XML-BASED SEMANTIC DATABASE DEFINITION LANGUAGE

Naphtali Rishe, Malek Adjouadi, Maxim Chekmasov,
Dmitry Vasilevsky, Scott Graham, Dayanara Hernandez
Florida International University, Miami, FL

Ouri Wolfson
University of Illinois at Chicago, Chicago, IL,

Keywords: Semantic binary data model, data definition language, extensible markup language.

Abstract: This paper analyzes different options for semantic database schema definition and describes a presentation format XSDL. Presentation of semantic database in a certain format implies that the format fully preserves the database content. If the database is exported to this format and then imported back to the database engine, the resulting database should be equivalent (Rishe, 1992) to the one that was exported. XSDL is used for information exchange, reviewing data from databases, debugging database applications and for recovery purposes. Among other requirements that XSDL meets are support of both schema and data, readability by the user (XSDL is a text format), full preservation of database content, support for simple and fast export/import algorithms, portability across platforms, and facilitation of data exchange.

1 INTRODUCTION

The Semantic Binary Database is perceived by its user as a set of facts about objects. These facts are of three types: facts stating that an object belongs to a category, facts stating that there is a relationship between objects, and facts relating objects to data, such as numbers, texts, dates, images, tabulated or analytical functions, etc. The database places the management levels of the end-users in direct control of the web of information the organization knows, schematically depicting the database.

A proprietary Schema Definition Language (SDL) has been developed and used to document semantic schemas (Rishe, 1994). With time this language has become less suitable for semantic database representation for several reasons.

1. SDL is a proprietary language. This negatively affects exchanging information with other parties.
2. SDL does not support export of data from the database; it is only intended for exporting database schemas.
3. SDL only supports the basic features of the Semantic Binary Data Model such as defining categories and relations. None of the extensions to Semantic Binary Data Model are supported. For example, category ordering based on object attributes is not supported.
4. SDL has many limitations and naming conventions. Databases with names of categories and relations that do not conform to SDL standard cannot be preserved on export to SDL.
5. SDL cannot be extended in a natural way. The format of commands that describe schema in SDL is fixed. Although SDL has a special command `.tC` to add extra information, it is not used to support all the extensions presently implemented in the semantic database engine. SDL is not structured; instead it is interpreted line by line. Thus, the only way to refer to an already specified category or relation is by its name. This makes it impossible for a database schema editing tool to rename a category or relation correctly even if the tool preserves all the unrecognized lines of the database description file.

To avoid these drawbacks, a new language, XSDL (XML-based Semantic Database Definition Language), is introduced as a representation of semantic databases (Vasilevsky, 2004). The language is based on XML (XML, 2004), which is the de-facto industry standard for exchanging structured information. The proposed XSDL is free from the abovementioned drawbacks of SDL.

1. XSDL is based on industry-standard XML, which is widely used for information exchange. This extends interoperability of the engine over different types of databases and systems that support the XML language.
2. XSDL supports export of both database schema and data.
3. XSDL supports all extensions of the Semantic Binary Model.
4. XSDL is free of limitations. It supports any characters in category names and relations by employing standard XML encoding.
5. XSDL is naturally extensible due to XML's containment property.

The structure of data stored in a database is defined by the database schema. When this data is exported to XSDL, it is represented in XML. XML has its own language, XSD (XML Schema Definition), to define structure. It may seem natural to use XSD to export the database schema. However, XSD is not used to represent semantic schemas, since it does not support several features of the semantic binary model. Although some features of the semantic binary model can be represented by XSD constructs, translation of these constructs back to semantic schema is hard and may be ambiguous. Thus, creating XSDL to represent semantic schemas is a more reasonable solution.

A detailed description of the XSDL format is given in the following sections. Note that the database engine supports export and import of XSDL format which fully preserves the database.

2 XSDL FORMAT DESCRIPTION

The format to be used for export/import operations is an XML document with the structure of tags defined below. It is an ASCII or Unicode file and it strictly adheres to the rules of the XML. This is an example of the semantic database description:

```
<Database Name="Simple Database">
   <Schema Name="Simple Schema">
      <Category Name="Student"
Type="Abstract" />
      <Category Name="Instructor"
Type="Abstract">
         <Relation Name="Teaches"
Range="Student" Cardinality="m:m" />
      </Category>
   </Schema>
   <Data>
      <Student>
         <Object ID="00ADE700FF" />
         <Object ID="00ADE70100" />
      </Student>
      <Instructor>
         <Object ID="00AD">
<Teaches>00ADE700FF</Teaches>
<Teaches>00ADE70100</Teaches>
         </Object>
      </Instructor>
   </Data>
</Database>
```

The example shows a simple database with two categories `Student` and `Instructor` and a `m:m` relation `Teaches` between them. The database contains two objects in category `Student` and one object in category `Instructor` that `Teaches` both students. Note that some properties of the schema objects are defined in their XML attributes, some properties of the schema objects are defined in their sub-elements, and some elements are empty.

The following sections describe the structure of an XSDL document. Descriptions of all the nodes are presented. Since nodes with the same tag may appear in different contexts, a node tag is shown with a prefix composed of its parent tag and a forward slash. This way, all possible combinations of parent tag/child tag are described. The root node, `Database`, is given without any prefix.

`<Database>` -- Any XSDL document contains one object `<Database>` that stores information about the database. The information may include a database schema, data, or both. The parameter is

`Name` – optional, specifies the database name.

`<Database/Comment>` -- content of `Comment` element is a description related to the database as a whole. This property is made a sub-element of the database and not an attribute since it is expected to be a long text. Long texts look better as values of elements rather than as attributes.

2.1 Schema Format

`<Database/Schema>` or `<Schema/Schema>`. This element contains the semantic schema or sub-schema of the database. In SDL, the main schema is composed of a plain set of several sub-schemas. The XSDL approach allows the sub-schemas to form a tree hierarchy. Therefore `<Schema>` elements can be nested like `<Schema/Schema>`. The parameter is

`Name` -- the name of the schema.

`<Schema/Comment>` -- value of `Comment` contains description of the current sub-schema.

`<Schema/Author>` -- value of `Author` contains the author's name for the sub-schema.

<Schema/Category> -- Category has information regarding one category of the sub-schema, the parameters are

Name -- the name of the category;

Type -- possible values are 'Abstract' and 'Concrete', specifies the type of the category;

IsMetacategory -- optional Boolean attribute, specifies if the category belongs to the metaschema of the database;

IsPredefined -- optional Boolean attribute, specifies that this is a predefined schema category like Integer or a user-defined category.

<Category/Comment> -- the value of Comment contains a description of the category.

If category is an abstract category <Category> element contains sub-elements that define relations having this category as a domain. These sub-elements are described below. If the category is a concrete category it belongs to one of the concrete types defined in the database metaschema. In this case, the category contains one sub-element with the name of the corresponding type.

2.2 Sub-elements for Concrete Categories

<Category/Binary> -- specifies that the category inherits from the category Binary. The parameters are

MinimumLength -- minimum possible length of an attribute;

MaximumLength -- maximum possible length of an attribute.

<Category/Fixed> -- specifies that the category inherits from the category Fixed. The parameters are

UpperBound -- upper bound for valid values;

LowerBound -- lower bound for valid values;

Step -- granularity of scale.

<Category/Integer> -- specifies that the category inherits from the category Integer. The parameters are

UpperBound -- upper bound for valid values;

LowerBound -- lower bound on valid values.

<Category/Enum> -- specifies that the category inherits from the category Enum.

<Enum/EnumItem> -- EnumItem is used to declare possible values of the category. The parameters are

Name -- symbolic name for this value;

Number -- optional numeric value, by default numeric values are assigned automatically.

<Category/UnicodeString> -- specifies that the category inherits from the category UnicodeString. The parameters are

ValidCharacters -- a set of valid characters;

Collation – collation;

MaxLength -- maximum length of the string.

<Category/ASCIIString> -- specifies that the category inherits from the category ASCIIString, where

MaxLength -- maximum length of the string.

<Category/DateTimeStamp> -- specifies that the category inherits from the category DateTimeStamp. The parameters are

LowerBound -- lower bound on values;

UpperBound -- upper bound on values;

LowestPrecision -- lowest precision allowed for the values;

HighestPrecision -- highest precision allowed for the values.

<Category/Float> -- specifies that the category inherits from the category Float. The parameters are

MantissaSize -- the size of mantissa;

ExponentSize -- the size of exponent.

<Category/PlainString> -- specifies that the category inherits from the category PlainString. The parameter is

MaxLength -- maximum length of string.

<Category/Integer32> -- specifies that the category inherits from the category Integer32. The parameters are

UpperBound -- upper bound for the values;

LowerBound -- lower bound for the values.

<Category/Natural32> -- specifies that the category inherits from the category Natural32. The parameters are

UpperBound -- upper bound for the values;

LowerBound -- lower bound for the values.

2.3 Sub-elements for Abstract Categories

<Category/Display> -- contains information on how the category should be displayed by the graphical tools displaying semantic database schemas, and represented on the printouts of said schemas. By default, the graphical tool should automatically place the category to achieve the best quality of display or printout. The parameters are

X -- X position of a category;

Y -- Y position of a category.

Both X and Y are coordinates in Cartesian coordinate system with horizontal axis X going from left to right and axis Y going from bottom to top. The aspect ratio of the coordinate system is 1. All coordinates of the categories are presented in the same coordinate system on the sub-schema level. When the sub-schema is displayed it is up to the graphical tool to position and scale the coordinate system so that the sub-schema fits well within the display/printout boundaries.

`<Category/RecordPlacement>` -- information on the category is placed as a record in the database metaschema. The parameter is

- `Length` -- optional length of a record, no read or write operation should be attempted beyond this size, by default, the size is the maximum of (`Attribute/Offset + Attribute/Length`) for every concrete relation in the category.

`<Category/Attribute>` -- information on a concrete relation for the category. The parameters are

- `Name` -- short name of the concrete relation;
- `Range` -- name of the range of the concrete relation;
- `IsTotal` -- optional, specifies whether the concrete relation is total or not, possible values are `True` and `False` (default).

`<Attribute/RecordPlacement>` -- information on how the concrete relation, represented by the parent node, should be placed in a record. The parameters are

- `Number` -- optionally specifies the order number of the concrete relation in a record;
- `Length` -- the size in bytes allocated for the field in the record, no read or write operation should be attempted beyond this size;
- `Count` -- the number of times the concrete relation should be repeated. If count is greater than `1`, then an array of fields should be stored;
- `Offset` -- optional, distance in bytes from the beginning of the record for the category to the place where the value for the concrete relation should be written. By default, this offset is determined automatically as the sum of all lengths of all attributes that come before the current one in the record. Typically, this XML attribute is omitted, but it can be specified if full control over record placement is desired.

`<Category/SortKey>` -- information on how the category is sorted; it is also the semantic key of the category. The parameter is

- `Mode` -- optionally specifies how duplicates (objects with the same values of all sort key items) should be treated. Possible values are `NoDuplicates` (default), `FIFO`, `LIFO` and `Manual`.

`<SortKey/KeyItem>` -- category sort key contains several items, representing relations of the category. The functionality of semantic key is achieved when only `Name` of relations is given. The parameters are

- `Number` – The significance number of this `KeyItem` in the sort key. If the sort key contains several items, the objects are first ordered by the value of the first item, then those that have the same value of the first item are ordered by the second item, etc.;
- `Name` -- the short name of the relation of the domain category. The value of this attribute is used to form the sort key;
- `Order` -- optional, specifies direct or reverse ordering on the item, valid values are `Direct` (default) and `Reverse`.

`<Category/Relation>` -- information regarding a relation having the current category as its domain. The parameters are

- `Name` -- short name of the relation;
- `Range` -- range of the relation;
- `IsTotal` -- optionally specifies whether the relation is total or not. Possible values are `True` and `False` (default).
- `IsPersistent` -- optionally specifies whether the relation is persistent. Possible values are `True` and `False` (default).
- `InKey` -- optionally specifies whether the relation is part of the semantic key for the category. Possible values are: `False` (default, means that the relation is not in the semantic key for the category), `True` (means that the relation is part of the semantic key of the category).

`<Relation/Comment>` -- value of the `Comment` contains a description of the relation.

`<Relation/DomainSortKey>` -- information on how the relation is sorted on the domain side. If several objects in the domain category are connected by this relation to the same object in the range category, those objects in the domain category will be retrieved in the order determined by this node. The parameter is

- `Mode` - optionally specifies how duplicates (objects with the same values of all sort key items) should be treated. Possible values are `NoDuplicates` (default), `FIFO`, `LIFO` and `Manual`.

`<DomainSortKey/KeyItem>` -- relation sort key contains the items representing attributes of the domain category. The parameters are

- `Number` -- significance number of this `KeyItem` in the sort key. If the sort key contains several items, then the objects are first ordered by the value of the first item, then those that have the same value of the first item are ordered by the second item, etc.;
- `Name` -- the short name of the attribute of the domain category. The value of this attribute is used to form the sort key;
- `Order` -- direct or reverse ordering on the item, valid values are `Direct` and `Reverse`.

`<Relation/RangeSortKey>` - information on how the relation is sorted on its range side. If several objects in the range category are connected by this relation to the same object in the domain category, those objects in the range category are retrieved in the order determined by this node. The parameter is

Mode -- optionally specifies how duplicates (objects with the same values of all sort key items) should be treated. Possible values are NoDuplicates (default), FIFO, LIFO and Manual.

<RangeSortKey/KeyItem> -- relation sort key contains several items, representing the attributes of the range category, the parameters are

Number -- significance number of this KeyItem in the sort key. If the sort key contains several items, the objects are first ordered by the value of the first item, then those that have the same value of the first item are ordered by the second item, etc.;

Name -- the short name of the attribute of the range category. The values of the attribute are used to form the sort key;

Order -- direct or reverse ordering on this item, valid values are Direct and Reverse.

<Category/Subcategory> -- specifies one of the subcategories of the category. The parameter is

Name -- the name of the subcategory.

<Subcategory/Comment> -- contains a comment about this inheritance.

<Category/CoveringGroup> -- specifies the covering group for the category. If the object belongs to the category it should belong to one of the categories in the covering group. Typically it is used to specify 'No Other' property, which declares that there are no objects that belong to this category but not to any of it's subcategories. The parameter is

Name -- the name of the covering group.

<CoveringGroup/Comment> -- the description of the covering group.

<CoveringGroup/CoveringItem> -- contains one item of the covering group. The parameter is

Name -- The name of the category belonging to the covering group.

<Schema/DisjointGroup> -- the categories in the disjoint group should be disjoint from each other.

<DisjointGroup/Comment> -- description of the disjoint group.

<DisjointGroup/DisjointItem> -- specifies the categories in the disjoint group. The parameter is

Name -- the name of the category belonging to the disjoint group.

2.4 Data Format

It is possible to represent data in two ways. ObjectsFirst is the format when every object in the database is represented by exactly one XML node. For each object, the information about its categories and relations is contained within the object definition. This format is natural semantically.

The other way is to group objects by category. The CategoriesFirst format addresses this by having all the objects grouped by categories. Nodes that represent categories are higher in the containment hierarchy than the object nodes. Within the category node, the object node corresponds to the object belonging to this category. If an object belongs to more than one category, there are several nodes corresponding to the object in every category it belongs to. All these nodes have the same ID and, therefore, represent the same object.

<Database/Data> -- contains all the data unloaded from the database, the parameter is

Format -- the format in which the data is presented. The valid values are ObjectsFirst and CategoriesFirst.

The format for ObjectsFirst is described as follows.

<Data/Object> -- object node. The node is created for each object in the database; it contains the information about the object. The parameter is

ID -- the unique ID of an object. If the object has a reference to another object, the reference is established by ID. For implementation purposes it will be a hexadecimal internal object ID.

<Object/Category> -- category node. Represents the fact that the object belongs to the category specified by name. The value of this tag is a list of all attributes and relations having the category as a domain. The parameter is

Name -- name of the category the object belongs to.

<Category/Relation> -- relation node. The value of the node represents the object or value to which the current object is connected by the relation specified. If the relation is abstract, the value is the unique ID of the object to which the current object is connected. For Integer, the value is a decimal representation of an integer. String is represented as appropriate in XML format. Binary data is represented by XML's CDATA section. Values of enumerated type are represented by their string values. The parameters are

Name -- name of the relation;

Number -- optional attribute for ordered m:m relations. If the object is connected to several objects by the relation and it has manual ordering, this attribute contains the order number.

Example of the database presented in ObjectsFirst format is below:

```
<Database>
   <Data>
```

```
      <Object ID="A00000">
         <Category Name="Student">
            <Grade>3</Grade>
         </Category>
         <Category Name="Instructor">
            <Relation Name="Teaches"
Number="1">A00000</Relation>
            <Relation Name="Teaches"
Number="2">A00001</Relation>
         </Category>
      </Object>
      <Object ID="A00001">
         <Category Name="Student">
            <Relation
Name="Grade">5</Relation>
         </Category>
      </Object>
   </Data>
</Database>
```

The format for `CategoriesFirst` is described as follows.
`<Data/Category>` -- category node. Represents the information about all objects belonging to the category specified by name. The parameter is
`Name` -- the name of the category.
`<Category/Object>` -- object node. Contains information about the object, namely the values of those relations that have the category specified by name in the parent node as a domain category. The parameter is
`ID` - unique object ID. Note that if the object appears in two or more categories, its ID is the same in all categories.
`<Object/Relation>` -- relation node. The value of the node represents the object or the value to which the current object is connected by the relation. The parameter is
`Name` -- the name of the relation.
An example of the same database presented in `CategoriesFirst` format is below:

```
<Database>
   <Data>
      <Category Name="Student">
         <Object ID="A00000">
            <Relation
Name="Grade">3</Relation>
         </Object>
         <Object ID="A00001">
            <Relation
Name="Grade">5</Relation>
         </Object>
      </Student>
      <Category Name="Instructor">
         <Object ID="A00000">
            <Relation Name="Teaches"
Number="1">A00000</Relation>
            <Relation Name="Teaches"
Number="2">A00001</Relation>
         </Object>
      </Category>
   </Data>
</Database>
```

In addition to the formats of data export described above, the alternative form of category nodes and relation nodes can be supported. This form may be used if category and relation names do not contain special characters that are not allowed in XML tags. In this case the category names and relation names are used as node tags:
`<CatName>` -- alternative form of category node;
`<RelName>` - alternative form of relation node.
However, in this case the XML does not have a fixed schema, thus making validation of the XML a hard task. XML files without a fixed schema are not preferable for information exchange.

3 CONCLUSION

We have presented XSDL -- the data definition language for the semantic binary model. The new approach is based on XML. XSDL proves to be more flexible in describing semantic database schemas than conventional SDL. In particular, XSDL is used for fast export/import operations of semantic databases and for efficient data exchange between the databases.

ACKNOWLEDGEMENTS

This material is based on work supported by the National Science Foundation under Grants No. HRD-0317692, EIA-0320956, EIA-0220562, CNS-0426125, IIS-0326284, CCF-0330342, IIS-0086144, and IIS-0209190.

REFERENCES

Rishe, N., 1992. *Database Design: the semantic modeling approach,* McGraw-Hill. 528 pp.

Rishe, N., 1994. *Semantic Schema Design Language,* available at request, http://hpdrc.cs.fiu.edu .

Vasilevsky, D., 2004. *Design Principles of Semantic Binary Database Management Systems*, PhD thesis, Florida International University, 165 p.

XML, 2004. Extensible Markup Language, *World Wide Web Consortium*. http://www.w3c.org/XML .