Organizing Multimedia Objects by Using Class Algebra

Daniel J. Buehrer
National Chung Cheng University, Taiwan, R.O.C.

WEB-BASED DEVELOPMENT

Web-based applications (Web services and service-oriented architectures) can be run via a Web-based browser. There are several approaches to writing such Web-based applications. A lightweight approach is suitable for handheld devices. In this approach, a Java servlet or a JSP page (Java 2 Platform, JSP), or an ASP application (Microsoft .NET, ASP) generates HTML (Hypertext Markup Language), XHTML, or XML documents (W3C Semantic Web Activity, XHTML, XML) to be displayed by the browser. Most browsers use an anchored URLs extension (e.g., .doc, .jpg, .xml, etc.) to choose an appropriate plug-in to display the URL when it is clicked. Besides displaying text and multimedia, Web servers and/or browsers can also execute Java applets or scripting languages to read and/or change persistent data. Previously, about 98% of these data were stored in relational or object-relational databases. However, recently more of these data are being stored in XML-based documents. Often these documents have an associated "schema" declaring the nesting of tags and the types of primitive values, or an "ontology" (Everett et al., 2002, Hunter, 2003) declaring classes, attributes, and relations that are used in the document.

Databases, XML documents, and knowledge bases have traditionally been only a small part of a company's programming environment. The goal of most development environments is to use UML or entity-relation diagrams to analyze and design suitable classes, design patterns, and active objects from other existing ones. Web-based development environments (Stal, 2002) help us to find suitable classes, design patterns, and active objects on the Web, perhaps for a fee, or perhaps for a limited development time. That is, in the future, the development environment itself will be built on a distributed knowledge base.

Will there be a programming language, and will it still have explicit calls to SQL? Currently, scripting languages have their own unique means of accessing SQL databases, and the whole area of Web-based programming and multimedia networking has become quite messy (Weinstein & Gelman, 2003). Although J2EE's Container-Managed Persistency (CMP), JDO, and many object-relational development environments provide ways to hide SQL calls, security, and transactions, many applications may still choose to use the richer functionality of direct JDBC.

Web-based development environments, however, are forcing some changes in the way that persistent data are stored. Heterogeneous databases can be said to have failed to provide a sufficient platform for sharing data because of the difficulty in integrating so many SQL meta-data definitions. There was no "class hierarchy" to organize the definitions of the SQL tables, and small incompatibilities in SQL definitions often caused major headaches. Although ODBC and JDBC are now SQL standards, they permit various implementations of the meta-data, and they provide no standard way of "including" meta-data definitions from other "superclass" tables. In order to counter this lack of a class hierarchy in object-relational databases, the Semantic Web Project is proposing to use XML-based documents to share data and their declarations.

How will the Web-based development environments and their applications access Web databases? Will they access SQL, XML, or both? The answer is almost certain to be both, as well as other spreadsheets, natural language documents, and multimedia.

ISSUES IN INTEGRATING XML AND RELATIONAL DATABASES

As long as the application-tier, Web-tier or client-tier programs access the data via a standard interface, it makes little difference to the programmer if the persistent data are stored in SQL databases or XML files. This interface should take care of transactions (Ahamad & Chelliah, 2002) and security, sequencing set/get/add/remove calls from multiple users and doing load balancing and fail-over across multiple servers.

J2EE from the Java Community Process and Microsoft's .NET have the two major candidates for this programmer interface. One goal of these platforms is to hide the SQL and XML calls behind standardized set/get/add/remove calls for enterprise components. However,

SQL programming and XML programming are currently fairly incompatible. There are several XML query languages for querying XML documents, and XSLT can be used to reformat XML documents. There are numerous XML schema mechanisms to enforce some type-checking, but higher-level schema mechanisms, such as RDF-Schema and OWL Ontologies (W3C RDF, OWL), are really needed before attribute and relation assignments can be strongly type-checked. OWL declarations essentially take the place of Corba IDL declarations, OMG ODL declarations, or SQL meta-data tables, by declaring ontology namespaces, a class IS-A hierarchy, attribute types and relation domains/ranges.

CLASS ALGEBRA'S ROLE

Class algebra (Buehrer, 2001; Buehrer & Chien, 2003; Buehrer & Lee, 1999; Buehrer, Tse-Wen & Chih-Ming, 2001; Buehrer, Tse-Wen, Chih-Ming & Hou, 2001; Buehrer & Wang, 2003) can serve as the theoretical basis for type-checking both OWL-based and SQL-based set/get/add/remove calls. Each SQL table is considered to be a class, and each line of the table is an object. The primary key of the object is used, along with the JNDI database service name and the table name, as a unique object identifier on the network. Each join of a foreign key to the associated primary key corresponds to a binary relation in class algebra.

For example, suppose that the user wants to find all images of Tom and his dog. One could locate Tom either by using the query @People{firstname= "Tom"} to find all people named Tom, or by following relations from his school, friends, or other objects to locate Tom. Then his pet dog(s) can be found by following the "hasPet" relation and limiting the answer to dogs. The intersection of the images containing Tom and the images containing his dog will be the images containing both Tom and his dog, as given by the query "query1@= @People{firstname= "Tom"}; query2@=query1.inImages @* (query1.hasPet @* @Dog) . inImages", where "inImages" is the relation from objects to the images that contain them and @* is the class intersection operator. These class algebra queries can be translated into SQL or XQuery, depending on where the data are stored. An interactive environment allows the user to search the IS-A hierarchy for a particular class, and then recursively constrain some of the attribute or relation ranges. The user can then combine various subqueries with Boolean operations, as given previously. The major advantage of class algebra queries is that the range is known. For example, query1 is People and query2 is Images. Therefore, all subclasses and their attributes and relations can be displayed when creating further queries. Moreover, a query may make use of the containment algorithm to optimize (Beneventano, Bergamaschi & Sartori, 2003) the processing of the query.

There are many details of the definition of class algebra that may be incompatible with various programming language typing constraints. For instance, in order to satisfy the laws of Boolean algebra, class algebra permits multiple inheritance, unlike Java and C#. Class algebra has tried to follow the Ontology Web Language, OWL, as much as possible. For example, attributes, relations, and methods are defined relative to an ontology name space rather than relative to a class. An object (or class) is permitted to be equivalent to other objects (or classes) that have different object (class) identifiers. For instance, anything known about "007" can be added to anything known about "James Bond," since they are names of the same concept (i.e., class). This facilitates sharing of equivalent objects and class definitions among organizations. The normalization process will replace such object (class) identifiers by a unique representative, and attribute and relation values will be merged together for this canonical oid (cid).

In relational algebra, the result of any operation is a relation. In class algebra, the result of any operation is a class. A class's "extent" is a DBTable, which is a collection of DBObjects. These are different from transient programming language objects, which can only be made persistent by serializing them as the value of an attribute of a DBObject. DBTables can have listeners that are guaranteed to hear updates to their contained DBObjects. The calls to get/set/add/remove DBObjects are implicitly queued as necessary to preserve ACID (atomicity, consistency, isolation, durability) properties of transactions.

A DBTable has methods for sorting, grouping, histograms, and report generation. It also has a "select" operation, which can choose a subset of objects that satisfy a given class algebra constraint, represented as a string (unlike J2EE "find" methods, which must be compiled). The constraint is first normalized to a "simplest" form. The simplest form is guaranteed to be computable in $O(n^3)$ time since it involves a simple sort of a transitive closure of Horn clauses that contain no explicit variables. A Prolog definition of the normalization process is available at *http://www.cs.ccu.edu.tw/~dan/fuzzyProlog.txt*.

As well as using the normalization procedure to optimize queries, the class "intents" (i.e., normalized constraints) are used to organize the definitions of classes into the class hierarchy. That is, the normalized class definitions are used to recognize when one class is a superclass of (i.e., contains) another, or when two class definitions are equivalent or have empty intersections.