior_prenasal_cartilage"/>
  <rdfs:subClassOf rdf:resource="#Nasal_capsule"/>
 </owl:Class>
 <owl:Class rdf:about="#Nasal_capsule">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Cranial_skeleton"/>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#Suspensory_elements"/>
 </owl:Class>
 <owl:Class rdf:about="#Solum_nasi">
  <owl:disjointWith rdf:resource="#Superior_prenasal_cartilage"/>
  <owl:disjointWith rdf:resource="#Tectum_nasi"/>
  <rdfs:subClassOf>
   <owl:Class rdf:about="#Nasal_capsule"/>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:about="#Suspensory_elements">
  <rdfs:subClassOf rdf:resource="#Cranial_skeleton"/>
  <owl:disjointWith rdf:resource="#Nasal_capsule"/>
 </owl:Class>
 <owl:Class rdf:about="#Skeletal_support_for_eminentia_olfactoria">
  <rdfs:subClassOf rdf:resource="#Solum_nasi"/>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:allValuesFrom>
     <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about="#Solar_ossification_of_eminentia"/>
       <owl:Class rdf:about="#Turbinal_fold"/>
      </owl:unionOf>
     </owl:Class>
    </owl:allValuesFrom>
    <owl:onProperty>
     <owl:SymmetricProperty rdf:about="#is_Synonym_Of"/>
    </owl:onProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
 </owl:Class>
 <owl:Class rdf:about="#Turbinal_fold">
  <rdfs:subClassOf>
   <owl:Class rdf:about="#English_term"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty>
     <owl:SymmetricProperty rdf:about="#is_Synonym_Of"/>
    </owl:onProperty>
    <owl:allValuesFrom>
     <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
       <owl:Class rdf:about
       ="#Skeletal_support_for_eminentia_olfactoria"/>
       <owl:Class rdf:about="#Solar_ossification_of_eminentia"/>
      </owl:unionOf>
     </owl:Class>
    </owl:allValuesFrom>
   </owl:Restriction>
  </rdfs:subClassOf>
 </owl:Class>
```

A portion of the corresponding *RDBOM* representation for the ontology shown in Fig. 1 is as follows:

terms	
ID:	*NAME:*
2	rdfs:subClassOf
4	rdfs:comment
9	owl:disjointWith
33	owl:allValuesFrom
35	is_Synonym_Of
81	owl:unionOf
127	Superior_prenasal_cartilage
132	Tectum_nasi
134	Solum_nasi
139	Nasal_capsule
192	Suspensory_elements
196	Cranial_skeleton
362	Floor of nasal capsule.
364	Skeletal_support_for_eminentia_olfactoria
365	Turbinal_fold
366	Solar_ossification_of_eminentia

term_usages		
ID:	*TERM_ID:*	*TERM_TYPE_ID:*
30	35	3
114	127	1
119	132	1
121	134	1
128	139	1
183	192	1
186	196	1
353	366	1
488	365	1
520	364	1

term_types	
ID:	*NAME:*
1	owl:Class
3	owl:SymmetricProperty

term2terms			
ID:	*TERM1_USAGE_ID:*	*RELATION_TERM_ID:*	*TERM2_USAGE_ID:*
1244	121	9	114
1246	121	2	128
1248	121	4	351
1265	121	9	119
1271	128	9	183
1272	128	2	186
1325	353	2	99
1562	186	2	42
1873	183	2	186
1875	183	4	410
1876	183	9	128
2143	119	9	121
2146	119	9	114
2152	119	2	128
2153	119	4	436
2895	114	2	128
2899	114	9	121
2915	114	9	119
2916	114	4	471
3129	488	2	99
3466	520	2	121

restrictions					
ID:	*TERM_ID:*	*VALUE_RELATION_ID:*	*VALUE_TERM_ID:*	*PROPERTY_TERM_USAGE_ID:*	*UNION_TERM_ID:*
32	366	33	81	30	12
74	365	33	81	30	30

unions	
ID:	*CLASS_TERM_ID:*
12	364
12	365
30	364
30	366
36	366
36	365

IV. END-USER FUNCTIONALITY

RDBOM was implemented as a Web-based application that allows users to query and update an ontology that is represented using the relational database schema given in the previous section. Available tools to translate an ontology from a language such as OWL to this relational database structure will be discussed in Section V.

Any user can add a comment about any class in the ontology, and can view the comments of other users. Query and update permissions are assigned by the ontology administrator at the node level; specifically, access is granted at the root node of each "branch" of the ontology for which the user is to be allowed update and/or query access.

A. Queries

As listed in the ontology maintenance requirements presented earlier, users should be able to perform both *content-based* queries (e.g., finding a term that contains a particular phrase), as well as *structural* queries (e.g., finding ancestors and descendents of a class, finding all relationships between two terms, etc.). In this section the queries that are currently available in *RDBOM* are discussed.

Content-based queries should allow the user to search the ontology based on the textual content of any class, instance, or property. To find any term containing a particular phrase, it is sufficient to search the *terms* table. To find all references to a particular class, the *term2terms* table can be searched, subsequently retrieving a term by joining the *terms* table with the *term_usages* table.

To find all entities containing a given phrase in a given property the *terms* table is used in conjunction with the *terms_usage* table to find the rows from the *term2terms* table that match the specified property and phrase. Similarly, to find all entities for which data are known for a given property, the search is performed for non-null values for the term.

To find all entities that are related via a particular relationship, a join is performed between the *terms*,

term_usages, and *term2terms* tables as in the following SQL statement, where the values represented using angle brackets ('<' and '>') represent actual literal values for the particular data:

```
SELECT DISTINCT a.name, a.id
  FROM terms a, terms b, terms c, term_usages u,
          term_usages v, term2terms t
  WHERE t.term1_usage_id = u.id
    AND t.relation_term_id = b.id
    AND t.term2_usage_id = v.id
    AND u.term_id = a.id
    AND v.term_id = c.id
    AND a.db_id = <DBID>
    AND b.name = '<RELATION>'
    AND c.name = '<ENTITY>'
  ORDER BY a.name
```

The results for each of the content-based queries is a linear list of a terms, any of which can be clicked on to select the corresponding node in the ontology tree display.

Given the hierarchical organization of ontologies, it is likely that the user might want to find the ancestors and descendants of a term, as well as the structural relationships between two terms. In a large ontology, such information may be difficult to determine simply by navigating a graphical (tree) display.

To find all ancestors of a particular class in *RDBOM*, the following SQL statement (using the returned values in place of '<NODEID>') is recursively executed until each call fails to return any rows:

```
SELECT a.name, a.id
  FROM terms a, terms h, term_usages d,
          term_usages e, term_usages f,
         term2terms c, properties p
  WHERE c.term1_usage_id = d.id
    AND c.relation_term_id = p.term_id
    AND p.tree = 'Y'
    AND h.id = e.term_id
    AND e.id = c.term1_usage_id
    AND a.id = f.term_id
    AND f.id = c.term2_usage_id
    AND d.term_id = <NODEID>
    AND a.db_id = <DBID>
  ORDER BY a.name
```

Each invocation of this query joins the term from the *terms* table (via the *terms_usage* table) to the *term2terms* table, restricting the relationship to parent-child relations (e.g. *subclass*, *part_of*, or *instance_of*) and using <NODEID> as the child class. The <NODEID> value is used instead of the name in the joins because it is guaranteed to be unique, whereas the name itself is not.

To find all descendants of a particular class, the following query is recursively executed to retrieve all subclasses of that class, using the results returned as the <NODEID> value for subsequent calls:

```
SELECT a.name, a.id, p.indicator
  FROM terms a, terms b, term_usages u,
          term_usages v,
         term2terms t, properties p
  WHERE t.term1_usage_id = u.id
    AND t.relation_term_id = b.id
    AND t.term2_usage_id = v.id
    AND u.term_id = a.id
    AND b.id = p.term_id
    AND p.tree = 'Y'
    AND v.term_id = <NODEID>
  ORDER BY a.name
```

The returned value p.indicator provides information about the particular relationship of the child to its parent; for example, this would show whether the child is related to its parent with an *is_a* relationship, or a *part_of* relationship.

In order to retrieve all associated instances, the following SQL statement is then executed for each class that was found by the previous query:

```
SELECT f.name, f.id, d.name, a.name, a.id, y.name
  FROM terms a, term_usages b, term2terms c,
          terms d, term_usages e, terms f,
          term_types y
  WHERE a.id = b.term_id
    AND b.term_type_id = y.id
    AND b.id = c.term2_usage_id
    AND c.relation_term_id = d.id
    AND d.id = <RELID>    -- from above query
    AND e.id = c.term1_usage_id
    AND e.term_id = f.id
    AND f.id = <NODEID>
  ORDER BY d.name, f.name, a.name
```

As with the content-based queries, the result of finding the ancestors or descendants of a class is a linear list. Any entry in the result list can then be clicked on to select the corresponding node in the ontology tree display.

In order to obtain a list of all relationships between two concepts, the specified classes are joined via the *terms* table and the *term_usages* table to the *term2terms* table, as in the following SQL statement:

```
SELECT a.name, b.name, c.name
  FROM terms a, terms b, terms c, term_usages d,
          term_usages e, term2terms f
  WHERE a.id = d.term_id
    AND d.id = f.term1_usage_id
    AND b.id = f.relation_term_id
    AND c.id = e.term_id
    AND e.id = f.term2_usage_id
    AND a.db_id = <DBID>
    AND ((a.name = '<TERM1>' AND
            c.name = '<TERM2>')
      OR (a.name = '<TERM2>' AND
            c.name = '<TERM1>'))
  ORDER BY c.name, a.name
```

To find the closest shared parent of two classes, it is necessary to first find all ancestors for one class and store those values in a list. The ancestors for the second class can then be found, checking each value to see whether it is contained in the list for the ancestors of the first class. This recursive search is terminated as soon as a match is found. The resulting term is then highlighted in the ontology tree display, and both paths are displayed for the user.

B. Updates

As listed in the ontology maintenance requirements discussed in Section II, users should be able to perform both *content-based* updates (e.g., modifying the value of an existing term or property) as well as *structural* updates (e.g., moving, deleting, or inserting a class, etc.). To provide support for version control (with the potential of later rolling back changes), the *RDBOM* system logs a description of each update in an administrative table of the relational database, including a timestamp and the identification of the user who issued the update.

An example of the *RDBOM* user-interface for performing content-based updates is shown in Fig. 2. Once a class is selected for update in the ontology tree display, the user can rename the node, modify the definition, add properties (that have been previously defined by the administrators of the ontology), or add instances. These updates largely just affect the *terms* table and possibly the *term2terms* table in the database.

An example of the *RDBOM* user-interface for performing structural updates is shown in Fig. 3. Once a class is selected for update in the ontology tree display, the user can create a new "child" node beneath it, move the node to another location in the tree (using the "cut" and "paste" functions), or delete the node. Currently, only "leaf" nodes in the ontology tree can be moved or deleted.

To create a child node, a new class is added to the *terms* table, and an entry is made in the *term_usages* table. It is also necessary to make appropriate entries in the *term2terms* table of the database for the intended 'subClass' relation.

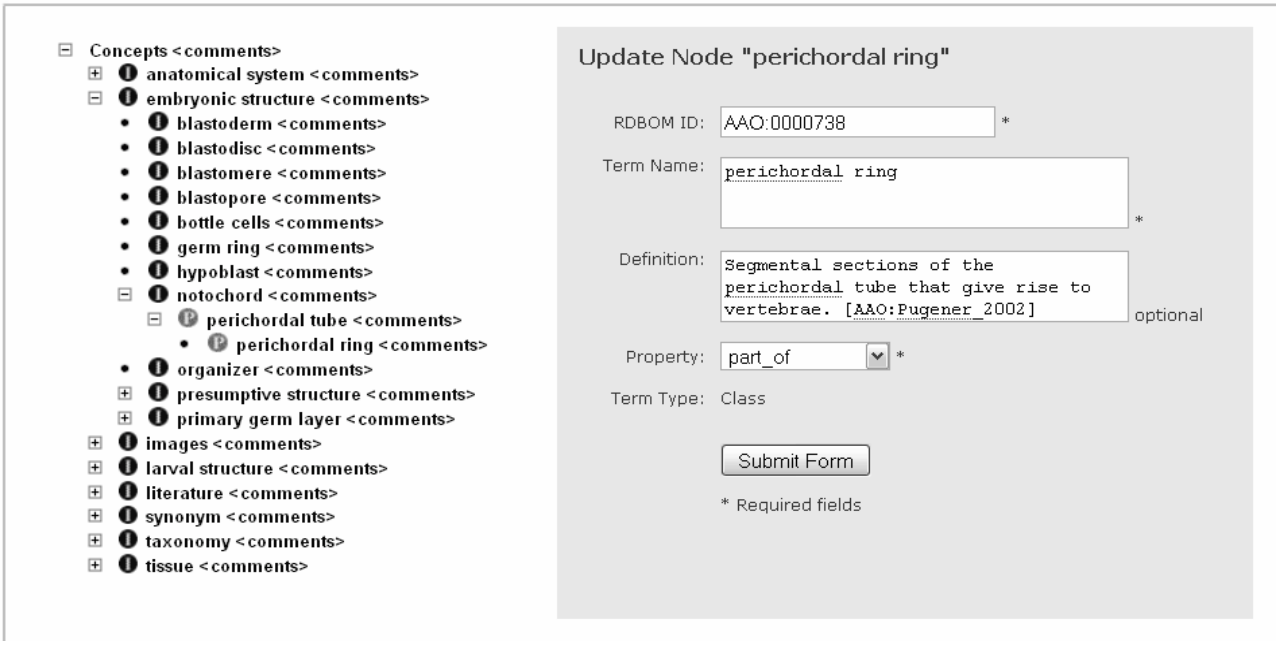

Fig. 2. *RDBOM* content-based update web interface

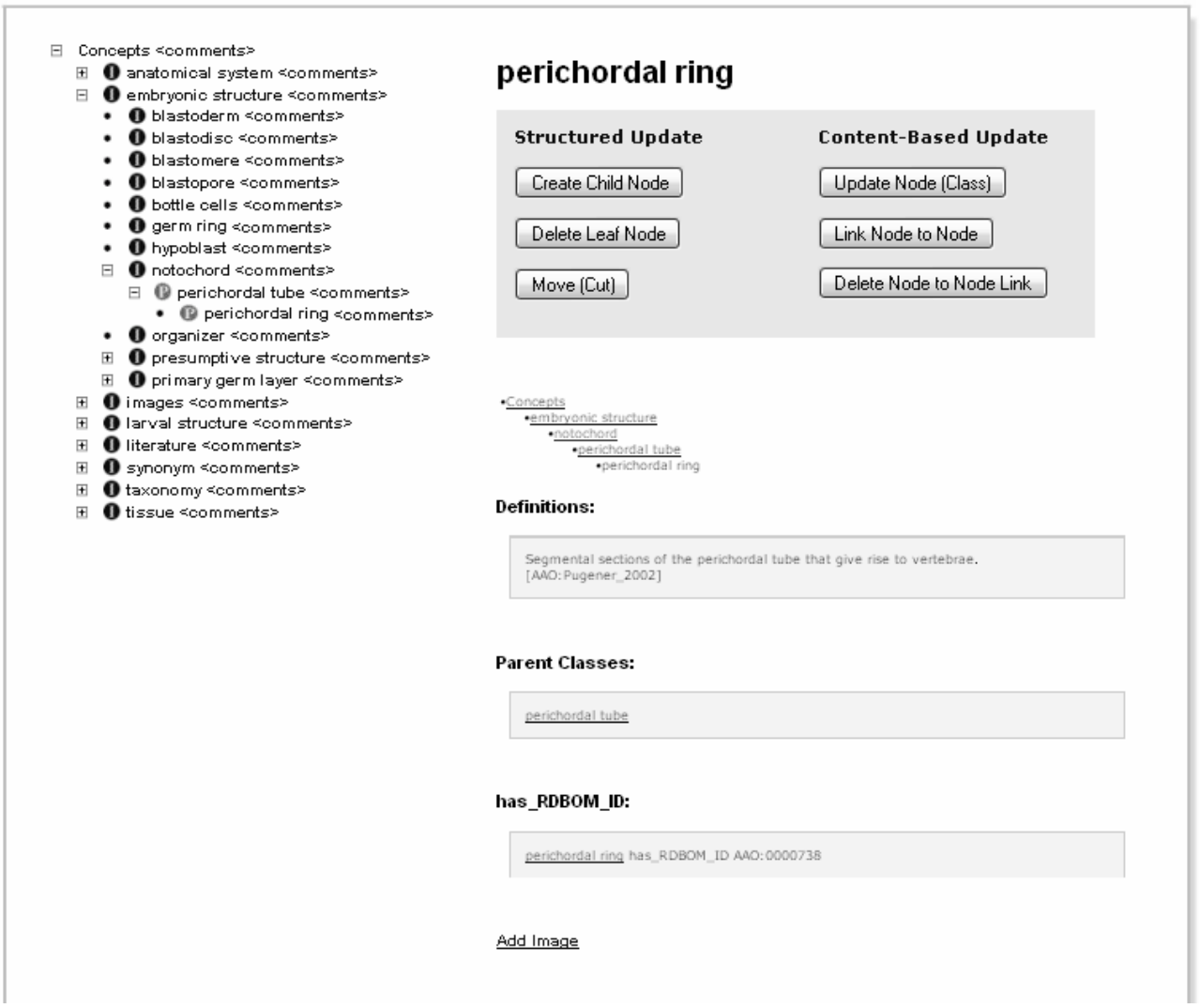

Fig. 3. *RDBOM* structural update web interface

In order to delete a leaf node, it is necessary to delete all relationships involving the term, not just the *subClass* or *instanceOf* relations. These updates can be performed with the following atomic transaction:

```
// To delete the node, first delete all records from
//  restrictions, unions, and term2terms,
// then from term_usages, and finally from terms.
BEGIN TRANSACTION;
// Find the id of the correct terms record.
SELECT id AS Term_ID
  FROM terms
  WHERE name = '<TERM>';
// Find the id of the correct term_usages record.
SELECT id AS Usage_ID
  FROM term_usages
  WHERE term_id = <Term_ID>;
// Using the values from the above SQL statements
// (Term_ID and Usage_ID),
// delete any restrictions or unions for this term.
DELETE FROM restrictions
  WHERE term_id = Term_ID
    OR value_term_id = Term_ID
    OR union_term_id = Term_ID;
DELETE FROM unions
  WHERE class_term_id = Term_ID;
// Remove any forum records or authorizations
// for this term.
DELETE FROM authorizations
  WHERE term_id = Term_ID;
DELETE FROM forum
  WHERE term_id = Term_ID;
// Delete any relationships involving this term.
DELETE FROM term2terms
  WHERE term1_usage_id = Usage_ID
    OR term2_usage_id = Usage_ID;
// Delete orphaned term_usages rows.
DELETE FROM term_usages
```

```
WHERE id IN
  (SELECT u.id
    FROM term_usages u, terms a
    WHERE u.id NOT IN
      (SELECT DISTINCT
            term1_usage_id FROM term2terms)
        AND u.id NOT IN
      (SELECT DISTINCT
            term2_usage_id FROM term2terms)
        AND u.term_type_id =
      (SELECT id FROM term_types
            WHERE name = 'class')
        AND u.term_id = a.id
        AND a.db_id = <DBID>);
// Finally delete the actual terms and end
// the transaction.
DELETE FROM terms WHERE id = Term_ID;
COMMIT TRANSACTION;
```

In order to move a node, it is sufficient to replace the parent's reference in the *term2terms* table with the reference to the new parent; all other relationships remain unchanged. It is necessary to first find the unique record id of the row to be updated in the *term2terms* table using the node names of the child and the current parent. In the following SQL statement, <USAGE_ID> is the usage ID of the node to be moved (from the *term_usages* table), <PARENT_ID> is the usage Id for the current parent node, and <NEWPARENT_ID> is the corresponding value for the new parent node:

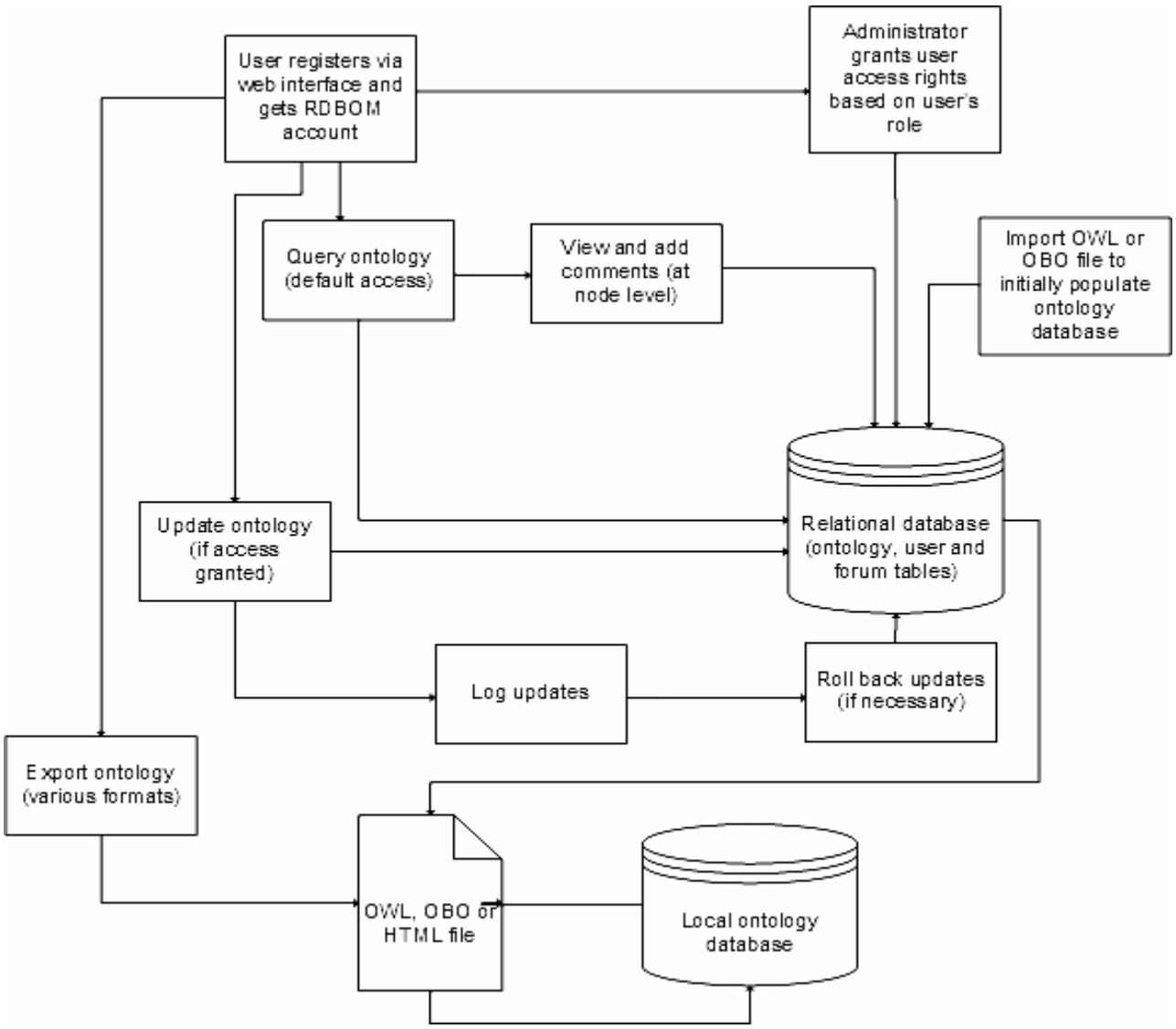

Fig. 4. *RDBOM* collaborative workflow

```
SELECT id
  FROM term2terms
  WHERE term1_usage_id = <USAGE_ID>
   AND term2_usage_id = <PARENT_ID>
   AND relation_term_id IN
    (SELECT term_id
      FROM properties
      WHERE ptree = 'Y')
```

An update is then made to the *term2terms* table as follows where <T2T_ID> is the value returned by the previous SQL statement:

```
UPDATE term2terms
  SET term2_usage_id = <NEWPARENT_ID>
  WHERE id = <T2T_ID>
```

Although other types of updates to ontologies are possible (e.g., defining new relations, adding new restrictions on relations, etc.) and are supported in ontology editors such as Protégé, this type of functionality has not been provided for general users in *RDBOM*; only administrators have permission to make such changes via an administrative update utility which is similar in design to the content-based and structural update interfaces that have been presented.

V. Workflow

The Web-based *RDBOM* software system provides a collaborative development environment for querying, maintaining, and annotating ontologies through a central relational database. The workflow process is diagrammed in Fig. 4, both from the perspective of users as well as administrators.

The ontology database can be defined from scratch, or initially populated from an OWL or OBO file by using a translation program that is provided with the *RDBOM* system. The translator utilizes a C#/asp.net framework to parse the import file entries and to insert the transformed data into the *RDBOM* database via the database management system's SQL calls.

Using a Web interface, a user must register for an account to access the ontology. Upon notification of the registration, the administrator can create a user account and grant access rights based on the user's intended role in the development of the ontology. Query and update permissions are assigned at the node level; specifically, access is granted at the root node of each "branch" of the ontology for which the user is to be allowed update and/or query access. To facilitate collaboration, users can also annotate any node in the ontology with comments, and view the comments of other users.

Updates made by users are automatically recorded in a transaction log. Each log entry contains a description of the update, a timestamp, and the user's account identification. This information is only available to *RDBOM* administrators, and can be used to monitor progress on the development of the ontology. The transaction log can also be used by the administrator to effectively undo a particular update. Currently, the undo facility is manual in the sense that the administrator must make the necessary edits to the ontology via the user update menu, and determine what other updates are required due to possible cascading effects. In the near future, this process will be automated.

To facilitate exporting the ontology to other data formats, translators for OWL and OBO are provided that maintain all information about the concepts, instances, and relationships in the ontology. This allows the user the option to utilize other ontology editors such as Protégé, or to simply display the ontology (as HTML) on a web page. It should be noted that if the ontology is modified in another ontology editor system, the database can be rebuilt later by using the *RDBOM* import utility (assuming that the ontology is in an OBO or an OWL format).

VI. Future Work

RDBOM currently is being used by a consortium of approximately 50 biologists who are developing an ontology for amphibian anatomy (*AmphibAnat*, www.amphibanat.org). This ontology currently contains over 10,000 classes, and is expected to grow to approximately 50,000 classes over the next three years.

In a recently held workshop of 28 amphibian biologists, a survey was administered to 21 of the (U.S. and international) participants who did not have much prior experience using other ontology editors. The results of the survey (shown in Table 2) are promising, and have identified areas for improvement in the user interface. The usability and usefulness of *RDBOM* will be more extensively evaluated in the near future with participants from a more diverse background. The usefulness and usability of *RDBOM* also should be empirically compared to other popular ontology editors such as OBO-EDIT and Protégé.

Additionally, the *RDBOM* database schema and software will be modified in the future to support modular ontologies as discussed in [9, 10]. This will facilitate merging, swapping, and comparison functions across multiple ontologies, and will further enhance ontology reuse.

VII. SUMMARY

As the emphasis on achieving a semantic web continues to escalate, ontologies for all types of domains increasingly will be developed. These ontologies may become large and complex, and as their size and complexity grows, so will the need for controlled-access, multi-user interfaces for ontology curation. The generic, functionally comprehensive approach implemented in *RDBOM* should facilitate collaborative development of such ontologies.

TABLE II SURVEY RESULTS

Question	avg	min	max
1. How much previous experience have you had with ontologies? (none = 1, a lot = 10)	2.5	1	7
2. How much experience have you had with the Protégé? (never used it = 1, proficient in using it = 10)	1.0	1	2
3. How much experience have you had with the OBO-Edit ? (never used it = 1, proficient in using it = 10)	1.1	1	2
4. Overall, does RDBOM allow you to perform the tasks necessary to query and modify an ontology? (never = 1, sometimes = 5, always = 10)	6.0	3	9
5. Overall, how would you rate the number of functions that are available to query and modify an ontology in RDBOM? (too few = 1, just right = 5, too many = 10)	5.2	3	10
6. Do you feel that you get sufficient feedback after doing operations in RDBOM? (never = 1, sometimes = 5, always = 10)	7.0	2	10
7. Overall, how easy is it to perform various operations in RDBOM? (difficult = 1, easy = 10)	5.4	2	10
8. How easy is it to navigate between the different windows in RDBOM? (difficult = 1, easy = 10)	5.1	1	9
9. How well does RDBOM distinguish ontology classes, properties and instances? (bad = 1, good = 10)	5.1	1	9
10. Do you feel that the RDBOM user interface is designed in a uniform way? (is not uniform = 1, is very uniform = 10)	7.6	2	10
11. How would you describe your experience using RDBOM? (boring = 1, fun = 10)	6.3	2	10
12. How would you rate the amount of documentation that is available in RDBOM? (not enough = 1, adequate = 10)	6.8	2	10
13. How easy is to understand the meaning of the RDBOM icons and menus? (difficult = 1, easy = 10)	7.1	2	10
14. How easy is the RDBOM terminology to understand? (never = 1, sometimes = 5, always = 10)	5.6	1	8
15. How easy is it to use the RDBOM documentation? (difficult = 1, easy = 10)	7.3	4	9
16. Overall, how clear is the purpose of each RDBOM function? (unclear = 1, clear = 10)	6.0	1	10
17. How easy is it to learn the RDBOM interface for ontology management? (difficult = 1, easy = 10)	6.8	4	9
18. How easy is it to remember how to use RDBOM? (difficult = 1, easy = 10)	6.9	4	10

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Davies, J., Fensel, D., and Van Harmelen, F. (eds.) (2004), *Towards the Semantic Web: Ontology-driven Knowledge Management.* West Sussex, England, John Wiley & Sons.
[2] Goodwin, R., Lee, J., Stanoi, G., and Leveraging, M. (2005), "Relational Database Systems for Large-Scale Ontology Management," *CIDR Conference*, www.alphaworks.ibm.com/ topics/semantics.
[3] Vysniauskas, E. and Nemuraite, L. (2006), "Transforming Ontology Representation from OWL to Relational Database," *Information Technology and Control*, vol. 35, no. 3A.
[4] Nyulas, C., O'Connor, M., and Tu, S. (2007), "DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé," *10th International Protégé Conference*, July 15-18.
[5] "GO Database Guide," *The Gene Ontology Consortium — Amigo*, www.geneontology.org/ GO.database.shtml.
[6] Denney, M. (2002), "Ontology Building: A Survey of Editing Tools," www.xml.com/pub/a/ 2002/11/06/ontologies.html
[7] Dean, M. and Schreiber, G. (eds) (2004), "OWL Web Ontology Language Reference," *W3C Recommendation*, www.w3.org/TR/2004/REC-owl-ref-20040210.
[8] Maglia, A., Leopold, J., Pugener, A., and Gauch, S. (2007), "An Anatomical Ontology for Amphibians," *Proceedings of the Pacific Symposium on Biocomputing (PSB 2007),* Wailea, Maui, pp. 367-378.
[9] Hitzler, P., Krotzsch, M., Ehrig, M., and Sure, Y. (2005), "What is Ontology Merging?" *American Association for Artificial Intelligence.*
[10] Cuenca-Grau, B., Parsia, B., Sirin, E., and Kalyanpur, A. (2006), "Modularity and Web Ontologies," *Proc. of KR.*
[11] Day-Richter, J. (2006), "The OBO Flat File Format Specification, version 1.2," *The Gene Ontology*, www.geneontology.org/GO.format.obo-1_2.shtml.