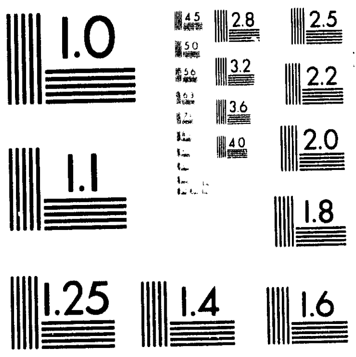

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

LBL-33706

Data Management Tools for Genomic Applications: A Progress Report

Victor M. Markowitz and I-Min A. Chen

Information and Computing Sciences Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

September 1993

This work was supported by the Office of Health and Environmental Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

[illegible]

Data Management Tools for Genomic Applications: A Progress Report [*]

Victor M. Markowitz and I-Min A. Chen

Information and Computing Sciences Division
Lawrence Berkeley Laboratory, Berkeley, CA 94720

Abstract

We report in this paper on the development of data management tools that allow scientists to construct and manipulate genomic databases in terms of application-specific objects and protocols. We are developing tools for specifying genomic database structures, as well as for entering, changing, maintaining, browsing and querying data in genomic databases. These tools are based on the Object-Protocol Model (OPM) developed by us and target commercial relational database management systems which are widely used in molecular biology laboratories. OPM allows scientists to interact with genomic databases in terms of their own frame of reference, namely genomic objects and protocols. Databases developed using the data management tools are easier to use, manage, and adapt.

1 Introduction

The information controlling the development of biological organisms is encoded in their *genome* in the form of polymeric molecules known as DNA. DNA information is encoded as a sequence of *nucleotides*. Regions of the DNA called *genes* specify the information for protein molecules. In higher organisms (yeast, plants, animals, humans) the DNA is organized into several linear *chromosomes*.

Several projects are attempting to determine the complete DNA sequence of various organisms. These projects require databases for managing DNA data and related information. Typically, the structure of a genomic database can be modeled in terms of *objects* characterized by (having) *attributes* that take values from a *domain* (set of values); objects that share common attributes can be organized (*classified*) into homogeneous sets of objects. For example, consider the *contig maps* used in determining the complete DNA sequence of various

[*]Issued as Technical Report LBL-33706. This work is supported by the Office of Health and Environmental Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

organisms, and consisting of ordered *DNA fragments*[1]. Contig maps can be modeled as objects that have attributes such as `contig_id`, `owner` (representing owners of contig maps), and (`fragment, position`) (representing component fragments and their positions in contig maps); similarly, fragments can be modeled as objects that have attributes such as `fragment_id` and `owner`.

Genomic databases also contain data on *protocols* representing experimental laboratory procedures. Given an input, a protocol instance (i.e., an elementary experiment) results in an output. Protocols often involve a series of subprotocol steps. The recursive specification of protocols in terms of component subprotocols is called *protocol expansion.* Protocol expansion reveals the composition of component subprotocols and/or alternative ways of performing the protocol. For example, consider a *construct* protocol for constructing contig maps of ordered DNA fragments: such a protocol is applied on DNA fragments (input) and result in contig maps (output). Protocol *construct* can be expanded into two alternative protocols, *overlap* and *constraint*, both followed by protocol *assemble*: protocol *overlap* compares two DNA fragments using a computer program, protocol *constraint* compares manually two DNA fragments according to certain constraints, and protocol *assemble* assembles DNA fragments into a contig map according to information in the connection tables regarding possible connecting positions of two DNA fragments.

Most genomic databases developed in the past few years use commercial relational database management systems (DBMSs). Relational DBMSs do not provide constructs for representing directly genomic-specific objects and protocols. These objects and protocols are usually represented in relational databases by several disconnected tuples scattered among multiple tables, logically tied together by primary key-foreign key references. Such representations are not only hard to comprehend, but also entail the development of large procedures for assembling data on application-specific objects from (i.e., by joining) several relations. Furthermore, because of the complexity of the relational representations for objects and protocols, the development, maintenance, and modification of such databases are tedious, error-prone, and time-consuming processes.

Data models such as the *Extended Entity-Relationship Model* (EERM) [10] and the *Semantic Data Model* (SDM) [4] provide constructs for modeling objects, sets of objects, and object associations, and therefore are better suited than relational DBMSs for specifying the structure of genomic databases. For example, in EERM atomic objects called entities are classified into *entity-sets*, and are qualified by *attributes* that take values from value-sets. Associations of entities are modeled as *relationships* classified in *relationship-sets*. EERM has a *generalization* mechanism that allows viewing similar (*specialization*) entity-sets as a single *generic* entity-set.

We have explored using EERM for describing genomic databases [9], and found that it is too restricted for specifying accurately their object structure. Such restrictions can be overcome by using *auxiliary* entity-sets and relationship-

[1] Since existing technology permits sequencing only fragments of a few hundred nucleotides, chromosomal DNA is cut into smaller fragments, the fragments are propagated as clones, and then assembled into contig maps.

sets. For example, contig maps, fragments and their owners can be represented by three EERM entity-sets called CONTIG_MAP, FRAGMENT, and PERSON, respectively. However, representing that contig maps and fragments can be owned by persons requires an auxiliary entity-set generalizing CONTIG_MAP and FRAGMENT, OWNED_OBJECT, together with an auxiliary relationship-set, OWNED_BY, associating OWNED_OBJECT with PERSON. Auxiliary constructs do not represent application-specific objects and therefore unnecessarily increase the complexity and obscure the semantics of databases.

The need to employ a diversity of continuously evolving mapping and sequencing strategies require facilities for efficiently constructing genomic databases that are easy to use and change. In order to attain the desired level of flexibility and adaptability, we decided to develop data management tools that allow scientists to rapidly construct and manipulate genomic databases in terms of genomic objects and protocols. The underlying data model for these tools is provided by the Object-Protocol Model (OPM) developed by us.

OPM has similarities with other object data models (cf. [5]), especially with SDM [4]. Similar to SDM, in OPM objects are classified into object classes and are qualified by attributes that take values from value classes. Unlike SDM, however, in OPM attributes can be *composite*, that is, consisting of multiple component simple attributes, and can be associated not only with single value classes, but also with *unions* of value classes. These constructs allow avoiding the creation of object classes that do not have an application-specific counterpart. Furthermore, unlike other data models (e.g., such as those reviewed in [5] or [6]), OPM provides a *protocol class* construct for modeling laboratory experiments. A protocol class in OPM can be associated with regular attributes as well as *input* and *output* attributes used for specifying input-output protocol connections. OPM also supports a *protocol expansion* mechanism for specifying a protocol class in terms of component subprotocol classes.

The data management tools we develop will benefit several molecular biology laboratories and genome centers. In particular, our project supports directly the large-scale sequencing project at University of Washington, Seattle, for characterizing up to six million bases of the human and mouse T-cell receptor loci and the development of the Integrated Genomic Database at the German Cancer Research Center, at Heildelberg.

The rest of the paper is organized as follows. Our approach to developing data management tools is described in section 2. The status of our work is reviewed in section 3. Section 4 briefly discusses future plans.

2 Approach

The data management tools are based on a data model developed by us, the *Object-Protocol Model* (OPM). OPM is briefly reviewed below. A complete description of OPM is provided in [1].

2.1 The Object-Protocol Model

OPM allows describing database structures in terms of *objects* characterized by *attributes* taking values from *value classes*, and classified into *object classes*. For example, the contig maps mentioned in the previous section can be represented in OPM by object class `CONTIG_MAP` having attributes **`contig_id`**, **`owner`**, and (**`fragment`**, **`position`**). Similarly, fragments can be represented by object class `FRAGMENT` having attributes **`fragment_id`**, **`sequence`**, **`length`**, and **`owner`**; and owners can be represented by object class `PERSON` having attributes **`person_id`**, **`name`** and **`owns`** (see figure 1).

Object classes can have subclass-superclass relationships. For example, one can specify a class `SCIENTIST` as a subclass of `PERSON`.

Attributes in OPM can be:

1. *atomic*, such as attribute **`contig_id`** of object class `CONTIG_MAP`, or *composite*, that is, consisting of aggregations of atomic attributes, such as attribute (**`fragment`**, **`position`**) of `CONTIG_MAP`;

2. *single-valued*, such as attribute **`person_id`** of object class `PERSON`, or *multi-valued*, such as attribute **`owns`** of `PERSON`;

3. *local*, such as attribute **`sequence`** of object class `FRAGMENT`, or *referential*, that is, representing references to other objects, such as attribute **`owner`** of `FRAGMENT`, representing references to `PERSON`;

4. associated with a *single* domain, such as attribute **`name`** of object class `PERSON`, or with a *union* of different domains, such as **`owns`** of `PERSON` whose domain is the union of object classes `CONTIG_MAP` and `FRAGMENT`;

5. *derived*, that is, attributes that have values derived from the values of other attributes using a derivation expression, such as attribute composition, arithmetic expressions, aggregate functions, or attribute inversion; for example, attribute **`owner`** of `CONTIG_MAP` in figure 1 is specified as the inverse of attribute **`owns`** of `PERSON` (i.e., the value of **`owner`** for a given contig map m is the person whose **`owns`** value contains m).

In addition to objects, OPM supports modeling laboratory *protocols*. Protocols are classified in protocol classes and can be qualified by both regular and special, input and output, attributes. For example, protocols *construct*, *overlap*, *constraint*, and *assemble* mentioned in the previous section can be described in OPM by the protocol classes shown in figure 2, where their inputs and outputs are modeled by the object classes shown in figure 1. Thus, the experiments for constructing contig maps of ordered DNA fragments can be represented by the instances of protocol class `CONSTRUCT` having an output attribute **`contig_map`** representing the result of construct protocols applied on fragments, where fragments are represented by input attribute **`fragments`**.

OPM has a protocol expansion mechanism for the recursive specification of protocols in terms of *alternative* protocols, *sequences* of protocols, and *optional* protocols; "*or*", ",", and "[ ]" are used to denote alternative, sequences

```
OBJECT CLASS FRAGMENT
   DESCRIPTION: DNA fragment
   ID: fragment_id
   ATTRIBUTE fragment_id: INTEGER          not null     single-valued
   ATTRIBUTE sequence: VARCHAR(750)                     single-valued
   ATTRIBUTE length: INTEGER                            single-valued
   ATTRIBUTE owner: PERSON                              single-valued
             DERIVATION: inverse of PERSON.owns
OBJECT CLASS CONNECTION_TABLE
   DESCRIPTION: connection table
   ID: table_id
   ATTRIBUTE table_id: INTEGER             not null     single-valued
   ATTRIBUTE left_entry: FRAGMENT
   ATTRIBUTE right_entry: FRAGMENT
   ATTRIBUTE distance: INTEGER
OBJECT CLASS CONTIG_MAP
   DESCRIPTION: contig map
   ID: contig_id
   ATTRIBUTE contig_id: INTEGER            not null     single-valued
   ATTRIBUTE (fragment, position): (FRAGMENT, INTEGER)  multi-valued
   ATTRIBUTE owner: PERSON                              single-valued
             DERIVATION: inverse of PERSON.owns
OBJECT CLASS PERSON
   DESCRIPTION: person
   ID: person_id
   ATTRIBUTE person_id: INTEGER            not null     single-valued
   ATTRIBUTE name: CHAR(80)
   ATTRIBUTE owns: CONTIG_MAP or FRAGMENT               multi-valued
```

Figure 1: Object Classes Representing the Input and Output for Protocols

of, and optional protocols, respectively, and *parentheses* are used for specifying complex protocol compositions. For example, consider protocol classes **CONSTRUCT**, **OVERLAP**, **CONSTRAINT**, and **ASSEMBLE** shown in figure 2. The expansion of **CONSTRUCT** in terms of **OVERLAP**, **CONSTRAINT**, and **ASSEMBLE** is expressed as follows (see figure 2): **EXPANSION: (OVERLAP or CONSTRAINT), ASSEMBLE.**

Input and *output* attributes associated with protocols represent the input and output of protocols, respectively, and can be used to express the inheritance of input or output attributes by component subprotocols from their generic protocols and the input—output connection of directly related protocols. Input—output attribute inheritance is expressed using *'input is-a ...'* statements (e.g., see attribute `fragments` of OVERLAP in figure 2) and *'output is-a ...'* statements (e.g., see attribute `contig_map` of **ASSEMBLE** in figure 2) in the specification of the input and output attributes associated with subprotocols. If a protocol is followed directly by another protocol, then the input of the latter may include some or all of the output of the former. Such input—output

```
PROTOCOL CLASS CONSTRUCT
   DESCRIPTION: construct a contig map
   ID: construct_id
   EXPANSION: (OVERLAP or CONSTRAINT), ASSEMBLE
   ATTRIBUTE construct_id: INTEGER     not null   single-valued
   ATTRIBUTE fragments: FRAGMENT       not null   multi-valued    input
   ATTRIBUTE contig_map: CONTIG_MAP    not null   single-valued   output
PROTOCOL CLASS OVERLAP
   DESCRIPTION: compare fragments using computer programs
   ID: overlap_id
   ATTRIBUTE overlap_id: INTEGER         not null   single-valued
   ATTRIBUTE fragments: FRAGMENT         not null   multi-valued
                              input isa CONSTRUCT.fragments
   ATTRIBUTE connect_table: CONNECTION_TABLE  not null          output
   ATTRIBUTE (program_name, program_version): (CHAR(40), CHAR(6))
PROTOCOL CLASS CONSTRAINT
   DESCRIPTION: manually compare fragments using constraints
   ID: constraint_id
   ATTRIBUTE constraint_id: INTEGER      not null    single-valued
   ATTRIBUTE fragments: FRAGMENT         not null    multi-valued
                              input isa CONSTRUCT.fragments
   ATTRIBUTE connect_table: CONNECTION_TABLE   not null   output
   ATTRIBUTE constraint_type: CHAR(80)         single-valued
PROTOCOL CLASS ASSEMBLE
   DESCRIPTION: assemble contigs
   ID: assemble_id
   ATTRIBUTE assemble_id: INTEGER        not null    single-valued
   ATTRIBUTE connect_table: CONNECTION_TABLE  not null
                              input from OVERLAP via connect_table
                                     or CONSTRAINT via connect_table
   ATTRIBUTE contig_map: CONTIG_MAP      not null    single-valued
                              output isa CONSTRUCT.contig_map
```

Figure 2: Protocol Classes Representing Protocol Construct and its Components

attribute connections are expressed using '*input from* ... *via* ...' statements (e.g., see attribute connect_table of ASSEMBLE in figure 2) in the specification of input attributes associated with protocols taking their input from other protocols.

2.2 Data Management Tool Development

Developing data management tools based on OPM and targeting relational DBMSs, involves mapping OPM constructs and data manipulation operations (retrievals and updates) into relational DBMS constructs and SQL queries. This mapping is very complex because of the discrepancy between the OPM and relational DBMS constructs, but can be simplified by introducing an intermediate EERM level that allows decomposing the OPM to relational DBMS mapping into simpler mappings between OPM and EERM, and between EERM and relational DBMS, respectively. The OPM to EERM mapping is easier to develop than the direct OPM to relational DBMS mapping because EERM schemas and queries are specified in terms of objects and object associations, and therefore are inherently more concise and simpler to specify than relational DBMS schemas and queries. Furthermore, EERM schemas and queries are independent of a specific DBMS, and therefore can be used across different DBMS platforms. The EERM version we use is the EERM described in [10], extended with two constructs (unary relationship-sets and a new form of directly associating entity-sets) described in [2].

We have developed a mapping of OPM schemas that generates EERM schemas together with queries for constructing OPM objects and protocols from entities and relationships. These queries are expressed in the *Concise Object Query Language* (COQL) [11], and involve associating a (primary) entity-set with attributes of other (auxiliary) entity-sets and relationship-sets, where the primary entity-set is associated with the auxiliary entity-sets and relationship-sets either directly or via other entity-sets and relationship-sets. Thus, a primary entity-set, its local and inherited attributes as well as the attributes of auxiliary entity-sets and relationship-sets can be specified in COQL using an OUTPUT statement, while the association of a primary entity-set with auxiliary entity-sets and relationship-sets can be expressed using CONNECTIONS statements. COQL also allows setting conditions on entity-sets and relationship-sets. Suppose that the contig maps and their owners mentioned above are represented by entity-sets `CONTIG_MAP` (with attribute `contig_id`) and `PERSON`, connected by relationship-set `OWNED_BY`. Then the following COQL query expresses the association of contig maps with their owners:

```
OUTPUT         CONTIG_MAP: contig_id, PERSON;
CONNECTIONS    CONTIG_MAP OWNED_BY PERSON;    END
```

In the COQL query above, `PERSON` is an auxiliary entity-set whose attributes are associated with `CONTIG_MAP` via relationship-set `OWNED_BY`.

Regarding the mapping of EERM schemas and queries into relational DBMS schemas and queries, we have developed tools that can automatically carry out the EERM to relational DBMS schema and query mapping. The EERM schema to relational schema mapping is presented in [10] and has been implemented as part of an EERM schema translation tool called SDT [8]. SDT automatically translates EER schemas into schema definitions for several relational DBMSs: Sybase, Ingres, Informix, and Oracle. The DBMS database definitions generated by SDT include procedures (e.g., triggers in Sybase) necessary for maintaining referential integrity and value constraints. The information about schemas and their mapping is subsequently stored in a *metadatabase*.

The COQL to SQL mapping is described in [11], and has been implemented as part of a COQL translation tool. Based on the metadatabase generated by SDT, the COQL translator maps a COQL query into one or several queries in the SQL dialect of the underlying relational DBMS. The COQL translator has been implemented for Sybase and will be implemented for other relational DBMSs as well.

3 Development Status

In this section we briefly review the status of the OPM data management tools.

3.1 The OPM Schema Editor

We have developed a graphical schema editor for interactively specifying, displaying, modifying, merging, and browsing OPM schemas.

The OPM schema editor allows specifying incrementally complex object and protocol structures by providing facilities for defining new schemas, modifying existing schemas, and merging schemas. A schema can be browsed using an Object Classes Listbox that lists in the main window the object classes of the schema (see figure 3). This listbox can be switched into a Protocol Classes Listbox or a Controlled Value Classes Listbox for browsing protocol classes and controlled value classes, respectively.

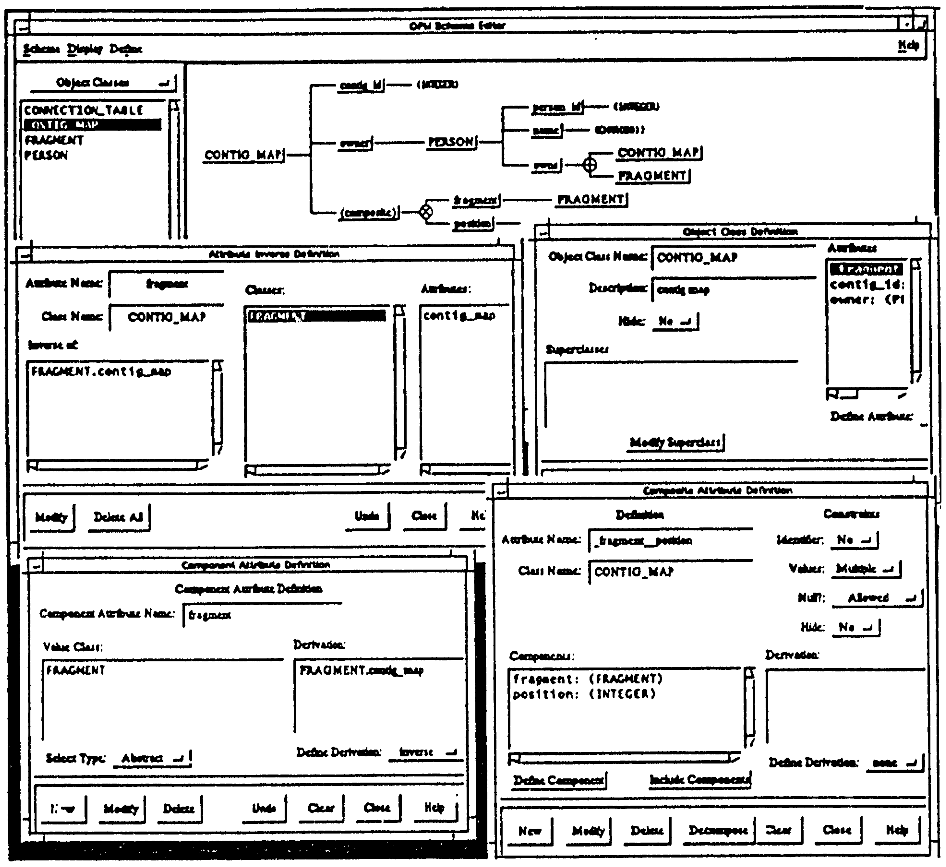
For an object class selected in the Object Classes Listbox, its connections to other classes (via attributes), and its superclasses and subclasses are displayed in the drawing area of the main window. This graphical display can be also used for browsing a schema by recursively expanding value classes associated with displayed attributes.

An object class can be defined or modified by double clicking on the name of an object class in the listbox or in the drawing area, or by selecting the **OPM Object Class** option of the **Define** menu item in the main window. The Object Class Definition window shown in figure 3 illustrates the definition of object class **CONTIG_MAP**. The Define Attribute option in this window allows defining or modifying attributes of the current class.

The Composite Attribute Definition and Component Attribute Definition windows shown in figure 3 illustrate the definition of a composite attribute, namely attribute **(fragment, position)** of **CONTIG_MAP**. The Attribute Inverse Definition window shown in the same figure allows specifying object cross referencing by defining attributes as inverses of other attributes.

Protocol classes can be browsed, defined or modified in a similar way. For a protocol class selected in the Protocol Classes Listbox, its connections to other classes via attributes, as well as the graphical representation (in a DFD like notation) of its expansion (if any) are displayed in the main window drawing area. A protocol object class can be defined or modified by double clicking on the name of a protocol class in the listbox or in the drawing area, or by selecting the **OPM Protocol Class** option of the **Define** menu item in the main window. Figure 4 illustrates the definition of a protocol class (**CONSTRUCT**) and its subprotocols. The Input/Output Attribute Definition window allows specifying the input and output attributes of protocols. Protocol expansion can be defined or modified using the Protocol Expansion window. The Input From Definition window allows specifying '*input from ... via*' connections. For example, in figure 4) attribute **connect_table** of protocol class **ASSEMBLE** is specified as: from **OVERLAP** via **connect_table** or from **CONSTRAINT** via **connect_table**

A schema can be saved in a text file by selecting **Generate OPM** option of the



Object Classes: CONNECTION_TABLE, CONTIG_MAP, FRAGMENT, PERSON

Attribute Inverse Definition — Attribute Name: fragment; Class Name: CONTIG_MAP; Inverse of: FRAGMENT.contig_map; Classes: FRAGMENT; Attributes: contig_map

Object Class Definition — Object Class Name: CONTIG_MAP; Description: contig map; Hide: No; Superclasses:; Attributes: fragment, contig_id:, owner: (PI

Composite Attribute Definition — Definition: Attribute Name: _fragment__position; Class Name: CONTIG_MAP; Constraints: Identifier: No; Values: Multiple; Null?: Allowed; Hide: No; Components: fragment: (FRAGMENT), position: (INTEGER); Derivation:; Define Derivation: none

Component Attribute Definition — Component Attribute Name: fragment; Value Class: FRAGMENT; Derivation: FRAGMENT.contig_map; Select Type: Abstract; Define Derivation: inverse

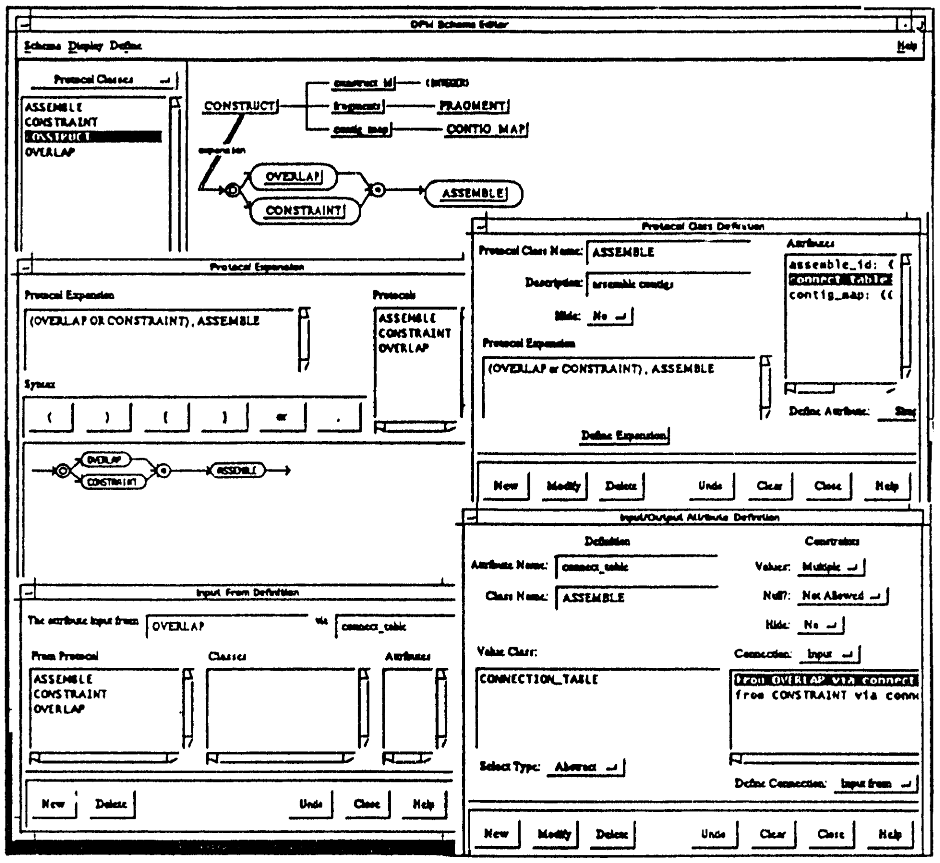
Figure 3: Specifying Object Classes using the OPM Schema Editor

Schema menu item in the main window. This file contains the schema definition in the OPM data definition language, and can be passed to the OPM schema translator described below, for generating the corresponding EER schema and COQL queries.

The OPM schema editor has been implemented on Sun SPARCstations using C++ and the X11 Motif graphical user interface toolkit and is described in [3].

3.2 The OPM Schema Translator

Since scientists in most molecular biology laboratories use commercial relational DBMSs (mainly Sybase), the tools we develop target relational DBMSs. Consequently, these tools involve mapping OPM schemas into relational schema definitions and SQL queries that express basic manipulations (retrievals and updates) of OPM objects and protocols.



Protocol Classes: ASSEMBLE, CONSTRAINT, CONSTRUCT, OVERLAP

Protocol Expansion: (OVERLAP OR CONSTRAINT), ASSEMBLE

Protocols: ASSEMBLE, CONSTRAINT, OVERLAP

Protocol Class Name: ASSEMBLE

Description: assemble contigs

Hide: No

Protocol Expansion: (OVERLAP or CONSTRAINT), ASSEMBLE

Attributes: assemble_id: (, connect_table, contig_map: ((

Attribute Name: connect_table

Class Name: ASSEMBLE

Value Class: CONNECTION_TABLE

Select Type: Abstract

Values: Multiple

Null?: Not Allowed

Hide: No

Connection: Input

from OVERLAP via connect

from CONSTRAINT via conn

Define Connection: Input from

The attribute input from OVERLAP via connect_table

From Protocol: ASSEMBLE, CONSTRAINT, OVERLAP

Figure 4: Specifying Protocol Classes using the OPM Schema Editor

As already mentioned above, our approach to mapping OPM schemas into relational definitions and queries is to use an intermediate EERM level, so that OPM schemas are mapped first into EERM schemas and queries, and then EERM schemas and queries are mapped into relational database schema definitions and queries. This approach allows reducing the development of a complex OPM to relational DBMS mapping to a simpler OPM to EERM mapping, while taking advantage of the existing EERM to relational DBMS translation tools [8, 11] for generating relational database definitions and queries from EERM schemas and queries.

The complete specification of the OPM schema mapping procedure and examples can be found in [2]. Informally, mapping OPM into EERM consists of mapping every OPM object or protocol class into an entity-set, and of incrementally constructing COQL queries associated with these entity-sets, that express the construction (retrieval) of OPM objects and protocols from EERM entities and relationships. Depending on their type (primitive, abstract, simple, composite, etc.), non-derived attributes of object or protocol classes are mapped into EERM attributes, direct entity-set associations, or relationship-sets. Derived attributes are not mapped into EERM schema components (with the exception of some inverse attributes) and entail only modifying COQL queries. For each (object or protocol) class, the mapping generates a COQL query for retrieving the instances in this class, including the values for all their non-derived, derived, and inherited attributes. The mapping also generates a *metadatabase* that contains information on the correspondence between the components of the OPM schema and the components of the generated EER schema and COQL queries. The OPM browsing and query tools we plan to develop will be based on this metadatabase.

The OPM schema translator has been developed on Sun SPARCstations in C++ using Lex++ and Yacc++.

4 Summary and Future Plans

We have briefly discussed the development of data management tools that allow specifying genomic database structures. These tools are based on the Object-Protocol Model (OPM) developed by us and target commercial relational database management systems. These tools have been applied to the development of a genomic database supporting the sequencing project at University of Washington, Seattle.

We are currently developing OPM data entry and browsing tools. These tools will provide facilities for: (i) inserting, deleting, and updating objects and protocols; (ii) selecting and displaying objects and protocols that satisfy certain conditions; (iii) browsing through selected sets of objects and protocols; (iv) recursively displaying, for a given object or protocol, related objects or protocols.

We also plan to develop a more complex OPM query language and a query tool based on this language. This tool will allow querying genomic databases in terms of objects and protocols, and will consist of two main components: (i) an OPM-based graphical interface will allow users to browse through OPM schema specifications and incrementally specify queries in terms of object and protocols; and (ii) a translator will map OPM queries into COQL queries, and subsequently into SQL queries.

The data management tools we develop are currently targeting the Sybase DBMS, mainly because Sybase is widely used in molecular biology laboratories and centers worldwide. We recognize that relational databases are cumbersome for implementing genomic databases. Since object-oriented DBMSs are more amenable to represent complex protocols and DNA sequences and provide mechanisms for incorporating application-specific (e.g., sequence alignment) operators, we plan to use such DBMSs for genomic databases. We will experiment with one of the C++ based object-oriented DBMSs, such as Object Store, and will extend our tools in order to ensure an easy transfer of genomic databases to these DBMSs.

Acknowledgements. The OPM schema editor was implemented by Ofer Ben-Shachar, Francis Pang, and Carol Jean Smith. SDT was implemented by Weiping Fang and Jun Wang. The COQL translator has been implemented by Ernest Szeto. We greatly appreciate their excellent work.

References

[1] Chen, I.A., and Markowitz, V.M., The Object-Protocol Model, Lawrence Berkeley Laboratory Technical Report LBL-32738, 1993.

[2] Chen, I.A., and Markowitz, V.M., Mapping Object-Protocol Schemas into Extended Entity-Relationship Schemas and Queries, Lawrence Berkeley Laboratory Technical Report LBL-33048, 1993.

[3] Chen, I.A., Markowitz, V.M., Ben-Shachar, O., and Pang, F., OPM Schema Editor 1.1: A Graphical Editor for Object-Protocol Schemas, Lawrence Berkeley Laboratory Technical Report LBL-33410, 1993.

[4] Hammer, M., and McLeod, D., Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, 6, 3, (1981), pp. 351-386.

[5] Hull, R., and King, R., Semantic Database Modeling: Survey, Applications, and Research Issues, *Computing Surveys* 19, 3 (1987), pp. 201-260.

[6] Ioannidis, Y.E., and Livny, M., MOOSE: Modeling Objects in a Simulation Environment, *Information Processing 89*, G.X. Ritter (ed), Elsevier Science Publishers B.V., 1989, pp. 821-826.

[8] Markowitz, V.M., Fang, W., and Wang, J., SDT 6.0. A Schema Definition and Translation Tool for Extended Entity-Relationship Schemas, Lawrence Berkeley Laboratory technical Report LBL-27843, 1993.

[9] Markowitz, V.M., Lewis, S., McCarthy, J., Olken, F., and Zorn, M., Data Management for Genomic Mapping Applications: A Case Study, *Proc. of the 6th International Conference on Scientific and Statistical Database Management*, 1992, pp. 45-57.

[10] Markowitz, V.M., and Shoshani, A., Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach, *ACM Transactions on Database Systems*, 17, 3, (1992), pp. 423-464.

[11] Markowitz, V.M., and Shoshani, A., Object Queries over Relational Databases: Language, Implementation, and Applications, *Proc. of the 9th International Conference on Data Engineering*, 1993, pp. 71-80.

DATE
FILMED
1/12/94
END