Constructing and Maintaining Scientific Database Views in the Framework of the Object-Protocol Model *

I-Min A. Chen, Anthony S. Kosky, Victor M. Markowitz and Ernest Szeto
Information and Computing Sciences Division
Lawrence Berkeley National Laboratory, Berkeley, CA 94720
{IAChen, Anthony_Kosky, VMMarkowitz, E_Szeto}@lbl.gov

Abstract

Scientific databases (ScDBs) are used to archive and retrieve data describing objects of scientific inquiry. Since these ScDBs must provide continuous and efficient access to large communities of scientists, they are often developed with reliable commercial relational database management systems (DBMSs) or file systems. However, relational DBMSs and flat files do not provide constructs for representing directly ScDB-specific objects and experimental procedures, and therefore they are often hard to develop, maintain, and explore.

In this paper, we present a retrofitting tool for constructing and maintaining ScDB views using an object-oriented data model, and describe our experience with retrofitting ScDBs that have been originally developed using relational DBMSs and file systems.

The retrofitting tool is part of a data management toolkit based on the Object-Protocol Model (OPM). The OPM toolkit provides facilities for developing databases defined using OPM and for querying and browsing such ScDBs in terms of OPM constructs. The OPM retrofitting tool allows constructing (one or several) OPM views for ScDBs that have not been originally developed with the OPM tools. ScDBs with native OPM schemas or retrofitted OPM views can be browsed and queried via OPM interfaces, reorganized, or incorporated into an OPM-based database federation.

1. Introduction

Numerous scientific databases (ScDBs) are used to archive and retrieve data describing scientific objects, such as proteins, DNA sequences, crystallographic structures, and scientific experiments. Some archival ScDBs are maintained as flat files, using various notations, possibly augmented with some indexing methods. For example NCBI's DNA sequence database, Genbank, is represented using the ASN.1 notation [21]. Such systems do not provide support for ad hoc queries or flexible data browsing and exploration. Other ScDBs are developed using commercial relational database management systems (DBMSs). For example the Genome Sequence Database (GSDB) at the National Center for Genome Resources, Sante Fe, is implemented using Sybase. Since relational constructs cannot be used for directly representing scientific objects and experimental procedures, these objects and procedures are usually represented in a relational ScDB by disconnected tuples that are scattered among multiple tables. The complexity of such representations makes the development and maintenance of large relational ScDBs error-prone and time-consuming processes. Object-oriented and object-relational DBMSs address some of these problems, but are often less robust and/or provide more limited query facilities than relational databases. An account of some of the problems encountered when developing an ScDB using such an object-oriented DBMS may be found in [14]. Consequently numerous ScDBs continue to be developed using commercial relational DBMSs.

An alternative approach is to implement object wrappers for files and relational databases, allowing users and applications to interact with ScDBs via object-oriented interfaces, while insulating them from the underlying systems. This approach underlies the OPM data management tools which allow specifying, querying, and manipulating ScDBs using an object-oriented data model, the *Object-Protocol Model* (OPM) [9], while implementing the ScDBs using commercial relational DBMSs. OPM has similarities with other object-oriented and semantic data models (e.g., [2, 16]) in supporting classes, inheritance, and (regular and derived) attributes, and in addition has constructs specifically designed for modeling scientific data

*This work is supported by the Office of Health and Environmental Research Program and the Mathematical, Information, and Computational Sciences Division of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

and experiments. The OPM data management tools provide facilities for (1) defining and modifying OPM schemas; (2) automatically generating relational database definitions and procedures from OPM schemas, including procedures for maintaining constraints (e.g., referential integrity constraints) and for implementing OPM retrieval and update methods; (3) browsing, data entry, and querying databases in terms of OPM constructs. The OPM tools have been used to develop several ScDBs, including the Genome Database (GDB) at Johns Hopkins University School of Medicine,[1] the Primary Database of the German Human Genome Research Center in Berlin,[2] the Electronic Notebook for the Spectro Microscopy Collaboratory at the Advanced Light Source Beamline 7 at the Lawrence Berkeley National Lab,[3] and the Event/STAR database at the Relativistic Heavy Ion Collider at the Brookhaven National Lab.[4] Further details regarding these tools, including examples of ScDBs developed using OPM, can be found at http://gizmo.lbl.gov/opm.html.

In this paper, we describe a *retrofitting* tool for constructing and maintaining OPM views on top of existing ScDBs, and discuss our experience with applying this tool to several large production ScDBs. For ScDBs developed with the OPM data management tools, the OPM retrofitting tool allows maintaining multiple ScDB views. For ScDBs that have not been developed with the OPM tools, the OPM retrofitting tool provides these ScDBs with semantically enhanced OPM views, and facilitates database reorganization. Retrofitted ScDBs can then be browsed and queried individually via their OPM views or can be assembled into a *multidatabase system* that can be explored using the OPM multidatabase tools.

Constructing OPM views for ScDBs is closely related to work on reverse engineering relational database schemas into schemas based on semantic or object-oriented data models (among others, [13, 17, 19]). Most reverse engineering techniques [13, 19] are based on versions of the Entity-Relationship (ER) Model [11] and follow an algorithmic approach where object structures are automatically inferred by analyzing relational schema patterns. These approaches are limited in their ability to distinguish tables representing classes of objects from tables representing complex attributes or relationships between objects, and rely on information, such as foreign key references, that are often missing from database definitions. Furthermore, an automatically generated object view does not necessarily match the users' perception of the underlying database.

The OPM retrofitting tool follows an *iterative* strategy of constructing OPM views for flat files or relational ScDBs, similar to that proposed in [22]. First, a canonical (default) OPM view is generated automatically from the underlying ScDB schema. Then this canonical OPM view can be refined using schema restructuring operations, such as renaming and/or removing classes and attributes, merging and splitting classes, adding or removing subclass relationships, defining derived classes and attributes, and so on. A *mapping dictionary* records information regarding the relationships between the view (OPM) constructs and their corresponding representations in the underlying database. This mapping dictionary is used to generate appropriate retrieval methods for the view attributes and classes, which are used by the OPM querying and browsing tools.

In contrast to most existing reverse engineering techniques, the retrofitting tool presented in this paper:

1. is based on a rich data model that includes constructs such as tuple attributes, set and list-valued attributes, union value classes, derived attributes and classes;
2. allows coping with poorly designed (e.g., incomplete) database schemas;
3. supports multiple customized views for the same underlying database;
4. provides the necessary infrastructure for browsing and querying the underlying database via the retrofitted views, as well as for reorganizing databases and constructing multidatabase systems.

The rest of the paper is organized as follows. In section 2 we briefly overview the Object-Protocol Model. The automatic construction of canonical OPM views and operations for restructuring OPM views are presented in section 3. Our experience with retrofitting ScDBs is discussed in section 4. Applications involving retrofitted ScDBs are briefly described in section 5. Section 6 contains concluding remarks.

2. The Object Protocol Model

The *Object-Protocol Model* (OPM) is the result of incorporating constructs for modeling scientific experiments (*protocols*) into an object data model [8, 9]. In this paper we refer only to the *object* part of OPM.

The object part of OPM is similar to other semantic [15, 16] and object data models [2]. Objects in OPM are uniquely identified by object identifiers (oids), and are classified into *classes*. Each class has a finite set of *attributes* associated with it. A subset of the attributes associated with an object class is specified as the external identifier (`ID`) for the objects in the class. A class can be defined as a *subclass* of other (super) classes. Each class *inherits* the attributes of all its (direct and transitive) superclasses.

[1] http://gdbgeneral.gdb.org/gdb/
[2] http://www.rzpd.de/
[3] http://www-itg.lbl.gov/~ssachs/notebook/project.html
[4] http://gizmo.lbl.gov/jopmDemo/star.html

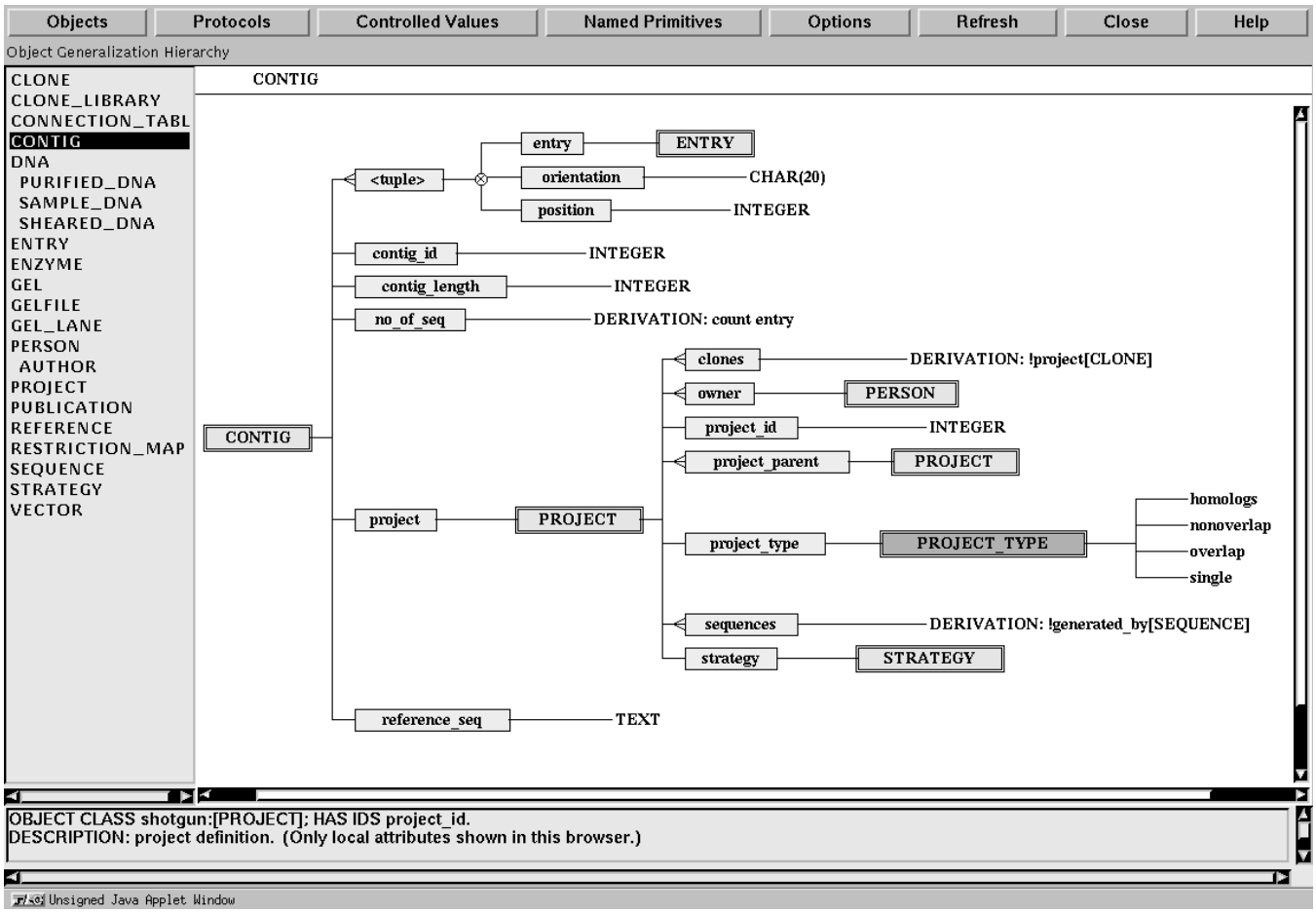

Figure 1. Diagrammatic Representation of Object Classes

An attribute can be *single-valued*, *set-valued* or *list-valued*, and can be required to have *non-null* values. In addition, an attribute can be *simple*, that is, take values from a single value class or union of value classes, or can consist of a *tuple* of simple attributes.

Figure 1 shows part of an OPM schema for a molecular biology application modeling *contig maps*, viewed using our Web-based OPM schema browser.[5] A contig map is used to represent the relative positions within a chromosome of known sequences or *fragments* of DNA, in order to determine the sequence of larger contiguous regions of DNA. Since existing experimental techniques only allow for the sequencing of DNA fragments of a few hundred nucleotides in length, combining overlapping fragments allows biologists to determine the sequences of larger regions of DNA, and eventually of the entire genome of an organism. Each contig map has properties such as `contig_id` representing the unique contig map identifier; `project` representing the project owning the contig map; `(entry, orientation, position)` representing the fragments with their positions in the contig map.

The OPM classes shown in Figure 1 are `CONTIG`, `ENTRY`, `PROJECT`, `PERSON` and `STRATEGY` (attributes of `ENTRY`, `PERSON` and `STRATEGY` are hidden). In this example, attributes `clones` and `owner` of `PROJECT` are set-valued, while attributes `contig_length` and `project` of `CONTIG` are single valued; simple attributes `entry`, `orientation` and `position` of `CONTIG` comprise a tuple attribute.

If the value class of an attribute is a system-provided data type (e.g., `INTEGER`) or a *controlled* class of enumerated atomic values (e.g., the controlled value class `PROJECT_TYPE` in Figure 1 containing four possible values), then the attribute is said to be *primitive*. If an attribute takes values from an object class or a union of object classes, then it is said to be *abstract*. Set and list-valued attributes can also be associated with *cardinality constraints* that define the minimum and maximum number of values

[5] http://gizmo.lbl.gov/jopmDemo/shotgun.html

the attribute can have for each object.

Derived attributes have values computed from the values of other attributes using arithmetic expressions, aggregate functions (`min`, `max`, `sum`, `avg`, `count`), or attribute composition. A composition derivation consists of a path or a union of paths of the following form: B_1 $[O_{i_1}]$ B_2 $[O_{i_2}]$ ... B_n $[O_{i_n}]$, where each O_{i_k} $(1 \leq k \leq n)$ denotes a class, and each B_k $(1 \leq k \leq n)$ denotes an attribute or *inverse* attribute associated with $O_{i_{(k-1)}}$ $(O_{i_0} = O_i)$. An inverse attribute of a class O, is the reverse of an attribute A associated with another class, O', where O is a value class of A; such an attribute is denoted `!`A. For example, derived attribute `clones` of class `PROJECT` is defined by derivation: `!project[CLONE]`: for an instance x of `PROJECT`, `clones` consists of instances of `CLONE` whose `project` values contain x.

OPM supports two types of derived object classes: subclasses of one or several object classes and superclasses of several object classes (e.g., see [1, 23]). A *derived object subclass*, O_s, is defined as a subclass of one or (intersection of) several object classes, O_1, ..., O_m $(m \geq 1)$, possibly associated with a condition. O_s consists of the subset of objects that belong to the intersection of classes O_i, $1 \leq i \leq m$, and satisfy the associated condition (if any). A *derived object superclass*, O_g, is defined as the superclass of object classes O_1, ..., O_m $(m \geq 2)$, and consists of the union of objects belonging to these classes. For example, class `Long_Contig` can be defined as a derived subclass of class `CONTIG` in Figure 1, consisting of fragments whose `contig_length` is greater than 10,000 base pairs.

3. Constructing Views for Scientific Databases

In this section we describe the procedure for constructing OPM views on top of ScDBs that are defined using relational database or ASN.1 notations. This procedure consists of first generating a canonical OPM view from the ScDB definition, and then refining this view using schema restructuring operations. The information regarding the mapping between OPM constructs and the native ASN.1 types or relational constructs is recorded in a mapping dictionary. Different OPM views can be constructed on top of the same database or file, each with its own mapping dictionary (see Figure 2).

3.1. Constructing Canonical OPM Views

The first stage of retrofitting involves automatically generating a canonical OPM view from the underlying ScDB schema definition. This stage of retrofitting currently targets ScDB schemas specified using relational database or ASN.1 notations, and can be extended to cover additional notations.

3.1.1. Canonical OPM Views for Relational ScDBs

The relational data model provides a very simple construct for representing data structures, the *relation* tabular structure consisting of columns representing atomic (non-decomposable) and single-valued *attributes*. From the large range of constraints provided by the relational data model for expressing inner relation or inter relation data dependencies (e.g., see [18]), we consider only constraints that are supported by commercial relational DBMSs, namely: (i) *keys* which ensure that a tuple in a relation is uniquely determined by some subset of its attributes; (ii) *null constraints* for restricting attributes to non-null values; (iii) *domain constraints* for restricting the range of values of an attribute; and (iv) *referential integrity constraints* for expressing existence dependencies between tuples by associating a *foreign* key in a referencing relation with the *primary* keys of referenced relations [12].

Several relations, together with a number of relational *constraints*, are usually needed for representing a single class of objects: a class with only simple, single-valued attributes can be represented using a single relation; however, representing an object class with set or list-valued attributes may require using additional auxiliary relations. *Object identifiers* (surrogate key values) can be used for associating tuples in different tables that represent data regarding the same conceptual object. In general, an object class can be represented by one *primary* relation, representing the single valued attributes of the class, and an auxiliary relation for each set or list-valued attribute of the class, with object identifiers used as primary keys for the primary table and foreign keys for each auxiliary table [10].

When constructing a canonical OPM view for a relational database we avoid making assumptions about which underlying relations containing data regarding the same class. Consequently each relation is interpreted as representing an individual object class and foreign keys are interpreted as class references:

1. each non-primitive relational data type (so called *user defined* data types) D, is mapped into a controlled value class, if D is associated with a set of values or ranges, or a named primitive value class otherwise;

2. each relation with a primary key, R_i, is mapped into an object class, O_i, where the primary key of R_i is interpreted as representing the object identifier of O_i; every non-foreign key relational attribute A of R_i is mapped into a primitive attribute of O_i, associated with the value class representing the data type of A; every foreign key F_{ij} of R_i, referencing relation R_j, is mapped into an abstract attribute of O_i, whose value class is class O_j representing R_j.

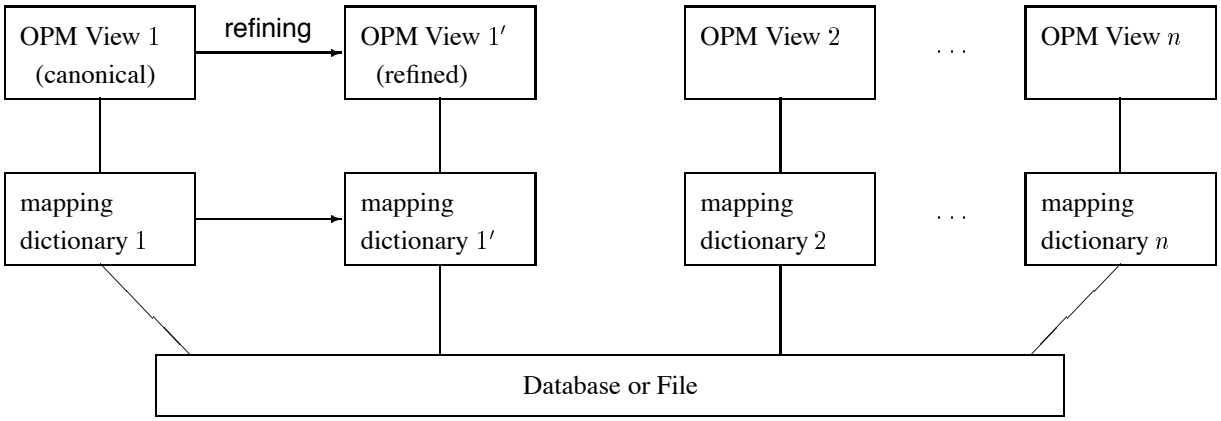

Figure 2. OPM Views on top of an Existing Database or File

3.1.2. Canonical OPM Views for ASN.1 ScDBs

ASN.1 (Abstract Syntax Notation One) is an ISO standard notation that is used for defining scientific (as well as other) data.[6] The National Center for Biotechnology Information (NCBI) is using ASN.1 for encoding the most comprehensive repository of DNA sequence information [20].

Objects can be represented in ASN.1 databases (files) as instances of *structured types*. A structured type consists of fields (attributes), where an attribute can be defined as having (1) a single value of type T_i, (2) a set of instances of type T_i, (3) a list of instances of type T_i, (4) one of several alternative (`choice`) instances of other types, T_i, $1 \leq i \leq n$, (5) an unordered (`set`) or ordered (`sequence`) tuple of component attributes defined as mentioned above, where each T_i can be a *simple* data type, such as (`integer`, `real`, and `boolean`), an *enumerated* type consisting of lists of named integer codes (e.g., male(1)), or a *structured* type. A type can be referenced by name in an attribute definition, or its definition can be embedded (nested) within the attribute definition.

A canonical OPM view is automatically generated from an ASN.1 definition as follows:

1. each *simple* ASN.1 type is mapped into an OPM named primitive value class;
2. each ASN.1 *enumerated* type is mapped into an OPM controlled value class;
3. each ASN.1 *structured* type, T_i, is mapped into an OPM object class, O_i, where each attribute A_j of T_i is mapped into an attribute of O_i as follows:
   (a) if A_j is defined as having values that are single/set-of/list-of values of simple or enumerated type T_k then A_j is mapped into a primitive single/set/list valued attribute of O_i taking values from the class representing T_k;
   (b) if A_j is defined as having values that are single/set-of/list-of instances of structured type T_k where the definition of T_k is not embedded (nested) into the definition of A_j, then A_j is mapped into an abstract single/set/list valued attribute of O_i taking values from the class representing T_k;
   (c) if A_j is defined as having values that are single/set-of/list-of tuples of (simple or structured) types $T_{k_1}, \ldots, T_{k_m}$, where the definitions of $T_{k_1}, \ldots, T_{k_m}$, are not embedded into the definition of A_j, then A_j is mapped into a (primitive or abstract) tuple attribute of O_i consisting of component attributes taking values from the classes representing $T_{k_1}, \ldots T_{k_m}$, respectively;
   (d) if the definition of A_j embeds the definition of a type T_k, then T_k is first mapped into an (auxiliary) OPM class before the steps described above are followed for mapping A_j.

[6] ASN.1 was originally intended for specifying the data that cross the interface between the Application and Presentation layers of the Open System Interconnection (OSI) architecture. For bibliography on ASN.1 as well as applications using ASN.1 see: http://www.inria.fr/rodeo/personnel/hoschka/asn1.html.

3.2. Refining OPM Views

The second stage of retrofitting involves iteratively refining the canonical OPM view using OPM schema restructuring operations. Refining an OPM view only results in changes to the OPM view definition and the corresponding changes in the mapping dictionary, and has no effect on the

underlying (relational or ASN.1) ScDB (see Figure 2). This is very important since often the underlying database cannot be changed, for example, in order to preserve existing applications. The OPM schema restructuring operations have similarities with those discussed in [3], but have a different effect on the underlying database. The OPM schema restructuring operations are briefly described below.

Manipulating attributes. OPM schema restructuring operations involving attributes include:

1. *Renaming attributes*.
2. *Converting between simple and tuple attributes*. Simple attributes can be grouped together to form a tuple attribute. A tuple attribute can be extended with additional component simple attributes. Conversely, a tuple attribute can be decomposed into a set of simple attributes consisting of its component attributes.
3. *Adding and removing attributes*. Attributes can be added to a class O_i by specifying the corresponding underlying ScDB constructs. Multiple OPM attributes can correspond to the same relational or ASN.1 attribute in the underlying ScDB. Attributes can be removed from an OPM view by marking them as deleted.
4. *Modifying attribute constraints*. The cardinality constraint associated with an OPM attribute can be modified. Similarly, the default value of an OPM attribute that is used for representing its null values can be changed.
5. *Changing the value class of an attribute*. The value class of an attribute can be changed and may entail a change of the attribute type such as from a primitive attribute to an abstract attribute.
6. *Adding and removing derived attributes*. Derived attributes can be added to, or removed from, a class. The procedures implementing the retrieval methods for derived attributes are automatically generated from the attribute derivations.

Manipulating classes. OPM schema restructuring operations involving classes include:

1. *Renaming classes*.
2. *Changing the type of classes*, where the type of a class can be *object*, *protocol*, *named primitive* or *controlled value* class.
3. *Adding and removing classes*. A class O_i can be added to an OPM view by specifying the corresponding underlying relations or types. Multiple classes can be derived from the same relation or type of the underlying ScDB. Classes can be removed from an OPM view by marking them as deleted.
4. *Adding and removing subclass relationships*. An object class O_i can be defined as a subclass of another object class O_j. Subclass relationships can also be removed from an OPM view.
5. *Adding and removing derived classes*. Derived classes can be added or removed from an OPM view. The procedures implementing the retrieval methods for derived classes are automatically generated from the class derivations.
6. *Converting classes into attributes*. A class can be converted into an attribute of another class. This operation can be used, for example, for converting 'artificial' OPM classes that represent set-valued or list-valued attributes but are not detected as such while constructing the canonical OPM view.

These operations can be applied sequentially in order to construct conceptually meaningful classes out of the simple translations of underlying schema found in the canonical OPM view.

Figure 3 shows the modification of an OPM view constructed for GSDB using the OPM retrofitting tool, via a Java based user interface: the upper-left and lower-left hand side windows show the attributes for classes `Feature` and `Feature_translations`, respectively, belonging to the canonical OPM view generated for the relational GSDB definition; the right-hand side window shows the result of converting class `Feature_translations` into tuple attribute `translations` of `Feature`, using the `Merge Class` retrofitting operation implemented via the lower-right hand side window, followed by the deletion of attributes `_aid` and `_oid` of class `Feature_translations`.

3.3. Documenting and Querying via OPM Views

ScDBs that have an OPM view can be documented, browsed and queried using Web based OPM tools [7]. Thus, the OPM view of an ScDB can be explored graphically on the Web using the OPM schema browser in the same way as shown in Figure 1 . Further, the OPM view of an ScDB can be used for documenting the ScDB in a variety of notations (including the native notation used for defining the ScDB) and formats (such as diagrams, LaTeX documents, etc). For one or several ScDBs with OPM views, a *Database Directory and Schema Library* documenting the ScDBs can be

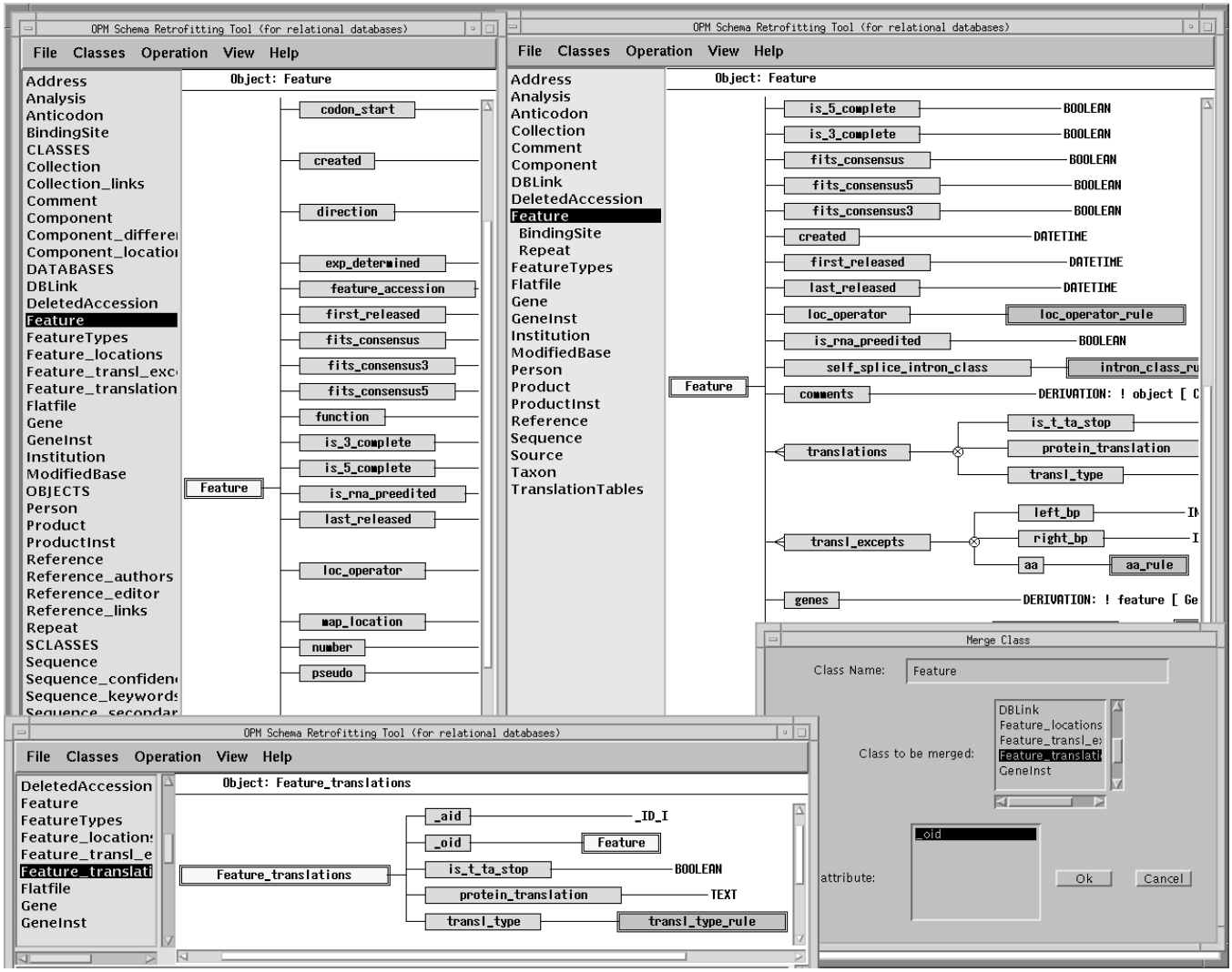

Figure 3. Refining OPM Views with the OPM Retrofitting Tool

generated automatically and can be accessed with a Web browser.

ScDBs with OPM views can be queried using Web based OPM tools. These tools allow specifying query (tree) structures in a graphical way, and then dynamically generating Web forms for specifying query conditions, as illustrated in Figure 4 . Queries over an OPM view are specified in the OPM query language (OPM-QL), an object-oriented query language similar to the ODMG standard, but with extensions supporting the unique features of OPM. OPM queries are evaluated using OPM *query translators* which map OPM-QL queries into queries specified in the query language of the underlying DBMS [5]. The mapping dictionary which is constructed during the retrofitting process, drives the OPM query translation and is used to determine the correspondence between constructs in the OPM view and the underlying ScDB constructs.

For ScDBs developed using a relational DBMS, such as Sybase or Oracle, an OPM query is translated into one or more queries expressed in the variant of SQL supported by the underlying DBMS, and subsequently the result of these SQL queries is converted into OPM objects. This conversion is also driven by the mapping directory.

For ScDBs represented using flat files, the OPM-QL translation depends on the access utilities provided for the ScDBs, and in general only a subset of OPM-QL can be translated. Further data processing is possible by using local query engines such as that of the OPM multidatabase query system (see section 5.1).

4. Retrofitting ScDBs: Case Studies

We discuss in this section our experience with retrofitting two major archival DNA sequence ScDBs, the Genome Sequence Database (GSDB) and Genbank, as part of our effort of constructing a federation of molecular biology ScDBs that will include the Genome Database (GDB), GSDB, the Protein Data Bank (PDB), and Genbank. GDB and PDB are implemented using OPM and the OPM tools, and therefore these ScDBs have native OPM schemas.

4.1. Retrofitting GSDB

The Genome Sequence Database (GSDB) at the National Center for Genome Resources, Santa Fe, is implemented using the Sybase relational DBMS. GSDB users are allowed read-only access through relational views. One view is defined on top of each relational table in order to restrict access to entries according to specific user permissions. Updates to GSDB are carried out through special data submission forms.

The OPM view of GSDB has been built on top of the relational views provided for public access. Accordingly, each GSDB relational view has been mapped to an object class in the canonical OPM view. Each GSDB relation or view is associated with an `id` attribute that has been mapped to the object identifier attribute of the corresponding OPM class. Primary and foreign key references are defined for GSDB relations, but not for the relational views. GSDB schema documentation is available on the Web as a postscript document.

The process of retrofitting GSDB with an OPM view consisted of the following steps:

1. The GSDB view definitions were expanded with primary and foreign key definitions. These definitions were based on the key definitions of the underlying tables and/or deduced from the schema documentation.

2. The canonical OPM view was generated from the GSDB relational tables and views.

3. Classes representing GSDB tables with restricted access were removed from the OPM view.

4. The conceptual objects modeled by GSDB, such as *sequences*, *features*, and *genes*, were identified. Some tables in GSDB represent such constructs, while other tables represent connections between objects or specializations of objects. Identifying the conceptual objects required studying the GSDB documentation, as well as the relational database definitions.

5. OPM constructs corresponding to GSDB attributes and tables that had administrative purposes rather representing application-specific concepts were also removed from the OPM view.

6. Some OPM classes were converted to attributes. These classes corresponded to relational tables or views representing set or list-valued attributes. Converting classes to attributes involved renaming and deleting attributes, merging classes, and in some cases converting tuple attributes into simple attributes.

7. Subclass (ISA) relationships were added to the OPM view. Some GSDB tables and views represent *specializations* of objects. Identifying such specializations required examining the GSDB documentation and relational definitions.

8. Related attributes were identified and grouped into tuple attributes. For example *baseseq-start* and *baseq-end* attributes, representing the cytogenetic position of a base sequence, were grouped into a tuple attribute *baseseq* with component attributes *start* and *end*.

9. Finally derived attributes were added to the OPM view. Most derived attributes consisted of compositions or inverses of other OPM attributes that were considered useful in exploring GSDB.

The retrofitting tool accepts restructuring commands either entered interactively or read from a script file. Because of the frequent revisions to the GSDB schema and refinements of the OPM views, the restructuring commands were recorded in a script file. This allowed editing the script file using a text editor and re-running the retrofitting process in order to generate a new OPM view whenever the underlying relational database changed or the OPM view was refined, for example by adding new derived attributes or classes.

The retrofitted OPM view for GSDB is available at http://gizmo.lbl.gov/jopmDemo/gsdb10.html, where it can be examined using the graphical OPM schema browser and queried using the OPM Web query tools. The OPM view documentation for GSDB has been included into a Molecular Biology Database Directory, that also includes Genbank, available on the Web at http://gizmo.lbl.gov/DM_TOOLS/OPM/MBD/MBD.html.

GSDB can be accessed via a Web form that allows searching on a limited (approx. 20) number of fields (attributes).[7] Using the OPM view for GSDB, one can construct queries involving any attribute that has a corresponding OPM representation. Figure 4 shows the specification

[7] See http://www.ncgr.org/gsdb/maestro/

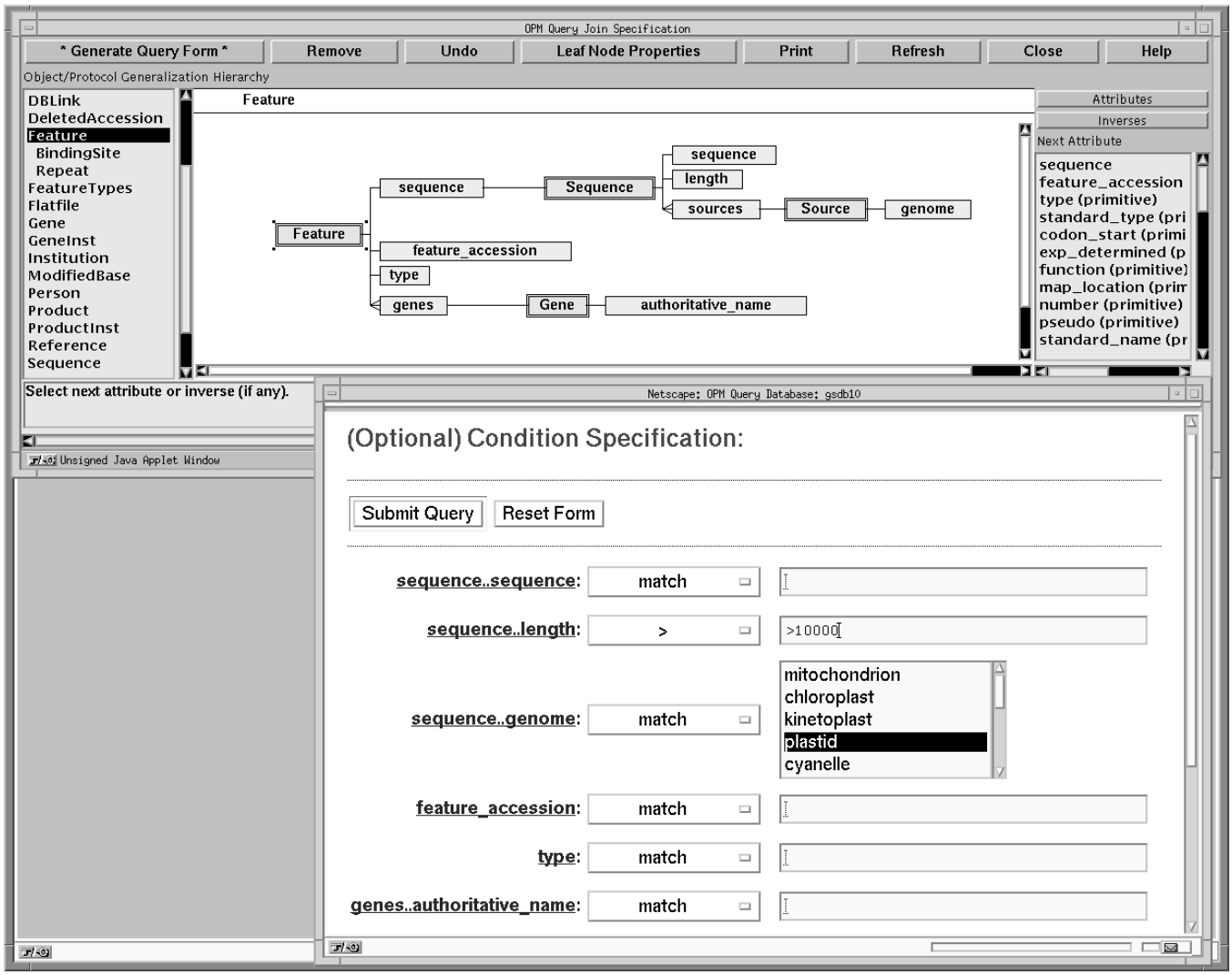

Figure 4. Querying GSDB with the OPM Web Query Tools

of a query over GSDB using the OPM Web query tools: the upper-side window shows the construction of a query tree containing attributes selected from the attributes associated with classes `Feature`, `Gene`, and `Sequence`; the lower-side window shows a dynamically generated Web query form containing the leaf attributes of the query tree, that can be used for specifying conditions and query submission.

Note that the performance of queries over retrofitted relational ScDBs may be limited by a number of factors, including the availability of indexes on various attributes and accessibility of the database server. In the case of native OPM ScDBs, the OPM tools automatically generate the index definitions needed for efficiently processing object-oriented queries. Retrofitted ScDBs, on the other hand, cannot be usually modified, and therefore the OPM query tools can use only indexes already defined for the underlying database. In the case of GSDB, certain important attributes are not indexed, leading to substantial performance differences for seemingly similar queries.

Further, in order to get good query performance, reliable query access to the underlying database is needed. Public query access to GSDB was found to be inconsistent, with great differences in the time required for evaluating identical queries, queries frequently becoming deadlocked, and sometimes even terminated after considerable delays.

4.2. Retrofitting Genbank

The Genbank DNA sequence repository at the National Center for Biotechnology Information (NCBI) is a file formatted using ASN.1 [21]. The process of retrofitting Gen-

bank with an OPM view consisted of the following steps:

1. The canonical OPM view was generated from the ASN.1 definition for Genbank. This stage involved name conversions for reconciling the different naming conventions employed in the ASN.1 definition for Genbank and OPM. For example, each ASN.1 name must start with a letter, and hyphens ("-") are allowed in names, while underscores ("_") are not allowed. Consequently, ASN.1 names had to be converted to (unique and legal) OPM names. Similarly, double quotes (") in ASN.1 descriptions have been replaced by single quotes (') in the corresponding OPM descriptions.

2. Auxiliary OPM classes were generated in order to represent ASN.1 nested types and unnamed enumerated vocabulary sets in the initial OPM canonical view. For example the ASN.1 type

```
Cit-art ::= SEQUENCE {
       -- article in journal or book
    title Title OPTIONAL ,
       -- title of paper (ANSI requires)
    authors Auth-list OPTIONAL ,
       -- authors (ANSI requires)
    from CHOICE {
       -- journal or book
        journal Cit-jour ,
        book Cit-book ,
        proc Cit-proc } }
```

   was mapped to the OPM classes:

```
OBJECT CLASS Cit_art
  DESCRIPTION:
     "article in journal or book"
  ATTRIBUTE title: [0,1] Title
     DESCRIPTION: "title of paper
                   (ANSI requires)"
  ATTRIBUTE authors: [0,1] Auth_list
     DESCRIPTION: "authors
                   (ANSI requires)"
  ATTRIBUTE from: [1,1] Cit_art_from

OBJECT CLASS Cit_art_from
  DESCRIPTION:
     "journal or book or proceedings"
  ATTRIBUTE journal: [0,1] Cit_jour
  ATTRIBUTE book: [0,1] Cit_book
  ATTRIBUTE proc: [0,1] Cit_proc
```

3. The OPM view for Genbank was refined by converting certain auxiliary OPM classes into tuple attributes of other classes, or by replacing a value class of an abstract attribute with a union of value classes. For example, the auxiliary OPM class `Cit_art_from` above was removed by replacing the value class of attribute `from` of class `Cit_art` to: `Cit_jour` or `Cit_book` or `Cit_proc`. Note that not all auxiliary OPM classes could be removed in this way, and therefore such modifications had to be handled one by one by refining the OPM view.

Genbank can be accessed via a Web based interface called Entrez [24]). Entrez also has a C API for accessing Genbank and other ASN.1 databases. Query access through Entrez is limited to a small number of "main entry" classes, and query conditions are limited to a small number of predetermined index fields. Further, accessing Entrez at the level of the C API requires hard-coding details of each database into the client application, meaning that a separate query tool would need to be developed for each NCBI database, and would need to be altered and re-compiled in order to reflect any changes to the underlying databases (such as new indexes being added).

After experimenting with a prototype OPM query tool based on Entrez and its C API, we decided against following this strategy for developing an OPM query tool for accessing flat file ScDBs, because of the limitations of this approach. We are currently working on developing generic OPM query tools for accessing flat file ScDBs such as Genbank. These tools will allow us to parse a variety of structured file formats, to add indexes without the need to recompile the OPM query tool nor alter the underlying flat file ScDB. We expect this query tool to be available in the fall of 1997.

5. Applications

An ScDB retrofitted with an OPM view can be (1) documented using OPM schema documentation tools, (2) browsed and queried using OPM query translators and interfaces, (3) incorporated into a federation of ScDBs with (native or retrofitted) OPM views; and (4) reorganized. Documenting and querying ScDBs that have OPM views has been discussed in the previous two sections. In this section we briefly describe the remaining applications for ScDBs with OPM views.

5.1. The OPM Multidatabase Query System

The OPM *multidatabase query tools* [6] provide facilities for querying multiple heterogeneous ScDBs that have a native OPM schema (developed with the OPM tools), or a retrofitted OPM view. The OPM multidatabase query processor executes multidatabase queries by splitting them into single database subqueries, which are processed using the

OPM query translators, and then performing local data processing and computation in order to combine the data resulting from the individual subqueries.

Since the query facilities supported by a particular DBMS or flat-file access system will determine the subset of the OPM-QL implemented by an OPM query translator, the multidatabase query system generates single-database subqueries dependent on the target DBMS and translator used, and performs local computation on the data retrieved in order to supplement the subset of OPM-QL supported. This capability can be also used in single-database mode in order to provide more general and flexible query facilities for systems such as ASN.1/Entrez which provide only limited query support.

Our approach to querying heterogeneous databases has similarities with the approach followed by Kleisli [4]: both approaches provide a single query language for querying ScDBs implemented using a variety of DBMSs. However, Kleisli is a purely value-based system and does not support any concept of schemas. Accordingly, Kleisli does not provide support in either formulating queries or interpreting their results: in order to use Kleisli to construct multidatabase queries, a programmer must have expert knowledge of each database involved in the query, its semantics, its data model and its native DBMS, as well as of CPL, the query language implemented by Kleisli.

We believe that providing an efficient, extensible mechanism for evaluating multidatabase queries is not in itself sufficient, and in addition it is necessary to provide support for exploring and documenting multiple heterogeneous ScDBs via a uniform model and interface, and to aid non-expert users in formulating multidatabase queries. Our approach requires an initial investment of time and effort in developing OPM views for ScDBs, but subsequently these views allow users to examine and construct complex queries across ScDBs.

5.2. Database Verification and Reorganization

An OPM view of an ScDB whose constraints have not been fully specified,[8] can be used for generating these constraints and/or for verifying whether the data in the ScDB satisfies the constraints entailed by the OPM view. Thus, the OPM Schema Translator generates both the DBMS integrity constraints and procedures required for maintaining the constraints expressed in the OPM view, as well as verification procedures that can be applied on an entire database for verifying whether it complies with the constraints entailed by the OPM view. These procedures detect all instances in the underlying database (i.e., the database subset) that are consistent with the OPM view.

[8] We encountered several production scientific and non-scientific databases with partial or no referential integrity constraint specifications.

An OPM view can be also employed for physically *reorganizing* an existing database. If a database is considered to represent inadequately the objects in the underlying application, or a database is scheduled to have a major revision, the OPM view for this database can be restructured and employed to *forward* engineer a new database. The OPM Schema Translator can then generate the specifications for this new database from the restructured OPM schema. During this process, the OPM Schema Translator generates a new mapping dictionary representing the correspondence between the OPM schema and the new database. The old and new mapping dictionaries can then be used for generating conversion procedures between the old and new versions of the database. A procedure following such a strategy has been applied recently for converting GDB 5 to GDB 6.

6. Concluding Remarks

We have described the procedure underlying a retrofitting tool that can be used for constructing and maintaining views for scientific databases (ScDBs), expressed using the Object-Protocol Model (OPM). The retrofitting tool generates a mapping dictionary that records the mapping information between the OPM views and the underlying ScDBs. The OPM view can be used for browsing and querying the underlying ScDB, for example with Web based OPM query tools, for constructing multidatabase systems, and for reorganizing ScDBs.

The retrofitting tool has been implemented as part of the OPM data management toolkit and has been employed to construct OPM views for several ScDBs, such as GSDB 1.0[7] and Genbank. The constructed OPM views have been subsequently used for constructing an OPM-based database federation consisting of GDB, PDB, Genbank and GSDB.

We plan to extend the retrofitting tool with richer and more powerful schema restructuring facilities and to support additional notations and DBMSs. We also intend to develop OPM schema evolution tools for reorganizing databases and specifying transformations between pre-existing OPM schemas. The major difference between schema retrofitting and schema evolution is that the former only involves changes to the view definition and the corresponding mapping dictionary but has no effects on the underlying database, and the latter involves physical changes to the underlying database structure and data.

Acknowledgements. We want to thank Stan Letovsky for providing numerous suggestions for improving the retrofitting tool. The anonymous reviewers provided useful suggestions that helped improving this paper.

References

[1] S. Abiteboul and A. Bonner. Objects and views. In *Proc. of the ACM SIGMOD Conference*, 1991.

[2] F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: The Story of O 2*. Morgan Kaufmann Publishers, Inc., 1992.

[3] J. Banerjee, et al. Data model issues for object-oriented applications. *ACM Trans. Office Inf. Syst.*, 5(1), 1987.

[4] P. Buneman, S. Davidson, K. Hart, C. Overton, and L. Wong. A data transformation system for biological data sources. In *Proc. of the 21st Int. Conference on Very Large Data Bases*, 1995.

[5] I. A. Chen, A. S. Kosky, V. M. Markowitz, and E. Szeto. The opm query translator. Technical Report LBL-38180, Lawrence Berkeley National Laboratory, 1996. Available at http://gizmo.lbl.gov/opm.html.

[6] I. A. Chen, A. S. Kosky, V. M. Markowitz, and E. Szeto. Opm*qs: The object-protocol model multidatabase query system. Technical Report LBL-38181, Lawrence Berkeley National Laboratory, 1996. Available at http://gizmo.lbl.gov/opm.html.

[7] I. A. Chen, A. S. Kosky, V. M. Markowitz, and E. Szeto. Exploring databases on the web. Technical Report LBNL-40340, Lawrence Berkeley National Laboratory, 1997.

[8] I. A. Chen and V. M. Markowitz. Modeling scientific experiments with an object data model. In *Proc. of the 11th Int. Conference on Data Engineering*, 1995.

[9] I. A. Chen and V. M. Markowitz. An overview of the object-protocol model (opm) and the opm data management tools. *Information Systems*, 20(5), 1995.

[10] I. A. Chen and V. M. Markowitz. Mapping object-protocol schemas into relational database schemas and procedures. Technical Report LBL-33048, Lawrence Berkeley National Laboratory, 1996. Available at http://gizmo.lbl.gov/opm.html.

[11] P. P. Chen. The entity-relationship model- towards a unified view of data. *ACM Trans. Database Syst.*, 1(1), Mar. 1976.

[12] C. J. Date. Referential integrity. In *Relational Database-Selected Writings*. Addison-Wesley, 1986.

[13] K. H. Davis and A. K. Arora. Converting a relational database model into an entity-relationship model. In S. March, editor, *Proc. of the 6th Int. Conference on Entity-Relationship Approach*. Elsevier Science Publishers B.V., 1987.

[14] N. Goodman. An object-oriented dbms war story: Developing a genome mapping database in c++. In W. Kim, editor, *Modern Database Management: Object-Oriented and Multidatabase Technologies*. ACM Press, 1994.

[15] M. Hammer and D. McLeod. Database description with sdm: A semantic database model. *ACM Trans. Database Syst.*, 6(3), Sept. 1981.

[16] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv.*, 19(3), Sept. 1987.

[17] P. Johannesson. A method for transforming relational schemas into conceptual schemas. In *Proc. of the 10th Int. Conference on Data Engineering*, 1994.

[18] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[19] V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Trans. Softw. Eng.*, 16(8), Aug. 1990.

[20] National Center for Biotechnology Information. *The NIH Generic Sequence Database*. http://www.ncbi.nlm.nih.gov/Web/Genbank/index.html.

[21] National Center for Biotechnology Information, Bethesda, Maryland. *Programmer's Reference*, Nov. 1991. See also http://www.inria.fr:80/rodeo/personnel/hoschka/asn1.html.

[22] W. J. Premerlani and M. R. Blaha. An approach for reverse engineering of relational databases. *Commun. ACM*, 37(5), May 1994.

[23] E. A. Rundensteiner. Multiview: A methodology for supporting multiple views in object-oriented databases. In *Proc. of the 18th VLDB Conference*, 1992.

[24] G. D. Shuler, J. A. Epstein, H. Ohkawa, and J. A. Kans. Entrez. In R. Doolittle, editor, *Methods in Enzymology*. Academic Press, Inc., In press. See also http://www3.ncbi.nlm.nih.gov/Entrez/.