An Approach to Object-Relational Mapping in Bioscience Domains

David Tuck, MD*, Ryan O’Connell, MD*, Peter Gershkovich, MD*, James Cowan**
Center for Medical Informatics*
Department of Pathology**
Yale University
New Haven, Connecticut

Abstract

Object-relational mapping (mapping object-oriented software systems to relational databases) remains a challenging undertaking because of the inherent differences between these two technologies. Effective application of this technique in biomedical informatics applications can provide robust, adaptive solutions but requires careful attention to specific issues in the bioscience domain. Mapping from a single, fixed, stably defined object to a table is straightforward, and several standard patterns have been described for mapping inheritance trees to tables. However, mapping becomes more problematic when it involves complex, evolving inheritance trees, sparse, dynamic attributes, or associations and aggregations whose requirements and definitions change frequently during development. All of these seem to be inherent, inescapable facts in the development of biological information systems.

Here we describe modifications of the basic patterns of object-relational mapping that are being applied to two systems in ongoing collaborations between biomedical investigators and informaticians. In this approach a small set of objects for each entity collaborates to provide the usual required services of an object-relational implementation including query capacity, transaction management, object relationships (associations, aggregation, composition, many-many relations) and persistence functions. This set of objects also allows dynamic attributes for a selected set of entities by merging an entity-attribute-value approach for data storage with the object-relational approach to managing relationships among objects. Templates are used for class generation to allow flexibility and synchronization of schema/object during design evolution, facilitating changing persistence strategies across the whole system for testing and development. The entity-attribute-value details as well as all database access are encapsulated in a persistence layer which would allow change of this layer to a conventional storage model, if the design evolution is stabilized or for performance requirements, with no change to the domain or presentation layers.

Background

Considerable thought, effort, and collaboration devoted to biological information systems from numerous groups have made an extensive body of data management tools and software, including schemas, models, and complete implementations, available through the internet, biomedical literature, as well as commercial sources [1,e.g]. However, a number of factors continue to demand that software solutions for many applications, especially at academic centers, be locally developed: specific interests and local requirements of investigators, locally available expertise in database or development platforms, the extensive resources or cost which may be necessary for conversion, training, or adaptation of external tools, and the instability or lack of consensus on standards and vocabularies.

The development of computer systems through collaboration between biological investigators and biomedical informaticians is challenging for a number of reasons including:

- the explosion of data, both investigator-generated experimental results and annotation or comparative data available from external sources;
- the heterogeneity of this data;
- competing, incomplete, rapidly evolving or completely absent standards and vocabularies;
- integration of disparate languages and cultures within biological sciences and medicine; the necessity for teamwork among numerous individuals with varying backgrounds and expertise;
and in particular the rapid change in the requirements of investigators, and their understanding of the possibilities available for managing and analyzing and interpreting the data they generate.

The benefits and increasing use of object-oriented systems [2] and the dominance of relational databases as the method of preference for storing data have led to the use of these technologies together, although considerable challenges are involved in their optimal combination. Three fundamental strategies for implementing a mapping of class inheritance structures to a relational database have been described in detail [3-7]. A variety of other options may be appropriate in some situations including serialization of objects [7] where querying based on object subcomponents is not critical. Each has advantages and disadvantages and must be selected based on features of different applications. Briefly, they are:

1) An entire class hierarchy is mapped into one table, where all the attributes of all the classes in the hierarchy are stored. The advantages of this approach are simplicity and ease of ad hoc reporting. But, space is wasted, storing attributes which are not relevant to certain subclasses; adding a new attribute to a table is required whenever any subclass has a new attribute; and, coupling is increased within the class hierarchy making change more cumbersome as the system becomes more complex.

2) One table per concrete class: each table includes both the specific attributes of a class and any attributes it inherits. This also provides efficient ad hoc reporting - all the attributes for any single class are stored in one table, but multiple tables must be modified for any change in a class higher in the hierarchy; entire objects must be copied from one table to another if the role changes within the hierarchy; and, maintaining data integrity is difficult when objects have multiple roles.

3) One table per class including abstract subclasses: this conforms best to object-oriented concepts: it supports polymorphism, and each attribute in the class inheritance tree is represented exactly once in a table. This overcomes the problem of sparse attributes, as only the appropriate classes have the required attributes, and it only requires modification of subclasses when an attribute changes. New subclasses only require adding a single new table with the appropriate attributes, with no change to other tables. However, this approach may markedly increase the number of tables in the database, both for storage and to maintain relationships. It requires increased overhead for accessing multiple tables for retrieval, and ad hoc reporting is more difficult, requiring proliferating views to simulate any desired combination of tables.

The entity-attribute-value (EAV) approach for modeling highly heterogeneous scientific data has been explored in detail recently, including object-oriented innovations [8] and its pros and cons reviewed [9]. A database table employing EAV design, which is also known as "row modeling", stores just a single fact per row rather than a set of facts per row. Data describing each fact is stored in a data library (or lookup table) containing metadata, or information that describes data in the EAV table. By use of a metadata table which describe the schema, EAV format gives the impression that data is organized in a conventional format. Unlike in a purely relational database, in which the physical schema resembles the logical schema, the physical layout of EAV data may look quite different from the relational schema it emulates. [10,11] EAV models have been noted to have advantages over conventional design for heterogeneous data, schemas which are not stable, where attributes are likely to evolve over time.

Choices among the three object-relational mapping patterns above involve some of the same issues as the EAV approach: efficient storage of sparse attributes, heterogeneity, flexibility in addition of attributes for an entity, and efficiency of object based queries [8]. None individually addresses all of these optimally. A hybrid approach, sharing a common underlying persistence strategy, in which an EAV structure is used for specific entities, allowing dynamic management of attributes, while other entities and relations between entities are handled by a conventional object-oriented system may have benefits over any of these fundamental object-relational mapping patterns individually.

Problem Description

A software engineering model and tools are described which are being explored for information systems for two projects at our institution involving either high throughput gene
expression microarray or tissue microarray data. Both share certain features: dynamic change of requirements is expected to be constant, with functionality expected to change every few months, if not more frequently. Standards are evolving; external references databases are evolving; reference ontologies are in development; the needs, interests, and priorities of investigators are constantly changing, often rapidly and dramatically as new information and tools become available. Development involves informal interactions among geneticists, molecular biologists, experimentalists, computer scientists, statisticians, clinicians, and informaticians, where complete upfront design with stable requirements and schemas is not a realistic possibility.

None of the above object-relations patterns by itself seemed optimally matched to the software development requirements. Although the third solution might be the best, it would require frequent modifications of the codebase and schema. Therefore, a hybrid approach was developed based primarily on the first pattern with the addition of the capacity for storing data in an EAV structure for selected entities where the greatest degree of complexity, instability, and demand for change during the course of development, were expected.

For the investigators involved with gene expression data, there were some general requirements that in combination were not entirely met by available software options. Many of these focused on the requirements for annotation, (either of genes or experiments) and the need for continually updating annotation data from heterogeneous external sources or adding new sources. Some specific needs include:

-creation or modification of gene lists based on either some objective functional, structural, or mechanistic criteria derived from external annotations, or on a investigator-defined criteria such as particular interests of their laboratory or results in other previous or ongoing experiments possibly from different techniques (DNA microarray, mass spectrometry, 2D-gel);

-annotation criteria for querying or categorizing genes needed to be based on heterogeneous sources with up to date information from various reliable external sources;

-aggregation of reporters for a single gene, based on these different annotations, to be reviewed manually by an investigator for consistency, outliers, exceptions, etc, across different experiments, technologies, or research methods;

-updating annotation data from publicly available reference data sources as they change;

-addition of new types of annotation as they become available.

Analogous considerations existed for the investigators involved in high throughput tissue array studies, though the areas of heterogeneity, instability and complexity suggested a benefit by the use of an EAV approach involving tissue description and clinical factors, rather than annotation.

Elements of the Solution

Both systems share the object-relational model and persistence mechanisms. Both use a multi-layered approach sharing the core functions of the persistence layer (connection management, object creation, retrieval, update, deletion functions), though with completely different domain layers, and with one using a web-based and the other using a client-based presentation layer (see figure 1).

Several types of object collaborate to maintain the persistence layer – a ‘Home’ object which will maintain the relationship of an object to a particular table and persistence support mechanism (in our implementation in the Java language using JDBC); a ‘Facade’ object [12,13], which uses minimal resources to provide a pointer to an entity’s data, but allows access to attributes of that entity on demand. This façade controls the strategy for retrieving different attributes or associated objects belonging to or related to that object; an ‘Entity’ object which is a more resource-intensive object managing all the data related to the object; and finally a ‘Support’ object which actually handles all database interactions.
The object-relational mapping approaches address some of the same problems, and therefore, a hybrid approach in which a simplified EAV structure allowing dynamic attributes of individual entities, while maintaining other aspects of relations between objects to be handled by a conventional object-oriented system – is described. In this way, associations, aggregations, composition, including many-to-many relations are maintained.

Dynamic attributes and their management. This allows the addition of new links by adding metadata; additional functionality such as adding new external databases; Adding new links to external databases such as Pfam, SMART, Gene Ontology, can be added simply by adding metadata describing the links – addition of further databases may be added by pointers into that database which then may require additional classes or modules but can be effectively decoupled from the domain layer itself.

Our Implementation Choices:
1) Object-relational mapping extended to include dynamic attributes; modification of straightforward entity-attribute-value structure for storage.

2) Template-based generation of elements of the object-relational map to provide easy regeneration of objects and rapid changes to – based on the Jakarta Velocity project [14].

3) Multi-layer pattern – persistence layer, domain layer, presentation layer

4) Presentation layer – uses model-view-controller architecture (MVC) based technologies – Struts for web interface [15], Swing for client based application.
5) Sharing elements of the persistence layer across different applications (Tissue microarray, DNA microarray).

7) External database annotations mapped to entity attributes with incorporation of various elements of external databases, balanced between need to query versus need to link out;

8) Semi-automated periodic update of external database references

9) Java programming language, as cross-OS solution required (Macintos, Linux, Windows users; existing Linux, Windows servers).

Conclusions

Software development for biological information systems remains a difficult challenge. Generic complete solutions are not currently available. Software must be resilient to change, even dramatic change, as investigators become aware of new information which may suddenly alter their focus and interest; and to accommodate expected ongoing design changes.

Information tool development itself can be seen as an exploratory, experimental process in which certain explorations yield no benefit, and other paths which appear promising should be rapidly followed up on. The expectation of change, and facility of change must be designed into such systems from the beginning.

The hybrid system described here uses standard object-relational mapping techniques where possible, in stable portions of the overall model, and isolates the advantages of an EAV model to areas of the greatest heterogeneity, unpredictability, and instability, where attributes can’t be known in advance. For our current applications, these areas include the evolving variety of methods for annotation of genes or experiments, description of tissue, targets, and clinical factors. This approach coupled with a template system facilitates re-use, modification, testing, and experimentation with the classes and strategies implementing both mechanisms.

References:

1. Microarray Express, European Bioinformatics Institute, from URL: http://www.ebi.ac.uk/microarray/ArrayExpress/


