File SIM

SIM

An Implementation of a Semantic, Object-Oriented Data Model

By Chris Britton

The purpose of this article is to communicate the concepts underpinning the Semantic Information Manager (SIM) database system, which has been implemented on the A Series.

SIM implements a semantic data model. The word "a," not "the," is used because there are many candidates but no standard semantic database system. SIM most closely resembles Semantic Data Model (SDM) of Hammer and McLeod (ref. 4). In fact, it may be that semantic database systems are going to become a subset of the more general class of object-oriented database systems. We will discuss that point later.

Most of our discussion here describes the SIM model. It is preceded by a discussion on general issues, in which we will assume there is a basic understanding of what a data model is (as used by database specialists, not application designers) and at least some knowledge of the relational data model.

SIM's Purposes

Research for the SIM project started in 1982. Detailed design was done throughout 1984 and programming started in earnest in 1985. The first release was included in the A Series Release 3.7 in late 1987.

When SIM was started, it was evident to many that the next generation of systems would be relational or relational-like systems. Since the A Series then did not have a relational database system, and it was impossible to turn the existing DMSII database system into a relational system, it was clear something new had to be implemented. There was a desire not to do a "me-too" relational system. The semantic data model was chosen because it looked like a potential successor to the relational data model and because it was expected to be easier to interface to existing DMSII databases, which can have semantics that are not easily mapped int. relational systems. There was also potential for improvement because the system knows more about the application. SIM was an attempt to not only catch up with the competition, but to overtake it.

The relational database concept was introduced in 1970 (ref. 1). At first, most serious criticisms were practical. "It's impossible to implement" was a common cry. From early on, there were also criticisms about its lack of semantics. The relational model's greatest strength and greatest weakness is that there is only one modeling concept — a table (or, more technically, a relation).

But a breakthrough had been made without worrying about the details of the implementation, it was now valid to ask the question, "What is the best way of expressing the data semantics?"

By the end of the 1970s there was a rash of alternative data models; the entity-relationship model, the binary relation model, the functional data model and so forth.

Some order was restored in the late '70s when Smith and Smith (ref. 5) introduced the three abstractions: Aggregation, Classification and Generalization. When I think of abstraction, I envision all the data decomposed into the smallest units and then ask the question, "How do I group the data into objects that I can understand?"

Aggregation is the name for objects grouped into a larger object using the "part of" relationship. The most common instance of aggregation is the relationships of attributes to entities — attributes are part of entities.

Classification is the "instance of" relationship and is most typically used to group like entities into a class; thus, entities are an instance of a class.

Generalization recognizes that one object can be a member of several different classes. For instance, a member of the class "employee" is also a member of the class "person." Generalization implements the "is a" relationship. The idea of abstraction is the conceptual basis on which semantic databases are founded.

It is not the intention of this article to describe either the history, the theory or the available semantic data models in theory. There are at least two major reviews of the area in the literature (refs. 2 and 3). The importance of some level of semantics, however, is clear from the work done on application database design. Almost every design methodology in existence uses some kind of higher level data model, usually the entity-relationship model.

When the A Series project started, work on semantic data models was well advanced and many thought that the semantic database would begin to displace relational databases by the end of the decade. This has not happened. In fact, the move from network and hierarchical database systems to relational database systems has been much slower than predicted.

A Look at SIM

Now, let's take a closer look at each aspect of the SIM data model — including both data structure and data manipulation features — and how they compare with the relational database equivalent.

Class, Entity, Attribute

An entity is an object of interest. The words entity and object are used synonymously in the industry but object is more fashionable. A class is a set of like entities. Thus, an individual man is an object while the set of men and women is a class. Objects can be real or abstract; if there is a noun that describes it, it is an object.

The properties of an object are called attributes. A Person's name, an Order's amount, a Department's location are all attributes of their respective objects. Attributes have a type that indicates the kind of data value it can contain. The allowable SIM types are Numeric, Integer, Real (as in Pascal), String (of alphanumeric characters), Boolean (as in Pascal), Symbolic (like Pascal-enumerated types), date, time, compound (e.g., a name is a Christian name, a middle initial and a surname), and user-defined.

Attributes can be single-valued or multi-valued. Single-valued data attributes are by far the most common, but multi-valued data attributes do exist. For instance, a person's phone number attribute might be multi-valued, indicating that he or she has several phone numbers.

Integrity options can be associated with attributes. You can force the user to put a value in the attribute by specifying that it is Required. You can ensure that every object in the class has a different value for an attribute by specifying it as Unique. You can restrict the values allowed for the attribute by specifying a range clause. Thus, we can define a user-defined type called weekdays that is a subtype of a symbolic type (days-of-the-week) but with a range restricted to Monday through Friday.

FIGURE 1

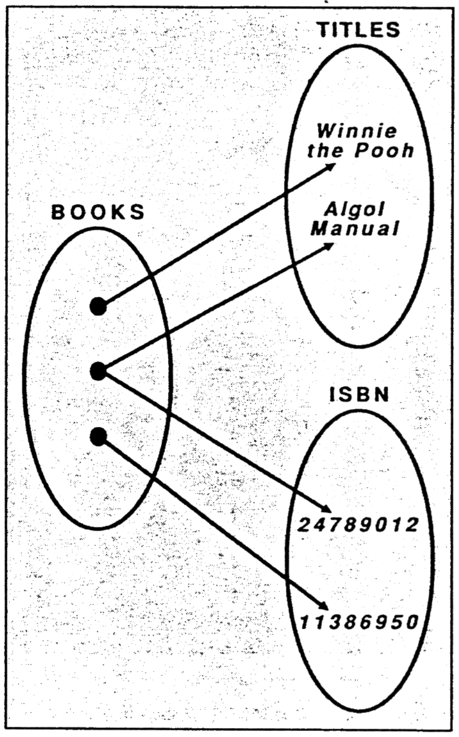


Figure 1 illustrates the relationship between entities and attributes. This relationship is intentionally shown in a way that is not tabular.

All access to the database is done by using Data Manipulation Language (DML), though the underlying use of DML is by default disguised from the end-user. This non-procedural language has been specially designed to use the features of the model. Some DML examples using classes and attributes follow.

1. Retrieve name and age of borrowers older than 70:
   From Borrowers
   Retrieve name, age
   Where age > 70
2. Retrieve borrowers whose hair color is the same as a color of a book and return the borrower's name and the book's color:
   From Books, Borrowers
   Retrieve color of Books,
   name of Borrowers
   Where color of Books
   = hair_color of Borrowers
3. Example of an insert into the book class:
   Insert Books
     (title := "Spot's Big Book of Words";
     ISBN := 24789012;
     author := "Eric Hill"
)
4. Update all books that have the text 'Spot' somewhere in the title and mark them as books for infants:
   Update Books
     (age_group := Infant
)
   Where title ISIN ? 'Spot'?
5. Delete up to 10 books whose International Book Serial Number (ISBN) is less than 100000:
   Delete Books [limit = 10]
   Where ISBN < 100000

What we have seen so far is similar to a relational database. One can draw a direct parallel between tables and classes, rows and entities, columns and attributes, and domains and type. The major difference, as we will see later, is that entities can be vastly more complex than a simple table row. The chief differences already evident are that there is no concept of primary key and that multi-valued attributes are allowed.

Primary keys in the relational database have two functions; they represent the entity in foreign keys and they have data values that are important in their own right. The first of these functions is irrelevant in SIM, hence there is no difference between a primary key and any other unique data attribute. As a consequence, every data value can be updated without regard to whether it impacts a relationship. Relational database designers have realized the importance of this feature and many (including E.F. Codd) advise using surrogates (i.e., system-maintained primary keys). In SIM, a surrogate exists but is hidden from the user.

Relationships

Relationships in SIM are implemented as attributes. It is as if the value of the attribute is not a data value but a reference to another entity. Thus, a class of Books might be specified as having a title of type string and a borrowed-by of type Borrower.

Class Books
  (title : string (50);
  borrowed-by : Borrower)

These attributes, like borrowed-by in the above example, are rather clumsily called "Entity-Valued Attributes" (to distinguish them from normal attributes, which are "data-valued attributes") or EVAs for short. This is illustrated in Figure 2, which shows the evolution from a data attribute — a book's author in this instance — to an EVA.

An EVA can point to any entity — even one in its own class. For instance:

```
Class Person
   (Spouse: Person)
```

Like data attributes, EVAs can be single-valued or multi-valued. Clearly, an EVA in one direction might simply be the opposite of an EVA going in the opposite direction. This can be denoted by defining the EVA as an inverse. Hence, we might have an inverse of the previously illustrated borrowed-by EVA in the Books class:

```
Class Borrower
   (books-out : Books, multi-valued,
                Inverse of borrowed-by)
```

Putting these features together, we can see how we can define one-to-one, one-to-many or many-to-many relationships.

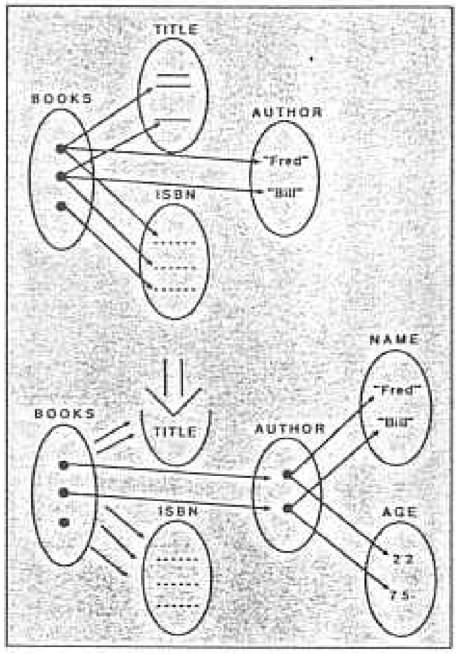
1. One-to-one relationship:
```
Class A
   (to_B : B, SV)
Class B
   (to_A : A, SV Inverse to_B)
```
2. One-to-many relationship:
```
Class A
   (to_B : B, MV)
Class B
   (to_A : A, SV Inverse to_B)
```
3. One-to-many relationship:
```
Class A
   (to_B : B, MV)
Class B
   (to_A : A, MV Inverse to_B)
```

Like data attributes, EVAs can be required, which means that the entity being referenced cannot be deleted. Also, like data attributes, multi-valued EVAs can have a maximum clause specified to restrict the size of the set.

The DML makes full use of the existence of EVAs, as the following examples show.

1. Retrieve the books with ISBN's greater than 100000 that are being borrowed by someone older than 70. Return the book's title and ISBN and the name and age and name of the spouse of the borrower:
```
From Books
Retrieve title, ISBN,
      (name, age, name of
       spouse) of borrowed-by,
Where ISBN > 100000 and age of
borrowed-by > 70
```
2. Retrieve all borrowers who have currently only borrowed red books and list the book titles:

FIGURE 2



```
From Borrower
Retrieve title of Books-out
Where color of all (Books-out)
= Red
```
3. Retrieve the libraries for whom the average age of their membership is greater than 60. Return the name of the library and the age of the youngest member:
```
From Library
Retrieve name, min (age of
members)
Where avg (age of members) > 60
```
4. Retrieve all the component subjects of computer science and all their components and so forth:
```
From Subject
Retrieve name of transitive
(sub-subject)
Where name = "Computer
Science"
```
(Note: 'transitive' stands for transitive closure.)
5. Update borrowers adding a book with library number 12678 to the list of books-out, and removing a book with library number 3232367 from the books-out list:
```
Update Borrower
 (Books-out := include Books
  with (Library-number
  = 12678);
  Books-out := exclude Books
  with (Library-number
  = 3232367)
 )
```
(Note: inverses are automatically fixed up.)

The way relationships are handled is, of course, completely different from how it is done in the relational model. There are several advantages to the SIM approach.

No primary key problems: As discussed above, all data attributes can be updated without concern for how relationships are impacted. Moreover, the choice of primary key can be delayed in the design process.

Increased semantics: In a relational database of any complexity, it is difficult to see how a table is related to all other tables. In SIM, it is easy because there is an attribute.

Powerful DML: The use of EVAs is equivalent to the relational database outer-directed join, but there are additional features such as quantifiers (all, some and no) and the transitive function that have no direct equivalent.

Many-to-many relationships are supported without an intersection table.

Increased integrity: A multi-valued EVA can be required because referential integrity is built in.

A common reaction to the SIM model is the question, "Why are there no relationship attributes?" The answer is simple. A relationship with attributes is an object of interest and should therefore be a class of objects.

In modeling terms, it is important to realize that a relationship can be both an attribute and an object. This fits in with how we speak about the world. For instance, we say a person has a spouse (i.e., spouse is an attribute of person) and we say two people are married (i.e., marriage is an object).

In the entity-relationship model, relationships are always treated as objects while in SIM relationships they are always treated as attributes. However, with an extension to the SIM model (planned but not implemented), we can look at a relationship both ways. This extension is derived attributes — that is, attributes derived from other information in the database. With this feature, we could specify:

```
Class Lendings — the relationship
                 object —
   (book-lent : Books;
    date-lent  : Date;
    borrower-lent-to : Borrowers)
Class Books
   (lent-to :        Lendings;
    borrowed-by : Borrowers
```

```
                          derived by
                          borrower-lent-to
                          of Lendings )
Class Borrowers
  (books-on-loan  : Lendings multi-
                    valued:
   books-out      : Books derived by
                    book-lent of
                    Books-on-loan )
```

In contrast to SIM, there are problems with the entity-relationship approach:

Relationship objects cannot participate in other relationships. For instance, assume there are several libraries with a pool of books that can be in any one of them. It might then be desirable to add a relationship from a Lendings object to the Library from which the book was borrowed. You cannot do that with the entity-relationship model.

A minor change can lead to a redesign. For instance, lendings would have to be turned into a ternary relationship or an entity.

The semantics of entity and relationship are unclear. It is sometimes hard to decide what is a relationship and what is an entity.

The semantics of having the relationship as an attribute is lost.

There is no way of expressing derived relationships. Note that relationships can be derived using entities just as well as using relationships (e.g., the relationship "products on order" between customer and products can be derived by customer/order plus order/order-line plus order-lines/product).

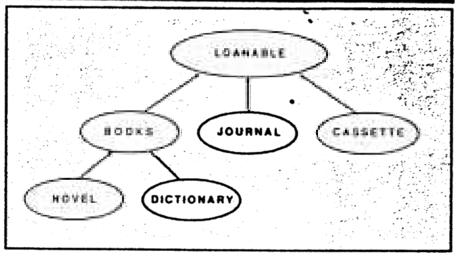
It is more difficult to build a powerful DML using the relationships.

So, in conclusion, the lack of relationship attributes in SIM was deliberate.

Generalization

An entity can belong to many classes. For instance, a library might lend cassettes and journals as well as books. We model this by introducing a new class — let us call it 'Loanables' — and making Books, Cassettes and Journals subclasses of Loanables. This notion of treating several classes as different examples of a more general class is called 'generalization,' and the opposite of generalization (subdividing a class into several classes) is called 'specialization.' Books might be specialized into novels, dictionaries and text books creating a hierarchy of classes. This structure is illustrated in Figure 3. In SIM, all classes underneath the root class are called subclasses. Some of the other semantic data models call them subtypes.

FIGURE 3



A subclass is treated like a class in that it can have attributes of any sort and they can be referenced by EVAs. The main difference is that a subclass inherits the attributes of all its superclasses. Hence, while library-number may be an attribute of Loanables, ISBN may be an attribute of Books and who-did-it may be an attribute of detective novels. We can access and update library-number any time exactly as if it were an attribute of novel. DML examples of using subclasses follow.

1. Retrieve library-number and title of any book that is on-loan:
   From Books
   Retrieve library-number, title
   Where on-loan
   (Note: On-loan is a Boolean attribute of Loanable.)
2. Delete a loanable object that is a book and where the title of the book is "Spot's Big Book of Words."
   Delete Loanable
   Where ISA Book and
     title as Book = "Spot's Big
     Book of Words"
   (Note: When an object is deleted from a subclass, it is deleted from all the subclasses of the class but not from any superclasses. If we had used "Delete Book . . .," the object would still exist as a Loanable.)
3. Update the Book with ISBN equals 212346643 marking it as on-loan.
   Update Books
     (on-loan := true)
   Where ISBN = 212346643

Relational databases have no similar concept to these examples. To represent these data structures in relational databases, you have two extreme positions and many compromises in between.

One choice is to use a single large table. The problem with this is that each row will have many nulls. For instance, a row that represents a journal will have all the Book attributes set to null. There is no automatic integrity option to ensure that a consistent set of attributes is null.

The other extreme is to have many small tables with one equivalent to each subclass. The problem then is to guarantee the integrity of the one-to-one links between the table rows. For instance, you must ensure that a Book is not a Cassette. Attribute inheritance can be emulated by using Views, but few relational systems will support the updatability of Views.

This problem has been recognized in relational databases and there is a proposal to put a similar feature into a forthcoming version of the SQL standard.

SIM on the A Series

SIM uses DMSII as its underlying database engine. An SIM database cannot be subverted by accessing the DMS database directly. The interface used to call the DMS software is unique to SIM. It is as if the database is a completely new implementation. The advantage of using DMS has been that SIM has inherited DMS' resiliency features and utilities. The disadvantage is that DMS lacks the extensible data structures that allow adding structures and attributes on the fly. This will be remedied shortly when DMS on-line reorganization is introduced.

The SIM database schema is defined using the ADDS data dictionary. ADDS is also accessed when the database is opened and it is ADDS that stores the information of where the database is physically stored (or, more strictly, where the DB Control file is stored).

An SIM database can be accessed either through a host language interface or through two utilities: Interactive Query Facility (IQF) and Workstation Query Facility (WQF). Host language extensions are available for Algol, Pascal and COBOL74.

IQF and WQF are both reporting tools for SIM databases. Both use a tick-the-box style of interface for building SIM queries and allow the user to write queries in DML. Both have browsing and update facilities. The difference is that IQF runs on TD-like terminals (i.e., simple A Series terminals) while WQF runs on B20s or PCs. IQF also has sophisticated report-painting facilities far superior to WQF, but WQF has better schema-reporting capabilities, including a graphic display. Although both can be used by end-users, IQF is targeted at the sophisticated end-user who uses the system regularly and needs the additional report features while WQF may be best for occasional users.

All the mainframe-based products — ADDS, IQF and Operations Control Manager (OCM) — run under a shell called INFOEXEC, which supplies a consistent interface and seamless menu traversal between these products.

Finally, there are two tools — LINC.VIEW and DMS.VIEW — that can let a user generate an SIM interface to existing LINC or DMS databases. Clearly, the semantic content of such databases is not as good as if the databases were designed from scratch using SIM, but they do make it possible to use the IQF and WQF utilities.

Other Data Models

How does SIM compare with other advanced data models? Alternative models can be divided into four categories, though in truth there are overlaps. These are:

- Artifical Intelligence (AI) databases
- Other semantic data models
- Relational database extensions
- Object-oriented databases.

AI databases, especially semantic networks, have many similar features to SIM. At least two key differences are worth highlighting. The first is that the distinction between schema and data is not clear cut in an AI system. In commercial database systems, the quantity of information in the schema is far less than that in the database proper. In AI database systems, it is much nearer parity and arguably the distinction between schema and data breaks down. The second difference is that the AI systems tend to use the ISA relationship (i.e., a subclass ISA superclass) in a subtly different way. They might say both — "A bird ISA animal" and "A bird ISA flying-machine" — while in SIM only the first is true (because Penguins are not flying machines). In SIM, a subclass is a strict subset of its superclasses, while in AI systems (and some non-AI databases) it may not be. The SIM notion of strict subclass has the useful property that any assertion about a class is true of all its subclasses. The AI view is much more loose and closely related to other AI concepts like Frame, where information can express the usual case, not the exceptions.

Many other semantic data models have been described in the literature but few of these have actually been implemented. Examples include the Entity/Relationship model, SDM, TAXIS, RM/T, GEM, SAM*, IFO and GENERIS.

Most of these that have been implemented are either oriented toward a specialist application or were established to make a specific academic point. SAM*, for instance, is a statistical database. IFO is designed to illustrate what the semantic concepts mean in a formal, mathematical sense.

The other way semantic concepts will get a wider airing is from relational database systems being extended with semantic concepts. Relational Technology, for instance, is working on a successor to its INGRES product that incorporates many semantic ideas like generalization. The SQL standard is still being developed and we are likely to see it incorporate some of these ideas.

A more recent development has been the interest in object-oriented database systems. This means many different things to many different people but clearly there is considerable overlap between the concepts of SIM and those of object-oriented data models. The SIM idea of inheritance by a subclass from its superclass is very similar to the object-oriented concept of inheritance. The idea of looking at a relationship as an attribute also has parallels in object-oriented databases. Both SIM and object-oriented systems share the fundamental tenet that the primary modeling concept is that there are things and properties of things. The agreement is so close that SIM can easily be classified as an object-oriented system.

One of the interesting aspects of these developments is how so many are feeling their way along in a similar direction from different starting points. More and more vendors of database systems are including generalization and some form of explicit relationships support. There is a trend toward putting more and more processing logic in the database schema. It started with View, continued with generalized assertions and will carry on further with derived attributes and the inclusion of object-oriented "methods" into the schema. Hopefully, in the future all of these similar paths will converge on a common point.

Summary

For now, though, SIM must be seen as a viable and advanced system with many powerful features. Major performance and feature enhancements for SIM are planned. Indeed, data models like SIM may be the wave of the future. The only difference is that they probably will be called object-oriented database systems instead of semantic data models.

References

1. "A Relational Model of Data for Large, Shared Data Banks," E. F., Codd, Communications of the ACM, Vol. 13, No. 6, June 1970.
2. "Semantic Data Models," J. Peckham and F. Maryanski, ACM Computing Surveys, Vol. 20, No. 3, September 1988.
3. "Semantic Database Modeling: Survey, Applications and Research Issues," R. Hull and R. King, ACM Computing Surveys, Vol. 19, No. 3, September 1987.
4. "Database Description with SDM: A Semantic Database Model," M. Hammer and D. McLeod, ACM Trans. Database, Syst. 6, 3, September 1981.
5. "Database Abstractions: Aggregation and Generalization," J. M. Smith and D. C. P. Smith, ACM Trans. Database, Syst. 2, 2, June 1977.

ABOUT THE AUTHOR

Chris Britton is now the A Series program manager for the Professional Services organization in Unisys Europe Africa division. He joined Burroughs in 1976 after taking a Natural Science degree in Cambridge University and spending a short period in the U.K. Civil Service. He spent the rest of the 1970s fixing DMS recovery problems. In 1983 he left Burroughs and spent 10 months in ICL working on IDMS design. He then rejoined the Europe Africa division of Burroughs in order to take a temporary assignment working on SIM in California. During 1984 and the first part of 1985 he worked in the Santa Ana facility on the functional and implementation design of SIM and had a leading part in formulating the overall SIM architecture. Since this time he has been doing European technical support and consultancy, and, more recently, program management.

Besides his interest in data management theory and practice, Chris has had wide experience in many aspects of the A Series, including MCP support and dabbling with CP2000s. Although his current assignment is non-technical, he has still not entirely escaped from his technical past and also spends time tracking the ISO IRDS standard.