Using a Class Algebra Ontology To Define Conversions between OWL/SQL/Java Beans

Daniel J. Buehrer and Wang Chun-Yao
Institute of Computer Science and Information Engineering
National Chung Cheng University
MinHseng Chiayi 621, Taiwan
dan@cs.ccu.edu.tw and cywang@mail.nhu.edu.tw

Abstract

This paper describes the xml definition of class algebra which is available at http://xbean.cs.ccu.edu.tw/~dan/classAlgebra.xml . A sample user-defined ontology document and instance document are available at http://xbean.cs.ccu.edu.tw/~dan/userOntology.xml and http://xbean.cs.ccu.edu.tw/~dan/userInstances.xml . The class algebra ontology is very similar to the OWL ontology, but it uses its own definition of pointers for non-partOf relations. This simplifies the underlying theory as well as the syntax. The same syntax is used to define the ontology (i.e. the schema) as well as instance documents. The class algebra ontology has sufficient information to enable conversion between class algebra instance documents, SQL tables, and Java persistent objects, all of which can be queried by class algebra queries and updated by class algebra assignment operators, RMI calls, or SOAP method calls. The state-space graph of all possible orderings of class algebra operators is searchable by efficient constraint-based search techniques. Operators include guarded class algebra assignments as well as traditional SOAP or RMI method calls.

1. Introduction

Implementations of the Ontology Web Language (OWL) are now starting to appear. There are several deficiencies of OWL that are evident in these implementations. One major detraction is the use of "ENTITY" definitions within rdf:resource pointers, rather than being able to use name space prefixes directly in the pointers. Another problem is the rather loose nesting of tags, with no clear definition of how objects should be nested within relations. Another problem is an unclear definition of how OWL instance documents can be translated to/from SQL and Java Beans. Finally, the OWL ontology definition should be readable as an OWL instance document, so that meta-objects and meta-relations can be queried using the same query mechanisms as for user-defined data. User-defined ontologies are defined as subclasses of the class algebra ontology, so they can make use of the standard "hasNameSpaces", "hasClassDefns", "hasAttributeDefns", "hasRelationDefns", and "hasInstanceFiles" relations from the ontology to its definitions.

2. PartOf Relations vs. Non-PartOf Relations

Class algebra distinguishes between two kinds of binary relations. PartOf relations are aggregate relations between a whole and its parts. The parts may either be attributes or relations to objects whose existence depends on the existence of the containing object. That is, the inverse "partOf" relation for these relations is a function, with the single surrounding object as its value. Each object must have a unique rdf:ID. The objects in a relation may either be accessed directly via the rdf:ID or indirectly via an iterator for the surrounding relation.

All relations have a value which is a List (i.e. all relation values are associative, "flattened" unions (i.e. appends) of values). There are various subclasses of relation values defined as follows:

Functional	Single-valued (cardinalty=1)
Set	No repetitions; idempotent: (x&x=x)
SortedSet(attrs)	x.attrs<y.attrs → before(x,y)
UndirectedGraph	symmetric: (reln(x,y)→reln(y,x))
ReducedGraph	transitive: (reln(x,y)&reln(y,z)→reln(x,z)
DAG	transitive, antisymmetric: reln(x,y)→~reln(y,x)
Tree	functional(inv_reln); e.g. "partOf" relation
ReachableSet	reln(x,y) & relnClosure(y,z) → relnClosure(x,z) reln(x,y)→relnClosure(x,y)
Leaves	relnClosure(x,y) & ~(∃ z) reln(y,z) → leaves(x.y)

A class algebra instance document consists of nested tags, alternating between class names (capitalized, as in Java), and relation names (starting with a lower-case letter, as for Java attributes or collection names). All tags act as constructors, constructing either a new object or new partOf 1-m relationship edges.

Every object has an "absolute" name which starts at a given name space prefix and lists all dotted partOf relations, with non-functional relations followed by a square-bracketed rdf:ID label or subscript. The subscript depends on the physical order of the list of parts. For instance, the name "hasObjects[Tom].friends[Jim].child[2].haircolor" refers to the hair color of one particular person, the second child of Tom's friend Jim.

The "short" name of an object is simply of the form "prefix:@ClassName[rdfIDValue]", where ClassName must be a subclass of the declared range of the relation in which it occurs. "@ClassName" is actually shorthand notation for "hasClassDefns[ClassName].extent", the extent (i.e. list of instances) of the class definition.

Both absolute and short names of objects are meaningful to both people and computers, and any class algebra object on the network has a unique absolute or short name after the prefix is expanded into a URL. The name space associated with the prefix actually may contain many URL's for SQL databases, XML files, or Java serialized files, but only one of these URL's is the writable URL, and the others are read-only copies.

3. OidLists

The relation domains and ranges are defined in the class algebra ontology. However, relationships may relate any objects in subclasses of those domains and ranges, and these subclasses may be stored on different machines in different formats. (All class instances are restricted to be in the same instance file). The relation values are always oidLists, regardless of whether the relation is stored in class algebra XML files, serialized persistent Java beans, or SQL tables. The oidList is a string composed of oidElements separated by semicolons. Each oidElement has the form prefix:@Classname[rdfIDList]. As mentioned, the prefix is associated with a unique "writable" file and "readable" copies in SQL databases, class algebra XML files, and/or Java serialized bean files. The Classname is a subclass of both the previous relation's range class and the next relation's domain class. The rdfIDList is a list of rdf:ID values or ranges, where the ranges depend on the physical order of the objects in the XML file, SQL database, or Java collection. The rdf:ID values must be unique within the name space of the prefix, so the Classname is theoretically redundant, but is supplied for easy translation to SQL tables, where each Classname is a SQL table name. When SQL tables are translated to class algebra, the primary keys are made unique by adding the table name, if necessary.

4. Class Algebra Queries

Basically, class algebra queries look like extended OidLists. The OidLists are extended with selections, within curly brackets. These selections are based on Boolean combinations of "in" containment predicates which correspond to subclass containments or primitive value lattice or set containments.

Class algebra queries can also contain a "…" operator instead of a single dot operator.

This indicates that the following query can be nested anywhere within the partOf relations of the current object, similar to XPath's use of the // operator. Also, a "*" can be used as a wildcard that can match any relation name.

Class algebra queries can also contain groupBy objects and their aggregate functions, cnt_, sum_, avg_, min_, max_, and std_. The attribute names of a group are created by appending the above aggregate name onto the other numeric attribute names (i.e. those not used in the groupBy) of the original objects. The other non-numeric attributes and relations only have a corresponding cnt_attrOrRelnName attribute in the corresponding group. For example, let the original objects are in a class with the following attributes:

attr1:int attr2:float attr3:date attr4:Integer attr5:Url reln1:Class1 reln2:C2

Then, the groupBy("attr1,reln1") query would produce the following attributes:

attr1:int reln1:Class1 cnt_attr2 sum_attr2 avg_attr2 min_attr2 max_attr2 std_attr2 cnt_attr3 cnt_attr4 sum_attr4 avg_attr4 min_attr4 max_attr4 std_attr4 cnt_attr5 cnt_reln2

5. Class Algebra Guarded Assignments

The selections in the above queries can be used to select which assignments are to be performed. Assignments have the following form:

<class algebra query>.relnOrAttrName <assignOp> <class algebra query>

The assignment operators are ":=", "+=", "-=", "@=", "@+=", and "@-=".

For example, the assignment

@People[Dan].friend += @People[George].friend{age in [21..inf]}

will explicitly add George's adult friends to Dan's friends.

The assignments

@Person{sex in [Female]}.husband @= this.spouse

@Person{sex in [Male]}.wife@= this.spouse

define implicit husband and wife relations. When the spouse changes, the corresponding husband or wife also change.

Guarded assignments are useful for writing NP-complete queries, which are not writable as simple class algebra queries. For example, a relaxation algorithm for the graph coloring problem could be stated as follows:

@Node[1].colors := @Colors[Red];

@System.backtrack:= @Boolean[false];

@Node.colors -= neighbor{cnt(colors) in [1]}.colors;

@System{cnt(@Node{cnt(neighbors)>cnt(colors)}>0)}.backtrack:=
@Boolean[true];

//Backtrack if all nodes have at least as many colors as neighbors

@Node.chooseOne{@System.backtrack&cnt(colors)=min(cnt(@Node.colors))}.colors:=
this.colors.chooseOne $\rightarrow$
@System.backtrack:= Boolean[false];

6. Method calls

OWL and class algebra without assignments basically describe data structures. However, OWL's n-triples can describe NP-complete labeling problems, so some rather sophisticated control must be used to answer general OWL queries. Class algebra moves that kind of control into the state-space search.

Traditional object-oriented method calls are generally of the form of state-space operators. They take the input state of "this" and transform it into a new state. The other arguments to the method call must evaluate to read-only constants. That is, the method call with evaluated arguments can be used as the name of an edge in the state-space search graph. The search algorithm should prevent the addition of states that are subsumed (equal to or having more values than another state). The state-space graph will then contain a superset of the states reachable by communicating agents. It simply assumes that any method can be called at any time with any arguments. Actual execution of the agents will only produce a subset of these states, depending on the control structures within the agents.

7. Summary and Conclusions

The tree structure of XML documents and the RDF schema definitions unnecessarily complicate the Semantic Web. OWL is going in the right direction, defining everything in terms of triples, but its syntax is still awkward in several ways. This paper has shown how some of these drawbacks of OWL can be overcome by class algebra XML documents. These documents can also be translated to/from SQL databases and Java serialized Bean files. The class algebra ontology takes the place of XML or RDF schemas by assuming that the Class names and relation/attribute names will become tag names, and that relation/attribute tags must strictly alternate with Class tags. Every class tag must also have an rdf:ID which is unique in the name space. This format is used for both the ontology definitions and the instance objects.

References

Web Services and Service-Oriented Architectureshttp://www.service-architecture.com/index.html

Java 2 Platform, Enterprise Edition http://java.sun.com/j2ee/

W3C Semantic Web Activity http://www.w3.org/2001/sw/news

W3C XML Extensible Markup Language http://www.w3.org/XML/

W3C RDF Resource Description Framework http://www.w3.org/RDF/

W3C OWL Web Ontology Language Overview http://www.w3.org/TR/2003/CR-owl-features-20030818/

M. Ahamad and M. Chelliah, “Flexible Robust Programming in Distributed Object Systems,” IEEE Transactions on Knowledge and Data Engineering, Vol. 14, No. 5, Sept/Oct 2002.

D. Beneventano, S. Bergamaschi, C. Sartori, “Description Logics for Semantic Query Optimization in Object-Oriented Database Systems,” ACM Transactions on Database Systems, Vol. 28, Issue 1, March, 2003.

D. J. Buehrer, Lo Tse-Wen, Hsieh Chih-Ming, Maxwell Hou, "The Containment Problem for Fuzzy Class Algebra," Intelligent Engineering Systems through Artificial Neural Networks, Volume 11, C. H. Dagli et al. (eds.), ASME Press, New York, 2001, pp.279-284. .

D. J. Buehrer and Lei-Ren Chien, "Knowledge Creation Using Class Algebra," 2003 Int’l. Conference on Natural Language Processing and Knowledge Engineering, Beijing, China, Oct. 26-29, pp.108-113.

D. Buehrer and Lei-Ren Chien, Reasoning with Class Algebra", Proceedings of IASTED International Conference on Neural Networks and Computational Intelligence (accepted), Grindewald, Switzerland, Feb. 23-25, 2004.