SIM: A Database System Based on the Semantic Data Model

D. Jagannathan, R. L. Guck, B. L. Fritchman,
J. P. Thompson and D. M. Tolbert.

Unisys Corporation, Irvine, CA 92718.

Abstract

SIM is a fully featured, commercially available database management system based on a semantic data model similar to Hammer and McLeod's SDM. SIM has two primary modeling goals. The first is to narrow the gap between a user's real-world perception of data and the conceptual view imposed by the database system because of modeling presuppositions or limitations. The second goal is to allow, as much as possible, the semantics of data to be defined in the schema and make the database system responsible for enforcing its integrity. SIM provides a rich set of constructs for schema definition, including those for specifying generalization hierarchies modeled by directed acyclic graphs, interobject relationships and integrity constraints. It also features a novel, easy-to-use, English-like DML. This paper describes the key modeling features of SIM, the architecture of the system and its implementation considerations.

1. Introduction

A data model consists of rules for defining the logical structure of data and associated operations. Expressive power, simplicity and freedom from implementation details are some desirable characteristics of a data model. The relational model was the first to emphasize both the structural and manipulative aspects of modeling as well as storage independence. It is built on mathematical foundations and its often-quoted advantages are the simplicity and completeness of its concepts. However, the principal weakness of the relational model is its lack of *semantic expressive power* -- it does not have constructs which can directly capture application semantics known to the database designer [HaMc81]. It requires that concepts of an application be fragmented to suit the model, forcing the resulting schema and queries on the database to lose their *conceptual naturalness* [Ship81]. Artificial steps in query formulations introduce a level of indirection and have procedural overtones.

Semantic data models address this weakness of the relational model. Their model of reality is usually based on abstract entities (objects) rather than records; they provide for interobject relationships and structural constraints. There is generally no consensus on what constitutes a semantic model and a number of them have been proposed. These include the binary relational model which views a database as objects and binary relationships between objects [Abri74], the entity-relationship model which treats the database as entities and n-ary relationships between entities [Chen76], semantically enriched relational models [Codd79], the functional model which treats entities, attributes and relationships as functions with zero or more arguments [Ship81], the semantic model SDM [HaMc81] and the object-oriented data models [Fish87,BKK87]. Many data definitional ideas of SIM are derived from SDM, while the DML is our own.

The SIM (Semantic Information Manager) project was initiated four years ago at Unisys with the goal of producing the next generation of DBMS tools. The semantic model was chosen because it is rich, expressive, conceptually natural and provides a path of growth and evolution for existing DMSII users. DMSII is a DBMS based on the network data model and runs on Unisys A Series machines. It has a large installed base of users and has been used to implement many large and complex applications. SIM has initially been built on top of DMSII and relies on DMSII for transaction, cursor and I/O management. However, the architecture of the system is designed such that virtually any data source, including other database systems, can be substituted in the place of DMSII. SIM forms the basis for the InfoExec[tm]* Environment which provides an array of database and

* InfoExec is a trademark of Unisys Corporation.

application tools, including ADDS, IQF and WQF. ADDS is a data dictionary system which can be used for, among other things, defining a SIM database. IQF is a menu-based query facility and WQF is a workstation-based graphically-oriented query language. The InfoExec Environment also supports SIM database interfaces in COBOL, ALGOL and Pascal.

All examples in this paper are based on the schema of the UNIVERSITY database in Section 7. This schema and examples based on it are described in the conceptual languages understood by SIM. The users, however, are not required to learn these languages and can instead use menu-based products.

2. The Semantic Data Model

Entities, relationships between entities, abstraction mechanisms and integrity constraints are the generally acknowledged key features of semantic models [Date83,HaMc81,KiMc84,SmSm77,TsLo82]. An entity is an abstract object that corresponds to some real or conceptual object in an application environment. A semantically meaningful collection of entities forms a *class*. Entities do not exist in isolation -- they are related to each other in various ways and the notion of *attribute* describes this relationship between entities. An attribute of an entity defines how it is related to other entities of another or perhaps the same class. Entities are represented by system-defined identifiers and their existence does not depend on any of their attributes. Attributes of an entity are said to be displayable if their range is one of a number of special, system-defined classes (for example, the class of all strings). Attributes can be single-valued or multi-valued.

Abstraction mechanisms allow complex information to be categorized and viewed in comprehensible ways. Classification, aggregation and generalization are the abstraction mechanisms normally used in semantic models. Classification represents a member/class relationship. Aggregation is a primitive that allows the relationship between entities to be treated by itself as an entity at a higher level [SmSm77]. Generalization allows each member of a class to be related to a member of a more generic class, called its *superclass*. The notion of generalization can be applied successively, yielding a hierarchy of classes.

Data types, attribute options and assertion predicates are the principal techniques used for constraint specification in semantic data models. Semantic models provide strong typing features that can be used in a natural way to constrain the values of an attribute. Strong typing also discourages users from making meaningless associations between components of data. Attribute options like unique, required, distinct and maximum and minimum cardinality are used to specify the structural integrity of data. These options are sufficient to describe the usual 1:1, 1:many and many:many relationships. Assertion predicates specify conditions that are to be satisfied by entities of a class. They are either stated as predicates that hold on the database at all times or as predicates tested after particular DML actions are executed. Assertions based on transitions [Date83] are also allowed.

3. Schema Definition in SIM

Most work on semantic data models has been concentrated on its utility as a logical database design tool. While some prototype database systems that implement a selected set of features of semantic models exist [GGKZ85,TsZa84,MBW80,Ship81, BKK87], to the best of our knowledge, SIM is one of the first large scale, fully featured commercial implementations.

3.1 Classes

The primary unit of data encapsulation in SIM is a *class*, which represents a meaningful collection of entities. A class is either a *base class* or a *subclass*. A base class is defined independently of all other classes in the database, while a subclass is defined based on one or more classes, called its *superclasses*. In the example schema, PERSON, COURSE and DEPARTMENT are base classes, STUDENT and INSTRUCTOR are subclasses whose superclass is PERSON and TEACHING-ASSISTANT is a subclass whose superclasses are STUDENT and INSTRUCTOR. In this paper we will use the unqualified term class in any context where either a base class or a subclass is applicable. Interclass connections are usually represented as a directed graph whose nodes are the classes and whose edges denote superclass-to-subclass connections. SIM requires that this graph be acyclic and the set of ancestors of any node contain at most one base class.

Every base class has a special system-maintained attribute called its *surrogate*. All subclasses eventually derived from a base class inherit its surrogate attribute. The surrogate value for every entity in a class must be unique, must not be null and cannot be changed once defined. In SIM, surrogates play a central role in the implementation of generalization hierarchies and entity relationships.

3.2 Attributes

In SIM, a distinction is made between *data-valued attributes* (DVA) and *entity-valued attributes* (EVA). A DVA describes a property of each entity in a class

by associating the entity with a value or a (multi)set of values from a domain of values. Definition of DVAs in SIM and *attributes* in the E-R model [Chen76] are similar. NAME and BIRTHDATE of the PERSON class are examples of DVAs. An EVA, on the other hand, describes a property of each entity of a class by relating it to an entity or entities of another or perhaps the same class. An EVA represents a binary relationship between the class that owns it (domain) and the class it points to (range). In the example schema, ADVISOR is an EVA of STUDENT whose value is an INSTRUCTOR. SIM automatically maintains the *inverse* of every declared EVA and guarantees that an EVA and its inverse will stay synchronized at all times. An inverse can also be explicitly named by the user. For example, ADVISEES is the inverse of ADVISOR. In DML, the term INVERSE(ADVISOR) can be used in any context where ADVISEES is allowed.

A purist can avoid the distinction between DVAs and EVAs by assuming standard base classes of integers, strings, etc. We could get rid of explicit type declarations from the model by requiring that they be declared as (pre-enumerated) subclasses. For example, the ID-NUMBER type can be represented by a subclass of the INTEGER base class with appropriate range conditions. While a purely functional definition of all attributes [HaMc81,Ship81] is aesthetically pleasing, we have observed that many users have difficulty understanding it. Explicit data types in SIM are more naturally imported into its host language interface programs. We feel that our approach is intuitive and more readily understood and we chose it because of historical considerations.

A subclass *inherits* all the attributes of all its ancestor classes in its generalization hierarchy. In the example schema, attributes of PERSON are to be seen as an integral part of STUDENT since every student must be a person. An attribute is said to be an *immediate* attribute of the base class or subclass it is declared in. In DML, an inherited attribute of a subclass can be used in any context where an immediate attribute is allowed and vice versa.

In SIM, every class that has subclasses must have a special attribute of *subrole* type declared with it (for example, PROFESSION of PERSON). A subrole is a special case of enumerated types and its value set must contain the names of all the immediate subclasses of the class in which it is used. Subrole attributes are system-maintained and can only be read. They can be included in the target list of a retrieve query and provide a convenient method to retrieve symbolically all the roles an entity participates in.

3.2.1 Attribute Options

REQUIRED, UNIQUE, MV, DISTINCT and MAX are the attribute options supported in SIM. REQUIRED implies that the value of an attribute cannot be null (a null is used to represent both "unknown" and "inapplicable" values [Date83]). UNIQUE implies that no two entities of the class can have a value in common. Null values are omitted from uniqueness considerations.

MV indicates that an attribute is multi-valued. By default, attributes are single-valued. The DISTINCT option on a multi-valued attribute implies a set of values as opposed to a multiset. MAX limits the number of values an MV attribute can take, which by default is unbounded.

When specified appropriately on EVAs and their inverses, attribute options define the structural properties of data. Let E1 be an EVA and INV-E1 its inverse in some schema. If both E1 and INV-E1 are single-valued, they define a 1:1 relationship. If E1 is multi-valued and INV-E1 is single-valued, they define a 1:many or many:1 relationship, depending on the point of view. If both are multi-valued, they define a many:many relationship. Partial, total, or absence of dependency on the relationship can be defined by specifying the Required option appropriately on E1 and INV-E1. Cardinality can be controlled by the MAX option. In the example schema, SPOUSE is a 1:1 relationship, ADVISOR:ADVISEES defines a many:1 relationship between STUDENT and INSTRUCTOR with a limit of 10 advisees per instructor, and COURSES-ENROLLED: STUDENTS-ENROLLED defines a many:many relationship between STUDENT and COURSE.

3.3 Integrity Constraints

As mentioned before, the structural integrity of data in SIM is defined by judicious use of generalization and attribute options. Since all relationships are maintained by the system, SIM can guarantee full referential integrity and "dangling reference" problems do not exist. SIM also allows the specification of integrity conditions with any class. An integrity condition can be any arbitrary DML selection expression with the class as perspective and may even include quantifiers and aggregate functions. V1 and V2 are examples of integrity conditions in the example schema. Based on the terms of the integrity condition, SIM will determine all possible events that may cause this condition to be violated and will make sure it does not happen. Integrity constraints are handled by a trigger detection / query enhancement mechanism that works efficiently for a subset of constraints. In its most general form, maintaining

integrity constraints not associated with explicit user-defined trigger events is an extremely difficult problem to solve efficiently, and we are experimenting with several algorithms. Currently, arbitrary integrity constraints have only been partially implemented.

4. Data Manipulation in SIM

SIM DML is a high-level, non-procedural language designed with a particular emphasis on its naturalness and ease of use. Constructs of the DML are a direct consequence of the features of the semantic model. It incorporates many ideas from GORDAS [ElWi81] and DAPLEX [Ship81].

4.1 Perspective Class

The notion of *perspective class* is of fundamental importance in SIM DML. It is based on the assumption that when formulating a query (either Retrieve or update), a user is primarily interested in entities of one class, called the perspective. Other classes in the database are viewed based on their relationship to the perspective class. *Qualification* is a syntactic process that relates attributes of various classes in the database to the perspective of a query. For example, if STUDENT is the perspective class of a query, STUDENT-NO OF STUDENT refers to an immediate attribute, NAME OF STUDENT refers to an inherited attribute (from PERSON), NAME OF ADVISOR OF STUDENT refers to the name of his advisor, TEACHERS OF COURSES-ENROLLED OF STUDENT refers to the instructors who teach the courses he is enrolled in. Within the context of a DML query, attributes derived indirectly from the perspective by more than one level of qualification are called its *extended* attributes.

The request "print the name of each student and the name of his advisor, if any" is expressed in DML as

From Student Retrieve Name, Name of Advisor.

Names of persons who are not students will not be printed. However, if a student does not have an advisor, SIM will still select and print his name with a null value for the advisor's name. The perspective of this query is STUDENT and NAME OF ADVISOR refers to an extended attribute. Derivation of values for extended attributes corresponds to the notion of *directed outer join* [Codd79] in relational systems and is a natural and direct result of the notion of perspective.

Sometimes it may be necessary to form queries with more than one perspective class (for example, retrieve all pairs of students who are taking the same set of courses). SIM provides such a facility and queries so formulated are called multi-perspective queries. Multiple perspective classes are related to each other by *value-based joins*, which establish dynamic relationships between classes. We strongly recommend the use of EVAs over value-based joins since they represent a static, schema-defined, efficient and natural way of establishing relationships.

4.2 Qualification

As mentioned before, qualification is a syntactic process by which an attribute is connected to a perspective class. Qualification of an attribute is usually of the form

<attr name> {OF <eva name> [AS <classname>]}
OF <perspective class name> [AS <class name>].

<attr name> can either be a DVA or an EVA. The 'AS' clause specifies role conversion from a class to another class in the same generalization hierarchy. It is normally used for converting the role of an entity from a superclass to a subclass. The following are examples of qualification from STUDENT:

Title of Courses-Enrolled of Student,
Teaching-Load of Student as Teaching-Assistant,
Student-No of Spouse as Student of Student.

It is not necessary to qualify every attribute in a DML query to its perspective. Qualification can be cut short at any stage where the context is sufficient for the system Parser to complete it unambiguously. For example, if STUDENT is the perspective,

Name of Advisor of Student,
Salary of Advisor of Student and
Name of Advisor, Salary

will yield identical results. Qualifications of multiple target list items can also be parenthetically factored for syntactic convenience.

4.3 Syntax of Retrieve Queries

A DML Retrieve query is expressed in the form

[FROM <perspective class list>]
RETRIEVE [TABLE [DISTINCT] ! STRUCTURE]
<target list>
[ORDER BY <order list>]
[WHERE <selection expression>].

<perspective class list> is the list of perspective classes for a query with optional associated reference variables. <target list> and <order list> are a list of expressions made up of constants, immediate, inherited and extended attributes of the perspectives,

and aggregate and other functions applied on such attributes.

4.4 Binding and Range Variables

The semantics of SIM queries are understood in terms of nested iterative loops similar to DAPLEX [Ship81,DGK82]. All occurrences of a perspective class name in a query are "bound" to one range (loop) variable. Similarly, all occurrences of an identically qualified EVA or multi-valued DVA are also bound to one range variable. Consider the following query:

Retrieve Name of Student,
  Title of Courses-Enrolled of Student,
  Credits of Courses-Enrolled of Student,
  Name of Teachers of Courses-Enrolled of Student
Where Soc-Sec-No of Student = 456887766.

For the student with soc-sec-no 456887766, this query will print his name and for each course that he is taking, its title, credits and the name of the instructors who teach it. This is possible only because all five occurrences of the literal "STUDENT" and all three occurrences of the literal "COURSES-ENROLLED" are bound to their respective range variables.

Implicit binding of names is broken in a few special constructs such as aggregate functions, transitive closure or quantifiers. Named range variables can also be explicitly established on a class, EVA or a multi-valued DVA for subsequent discriminating use.

4.5 Semantics of Retrieve Queries

Qualification and binding rules of SIM, taken together, give rise to the concept of a *query tree*. Assume for the moment that queries are allowed to have only one perspective class. We can construct a tree QT such that its nodes, $X_1,X_2...X_n$, represent implicit or explicit range variables and its edges represent EVAs or multi-valued DVAs. The root of this tree (X_1) is the range variable of the perspective class. Qualification of every attribute in a query results in it being associated with one range variable in QT. Label each variable X_i in QT as follows: label it "TYPE 3" if it and all its descendants are used only in the target list of the query and not in its selection expression; label it "TYPE 2" if it and all its descendants are used only in the selection expression of the query and not in its target list; label it "TYPE 1" otherwise. X_1 is always labeled TYPE 1. Without loss of generality, we can assume that $X_1,X_2..X_m$, m < = n, are either TYPE 1 or TYPE 3 nodes in the depth-first order of their appearance in QT and $X_{m+1}...X_n$, are the TYPE 2 nodes in the depth-first order of their appearance. Let domain(X_i) denote the entities or values X_i ranges over. Entities of the perspective class constitute domain(X_1) while every other domain is defined based on an attribute and a given instance of the range variable of its parent node. To make our definitions simpler, we will assume that the domain of TYPE 3 variables will never be empty (when empty, adding a dummy instance all of whose attributes are null will achieve this). Semantics of a DML query are defined based on its QT by the following program (DAPLEX notation):

```
for each X1 in domain(X1)
 for each X2 in domain(X2)
       ...
  for each Xm in domain(Xm)
       such that
   for some Xm+1 in domain(Xm+1)
         ...
    for some Xn in domain(Xn)
     if <selection expression> is true then
      print <target list>
    end for;
   end for;
  end for;
 end for;
end for;
```

Note that the order in which these loops are nested prescribes the order in which the output data is returned. Such an ordering is a direct consequence of the notion of perspective. The output of the program above is termed "fully tabular", in which one format describes every output record. SIM provides other forms of output that impose additional structuring on the output. They provide multiple record formats, and every output record is described by one of these formats. In the "fully structured" case, the number of different output formats is equal to the count of TYPE 1 and TYPE 3 variables in the query. Such forms of output are particularly useful in the host language interfaces to SIM (the details are omitted from here because of space considerations).

Multiple perspective classes can be handled as a cross product with minor extensions to the program above.

4.6 Aggregate Functions

In SIM, aggregate functions are specified naturally by delimiting their scope in a qualification. Examples:

AVG(Salary of Instructor),
AVG(Salary of Instructors-employed) of Department,
COUNT(Teachers of Courses-enrolled) of Student.

The first gives the average salary of all instructors in the database, the second gives the average salary of instructors employed by each department (it's a dynamically derived attribute of department) and the third gives, for each student, the count of teachers of all the courses he is enrolled in. Quantifiers (all, some and no) follow a similar syntax.

4.7 Transitive Closure

The transitive closure operation is expressed in syntax similar to that of aggregate functions. The following query will retrieve all the prerequisites of Calculus I:

Retrieve Title of Transitive(prerequisite) of Course
Where Title of Course = "Calculus I".

The tree structure of a transitive closure will be preserved in a fully structured output, based on the notion of *level numbers* for output records. Transitive closure can be performed on any cyclic chain of EVAs (the single reflexive EVA in the example above is a cyclic chain one element long).

4.8 Update Statements

An Insert in SIM is of the form:

INSERT <class name1>
[FROM <class name2> WHERE <boolean expn>]
[(<assignment list>)]

If the FROM clause is omitted, all superclass roles of <class name1> up to and including the root of the hierarchy will be inserted along with <class name1>. If a FROM clause is specified, <class name2> must be an ancestor of <class name1> in the hierarchy and all superclass roles of <class name1> up to but not including <class name2> will be automatically inserted as needed. The boolean expression selects the entity whose role is being extended. Immediate attributes of all inserted classes can be assigned values in one INSERT statement.

A Modify in SIM is of the form:

MODIFY <class name> (<assignment list>)
WHERE <boolean expn>.

All immediate and inherited attributes of <class name> can be modified in one statement.

Keywords INCLUDE and EXCLUDE define corresponding operations on multi-valued attributes. EVA assignment is particularly simple:

<eva name> := [INCLUDE ! EXCLUDE]
<object name> WITH (<boolean expn>).

<object name> refers to a class name for single-valued EVA assignments and multi-valued EVA inclusions. It refers to the same EVA name for exclusions. If a class name is used, it must be the range class of the EVA.

A delete statement is of the form

DELETE <class name> WHERE <boolean expn>.

When an entity is deleted, all its subclass roles will be deleted, while its superclass roles will remain unaffected. For example, if an entity of STUDENT is deleted, it will continue to exist in class PERSON. However, if an entity of PERSON is deleted, it will also be deleted from STUDENT, INSTRUCTOR and TEACHING-ASSISTANT classes (if present). When an entity of a class or subclass is deleted, its immediate EVAs, if any, will be automatically deleted.

4.9 Miscellaneous

The DML also supports quantifiers, pattern matching and an array of operators and primitive functions. Null values are treated uniformly in expression evaluation, and SIM follows the 3-valued logic. Examples below illustrate the power and ease of use of SIM DML.

1. Insert John Doe as a STUDENT and enroll him in Algebra I.

```
Insert student(name := "John Doe",
 soc-sec-no := 456887766,
 courses-enrolled:=course with (title="Algebra I")).
```

2. Make John Doe an Instructor too.

```
Insert instructor
From person Where name = "John Doe"
(employee-nbr := 1729).
```

3. Let John Doe drop Algebra I and let Joe Bloke be his advisor.

```
Modify student (
  courses-enrolled := exclude courses-enrolled
                with (title = "Algebra I"),
  advisor := instructor with (name = "Joe Bloke"))
Where name of student = "John Doe"
```

4. If an instructor teaches more than 3 courses and advises students from other departments, give him a 10% raise.

```
Modify instructor( salary := 1.1 * salary)
Where count(courses-taught) of instructor > 3 and
    assigned-department neq
      some(major-department of advisees).
```

5. Find the minimum number of courses that must be completed before one enrolls in Quantum Chromodynamics.

```
From course
Retrieve count distinct (transitive(prerequisite))
Where title = "Quantum Chromodynamics".
```

6. Print the name of each instructor who advises some student from the Physics department and the courses he teaches, if any.

```
Retrieve name of instructor, title of courses-taught
Where name of major-department of advisees =
                              "Physics".
```

7. Print student, instructor pairs where the student is older than the instructor and the instructor is not a teaching assistant and is not the student's advisor.

```
From student, instructor
Retrieve name of student, name of Instructor
Where birthdate of student <
  birthdate of instructor and
  advisor of student NEQ instructor and
  not instructor isa teaching-assistant.
```

5. Implementation Considerations

SIM has been implemented on Unisys A series machines and its implementation goals are DMSII evolution, heterogeneous data access and performance.

A utility program allows any existing DMSII database to be viewed as a SIM database. Semantics of data not readily apparent from its DMSII description can be made known to SIM by the user. For example, a foreign-key based relationship between DMSII structures can be defined as a SIM EVA.

SIM's architecture has been designed to be flexible enough to accommodate a variety of sources and types of data, including files, transitory data such as from process interfaces and foreign databases based on relational, network or other models. The utility of semantic models in this context has been pointed out before [SBDG81].

SIM is capable of supporting commercial application systems that span a wide range, including systems that require very high transaction processing rates.

5.1 Architecture

The goals mentioned above have led to a highly modular architecture for implementation with well-defined, formal interfaces between modules. Query Driver, Parser/Optimizer, Directory (catalog) Manager and LUC Mapper are the modules comprising SIM (see Figure 1). The runtime (process) architecture, dynamic code generation for queries and binding have been designed to take maximum advantage of the features of the stack architecture of the A series machines and the inter-process communication they support. These details are omitted from here for lack of space.

The LUC Mapper is a key module of SIM's implementation. It extends the capabilities of any underlying physical or logical data source and presents a uniform, simplified view of data and operations associated with it. The objects supported by the Mapper are LUCs (Logical Underlying Components), relationships between LUCs and integrity constraints. A LUC is a collection of records all of whose fields are single-valued. Relationships between LUCs come in three flavors, based on the SIM objects they represent: class-subclass links (always 1:1), Multi-valued DVAs (1:many between an independent LUC and a dependent LUC) and EVAs (1:1, 1:many or many:many between two independent LUCs). Every SIM schema has a standard translation into a LUC schema with a LUC for every class, subclass and multi-valued DVA. A cursor can be opened on a LUC or on a relationship and it delivers one record of the LUC at a time. Relationship cursors deliver one record of the range LUC and the Mapper assumes the responsibility of traversing a relationship, no matter how it is physically mapped. The Mapper assures the structural integrity of data reflected in LUC interconnections. For example, when a record of a superclass LUC is deleted, the Mapper will automatically delete corresponding subclass records and delete instances of all EVAs the deleted records participate in. Structural integrity is maintained by the Mapper for performance reasons. For example, if class and subclass records are mapped into one physical record, the Mapper will perform one delete instead of the two operations that may be needed otherwise. Integrity constraints specified by the user as ASSERTs are handled by the Parser/Optimizer using query augmentation techniques.

SIM optimizes a query by building a query graph (whose nodes are LUC objects), enumerating strategies, estimating the cost of processing for each strategy and choosing the one with the least cost. We have extended relational query optimization techniques to handle generalization hierarchies, EVAs and the perspective-oriented ordering and duplicate value semantics [DGK82] implied by the DML. For example, when listing students and their courses, DML implies an implicit ordering of output based on student surrogates. Transformation of a query graph for a strategy is tested to see if it is *semantics-preserving*, and, if it is not, the cost of

reordering/sorting output is added to the cost of a strategy. Cardinality of LUCs and relationships, blocking factors, indexes and the cost of accessing the first and subsequent instances of a relationship are some of the optimization parameters used. This technique enables the Optimizer to do its job without considering physical mapping details. For example, the I/O cost of accessing the first instance of a relationship will be 0 if the relationship is implemented by clustering and 1 block access if it is implemented by absolute addresses (pointers). Statistical optimization is not fully implemented yet.

5.2 Physical Mapping Options

The high-level objects of the model must be mapped into record-based units for physical storage. SIM uses a carefully balanced set of rules to determine the mapping. The user can override the default and choose any access method or mapping supported by the underlying system. The default mappings are described below.

LUCs in a tree structured generalization hierarchy are physically mapped into a storage unit with variable-format records based on record types. The number of record types needed will be equal to the number of nodes in the tree. This ensures that all immediate and inherited single-valued DVAs applicable to a class will be in one physical record. It is also efficient in terms of space. A class defined as the subclass of two or more immediate superclasses is mapped into a separate storage unit with 1:1 subclass links connecting it to its parent LUCs.

LUCs of multi-valued DVAs without the MAX option (unbounded) are mapped into a separate storage unit. Those with the MAX option are stored as arrays in the same physical record with their owner.

1:1 EVAs are mapped based on foreign-keys. Many:many EVAs without the Distinct option and 1:many EVAs are mapped into a storage unit termed the Common EVA Structure. This structure has records of the form <surrogate1> <relationship-id> <surrogate2>. The surrogates can be direct keys (record number), random keys (based on hashing) or index sequential keys. Every many:many EVA with the Distinct option gets a separate structure like the one described above. The default for 1:many EVAs was chosen to avoid the additional index structure that will be needed with a foreign-key based mapping. There are a variety of ways in which EVAs can be mapped, including absolute addresses and embedded structures. The mapping of EVAs is the key factor in determining SIM's performance.

User-declared attributes which are Unique and Required can be defined to be the surrogate of a class. By default, the system will create its own surrogate attribute.

6. Conclusion and Future Developments

We have described SIM, a database system based on the semantic data model. SIM provides a conceptually natural view of data by moving away from the notational simplicity of modeling with a minimally complete set of constructs. Entities, generalization hierarchies, schema-defined interobject relationships and integrity constraints are the key concepts of the model. The DML of this system is designed to take advantage of and directly support these features. The DML notions of perspective and qualification by EVAs are a natural complement to the system's schema definition features.

Our experience with a large number of test databases is a testimony to the power and utility of the concepts mentioned before. The stand-alone data dictionary ADDS is itself a SIM database. It consists of 13 base classes, 209 subclasses, 39 EVA-inverse pairs, 530 DVAs and at its deepest, one hierarchy represents 5 levels of generalization.

We are currently working on several extensions to the model. Work under progress includes the design of a view mechanism, derived attributes, system-maintained ordering of classes and EVAs, temporal data, efficient algorithms for various categories of integrity constraints and experiments in quantifying the naturalness and ease of use of DDL and DML concepts.

7. Example Schema

```
(* The schema diagram is in Figure 2. *)
Type degree = symbolic (BS, MBA, MS, PHD);
Type id-number = integer (1001..39999,60001..99999);

Class Person (
  name: string[30];
  soc-sec-no: integer, unique, required;
  birthdate: date;
  spouse: person inverse is spouse;
  profession: subrole (student,instructor) mv );

Subclass Student of Person (
  student-nbr: id-number;
  advisor: instructor inverse is advisees;
  instructor-status: subrole(teaching-assistant);
  courses-enrolled: course inverse is
    students-enrolled mv (distinct);
  major-department: department );

Verify v1 on Student
  assert sum(credits of courses-enrolled) > = 12
  else "student is taking too few credits";
```

```
Subclass Instructor of Person (
 employee-nbr: id-number unique required;
 salary: number[9,2];
 bonus: number[9,2];
 student-status: subrole(teaching-assistant);
 advisees: student inverse is advisor mv (max 10);
 courses-taught: course inverse is
   teachers mv (max 3,distinct);
 assigned-department: department inverse is
   instructors-employed );

Verify v2 on instructor
 assert salary + bonus < 100000
 else "instructor makes too much money";

Subclass Teaching-assistant of Student and Instructor(
 teaching load: integer (1..20) );

Class Course (
 course-no: integer (1..9999) unique required;
 title: string[30] required;
 credits: integer (1..15) required;
 students-enrolled: student inverse is
   courses-enrolled mv;
 teachers: instructor inverse is
   courses-taught mv (max 7);
 prerequisites: course inverse is prerequisite-of mv;
 prerequisite-of: course inverse is prerequisites mv );

Class Department (
 dept-nbr: integer(100..999) required unique;
 name: string[30] required;
 instructors-employed: instructor inverse is
   assigned-department mv;
 courses-offered: course mv );
```

References

[Abri74] J.R Abrial. Data Semantics. In Database Management, J. Klimbie and K. Koffeman Eds. North-Holland Amsterdam 1974.

[BKK88] J. Banerjee, W. Kim and K. Kim. Queries in Object-Oriented Databases. In Proc. IEEE Intl. Conf. on Data Engg.. February 88.

[Chen76] P.P.S Chen. The entity-relationship Model: Toward a unified view of data. ACM Trans. Database Syst. 1,1 March 76.

[Codd79] E.F. Codd. Extending the Database Relational Model to capture more meaning. ACM Trans. Database Syst. 4,4 Dec 79.

[Date83] C.J. Date. An Introduction to Database Systems, Volume 2. Addison-Wesley 83.

[DGK82] U. Dayal, N. Goodman, R.H. Katz. An Extended Relational Algebra with Control over Duplicate Elimination. In Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems. March 82.

[ElWi81] R. El Masri, G. Wiederhold. GORDAS: A Formal High-Level Query Language for the Entity-Relationship Model. In Proc. 2nd Intl. Conference on Entity-Relationship Approach. October 81.

[Fish87] D. H. Fishman et al. Iris: An Object-Oriented Database System. ACM Trans. Office Info. Systems. Vol 5, No 1. January 87.

[GGKZ85] K. Goldman, S. Goldman, P. Kanellakis, S. Zdonik. ISIS: Interface for a Semantic Information System. In Proc. ACM SIGMOD Intl. Conference on Management of Data. May 85.

[HaMc81] M. Hammer, D. McLeod. Database Description with SDM: A Semantic Data Model. ACM Trans. Database Syst. 6,3 Sept 81.

[Kent79] W. Kent. Limitations of Record-Based Information Models. ACM Trans. Database Syst. 4,1 March 79.

[KiMc84] R. King, D. McLeod. Semantic Database Models. In S.B. Yao (Ed). Principles of Database Design. Prentice Hall 84.

[MBW80] J. Mylopoulos, P.A. Bernstein, H.K.T Wong. A Language Facility for Designing Database-Intensive Applications. ACM Trans. Database Syst. 5,2 June 80.

[Ship81] D.W. Shipman. The Functional Data model and the Data Language DAPLEX. ACM Trans. Database Syst. 6, 1 March 81.

[SBDG81] J. Smith, P. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong. Multibase - Integrating Heterogeneous Distributed Database Systems. In Proc. AFIPS NCC 81.

[SmSm77] J.M. Smith, C.P. Smith. Database Abstractions: Aggregation and Generalization. ACM Trans. Database Syst. 2,2 June 77.

[TsLo82] D.C. Tsichritis, F.H. Lochovsky. Data Models. Prentice Hall 1982.

[TsZa84] S. Tsur, C. Zaniolo. In Implementation of GEM - supporting a semantic data model on a relational back-end. In Proc. ACM SIGMOD Intl. Conference on Management of Data. May 84.

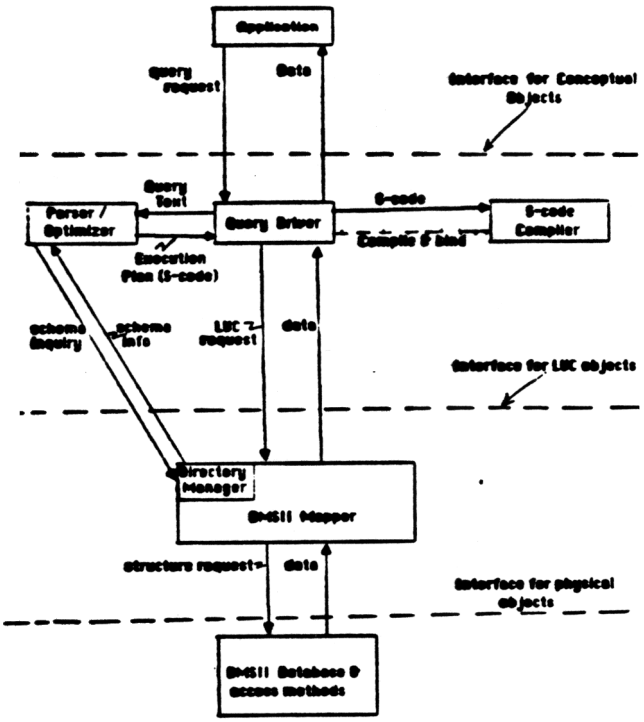

Figure 1. Architecture of SIM

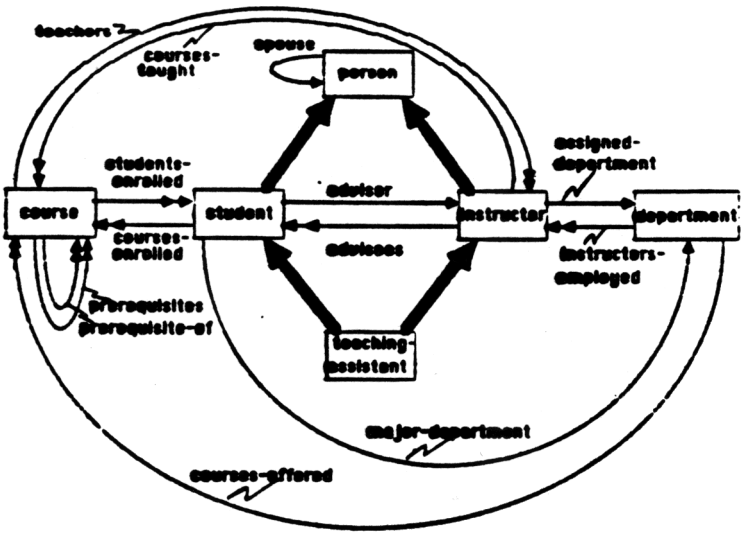

Figure 2. UNIVERSITY database schema