5

Schema Evolution in the STAR Framework[1]

Miguel R. Fornari[2], Lia G. Golendziner, Flávio R. Wagner

Universidade Federal do Rio Grande do Sul, Instituto de Informática
Caixa Postal 15064, 91510-970 Porto Alegre RS, Brazil

E-mail: {miguel, lia, flavio}@inf.ufrgs.br

Abstract

The STAR data model supports the definition of object schemata, according either to some design methodology or to the designer's decision. Object schemata allow a flexible management of the various representations that are created during the design of a particular object. Object schemata can evolve or even be dynamically defined, departing from an existing object schema and making changes to it. Schema evolution facilities are a valuable support for both the definition of new design objects and design methodology management. Schema evolution is maintained through versions, so that version management is applied not only to design objects themselves, but also to object schemata. Consistency is guaranteed for schema operations, based on a set of invariant rules.

1 Introduction

Typical EDA frameworks are built upon a database management system that offers data representation facilities and basic versioning mechanisms. On top of this layer, various servers, eventually implemented as domain-neutral tools, are available. Typical servers support the management of versions, configurations (both aspects of data management), and design methodologies.

In a complex design environment, where prototyping is a common way of developing systems, there is a great need to change the schema definition [Bert91]. This process requires a special tool, incorporated to the database, to allow operations over the schema [Atki89]. Schema updates should not represent an extreme overhead and, at the end of the changes, schema and instances must be correct and consistent. There is also a need for flexibility, which tends to make this kind of operation more expensive to the system.

This work describes an innovative data definition layer that allows the creation and evolution of object schemata in the STAR framework [WGF94]. The mechanism uses object

[1]This work was partially supported by CNPq and CAPES.

[2]Currently at Universidade Luterana do Brasil (ULBRA). Caixa Postal 124, CEP 92420-280, Canoas-RS-Brazil.

F. J. Rammig et al. (eds.), *Electronic Design Automation Frameworks*

versions to represent several states of the same object and to permit the return to previous states of one object in a natural way. Changes to an object schema are not done "in-place", but new versions are generated, so that a schema evolution history is kept. The main objective of this mechanism is to allow the construction of a new object schema based on an existing one. Tested object schemata can then be reused, improving design team productivity and increasing reliability of resulting systems.

The remaining of the paper is organized as follows. Section 2 introduces the STAR data model, the design methodology layer and the version model. Section 3 presents the schema definition and evolution layer. Section 4 presents a comparison of this mechanism with those existing in other frameworks and object-oriented database systems and Section 5 concludes with final remarks.

2 The STAR framework

2.1 The STAR data model

In the STAR data model, shown in Figure 1, each *Design* object gathers an arbitrary number of *ViewGroups* and *Views*. The *ViewGroups* may in turn gather, according to user- or methodology-defined criteria, any number of other *ViewGroups* and *Views*, building a tree-like hierarchical object schema.

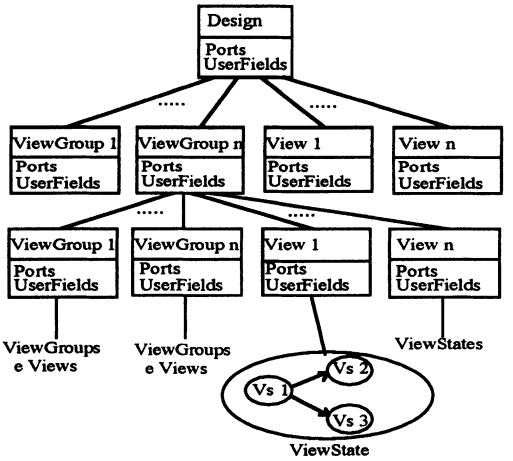

Figure 1: The STAR data model

The fact that *Views* may be defined at any level of the object schema offers an unlimited number of ways for organizing the different representations of the *Design*. Since the system does not enforce any grouping criterion, it is left to the user or to the design methodology to decide how *Views* will be organized. *Views* are of one of the types: HDL, for behavioral descriptions, MHD, for structural descriptions and *Layout*, for geometric descriptions.

The object schema is a generalization hierarchy, in that each node is an abstraction of the subtree below it. Properties defined at each node may be inherited by its descendant nodes (inheritance is optional). Inheritance occurs among instances: not only the existence of an attribute is inherited by the descendant nodes, but also its value, when defined. Inheritance may be by default, when descendant nodes may redefine attributes to more specialized domains and modify attribute values, or strict, when redefinition at descendant nodes is not possible.

The purpose of *Design*, *ViewGroup* and *View* nodes of the object schema is to organize the various representations of a *Design* object and to guarantee the consistency of the common attributes through the inheritance mechanism. Therefore, these nodes contain only the attributes to be shared by the representations they gather.

Real design data, such as structural decomposition, HDL descriptions, and layout masks, are contained in the *ViewStates*, that are revisions created for each of the *Views*.

There are three types of attributes for each node of the schema: *UserFields*, *Ports* and *Parameters*.

UserFields are user-defined object attributes, which have a name and a domain specified. The domain can be simple (character, string, integer, real, boolean) or composed (record, array, set, subset and enumeration). Inheritance is optional for *UserFields,* that is, attributes can be defined as "local" to one node and not passed down to the inheritance hierarchy. If the inheritance type is defined as default for a *UserField*, its domain can be redefined, but only making restrictions to it. The new domain must be a subset of the original one.

Ports are interface signals and may contain in turn their own user-defined attributes. *Port*'s definition specifies: a domain, that indicates the type of information contained; a direction (in, out or inout); and the number of wires. There are two types of *Ports*: *PortWires* with just one wire and *PortBundles* composed by a set of *PortWires*. *Ports* are interconnected by *Nets*.

Parameters allow the user to build generic, parameterized objects. *Parameters* have only a name and a domain. Inheritance is mandatory regarding the existence of *Ports* and *Parameters*.

The STAR data model allows the specification of relationships between objects, which is done through an object of type *Correlation*. A *Correlation* has a direction and a mode. The direction (*bi-directional, directed* or *non-directed*) indicates an existence dependency. For example, a correlation defined as A $\rightarrow$ B (A directed to B) indicates that B can only exist while A exists. The mode (*protect* or *delete*) indicates the action to be done when the removal of an object is required. If the mode is *delete,* a removal of a node causes the removal of the nodes that depend on it through *Correlations*. Otherwise (protect mode), a removal cannot be executed if there are dependent nodes. In a non-directed *Correlation*, the mode is irrelevant. *Correlations* can also have *UserFields* and a relationship criterion, for documentation purposes.

It is important to notice that an object schema in STAR is a hierarchy that presents inheritance not only for attribute definition (as occur for most of the OODBMS) but also for attribute values, which can be defined at any level of the hierarchy. Value inheritance brings up two problems into consideration: modifications in object descriptions and modifications in attribute values.

2.2 Design methodology management

A design methodology is a set of design rules that either enforce or guide the activities performed by the user, so as to obtain objects with desired properties. The definition of a

design methodology in the STAR framework is based on three main principles [WGF94]: the definition of the object schema for the design objects, the specification of the task flow and the hierarchization of design strategies. The object schema has been already discussed above.

Task flow is expressed through a condition-driven model. A task is described with a 4-tuple (*name, pre-conditions, tool, post-conditions*). A task is eligible for execution when its pre-conditions hold. These conditions can express the existence of objects or properties of them, explicitly modeled as attributes in the object schema. To execute the task, the specified tool is used. Post-conditions describe the properties expected from the objects after a task is executed. Again, a set of new objects, generated by the tool can be expected as the result of the execution. If the post-conditions are not achieved, the task fails, though new object representations might have been created. A task execution is considered a long database transaction, whose effects can be undone if the user asks for. It is left to the user to select among many enabled tasks. A methodology succeeds when all its tasks have succeeded. Tasks may be executed stand-alone or within a design strategy.

Design methodologies can be organized in a hierarchical way. A new design methodology can be derived from a previous one by extending the object schema (using the schema evolution mechanism) or defining new tasks.

Task definition must be consistent with the object schemata that are known to this methodology, that is, all referenced objects and attributes must exist and attribute comparisons must be done with correct domain values.

2.3 Version management

The STAR framework provides a two-level versioning support [Wagn92]. At a conceptual level, the object schema defines *ViewGroups* and *Views* that represent different design views and alternatives, according to user or methodology control. At a lower level, revisions are automatically generated by the system, when updates are done to specific representations of the design object.

There are two revision mechanisms. First, to each View an acyclic graph of ViewStates is appended. They contain the real design data that corresponds to the various design representations (layouts, HDL, descriptions, and so on). Another mechanism allows the sequential versioning of the other nodes of the object schema (*Design*, *ViewGroup*, and *View*), due to changes made to attributes that were defined as versionable. The system maintains the correspondence between *ViewStates* and versions of ascendant nodes, thus linking each *ViewState* to the inherited attributes that were valid at the time of its creation.

Versions have an associated status, representing their design stage, which can be *in progress, stable* or *consolidated*. In progress versions can be changed or deleted. Stable versions can be deleted, but not changed. To maintain the historical sequence of versions, they are only logically deleted. Consolidated versions can not be changed nor deleted. They can only be selected and read. When a version is promoted (to stabilized or consolidated), its predecessors are also promoted to the same status.

Attributes (*UserFields*, *Ports* or *Parameters*) can be defined as versionable or non-versionable. A modification on a versionable attribute implies a creation of a new version, exception made for versionable attributes of in progress versions, which can be modified. The automatic revision control guarantees that when a stabilized object is modified, a new version is

created as a copy of it, but with the modified values (if the modified attribute was defined as versionable). A non-versionable attribute can not be modified in any way. The new version is created with in progress status.

For all objects having versions in the object schema, there is the notion of *current version* (by default, the most recent one). The user can query old versions without changing the current one. Changing current versions is made through a selection operation, in one of the two following ways:

Partial: A version is selected from one node of the object schema and only the current version of this node is changed. Current versions of all other nodes in the object schema are not changed. It is user's responsibility to verify consistency among the version selected for this node and the others in the object schema.

Total: The user chooses either a specific version in one of the nodes or a *ViewState*. The system then changes the current version of all the ascendant nodes to the version that were valid at the creation time of the chosen version (or *ViewState*).

3 Schema Evolution

Due to the nature of the design process, an object schema may need to be dynamically modified in several ways, reflecting new specifications and user requirements, inclusion of a new tool to the environment and correction of modeling errors. In particular, schema evolution is an essential feature for supporting design methodology management in an evolving environment, where the inclusion of new tools and strategies, during the design process, may impose the incorporation of new types of object representations and new attributes to the already existing object schemata.

A mechanism for the definition and evolution of object schemata has been developed for the STAR framework. As in object-oriented databases [PeSt87, KiCh88, Deux91], this mechanism is based on *schema invariants*, which are basic conditions that must always hold to insure that the object schema is in a consistent state.

Object schemata can be created from scratch, a simple situation for the schema evolution manager, but it is highly desirable to develop an object schema from an existing one, well tested and approved. To achieve this, the first operation must be the copy of one existing schema to a new area. The inclusion of new *ViewGroups* and *Views* does not affect *ViewStates* that already exist. Modifications in nodes having associated ViewStates can also be done. In this case, new *ViewStates* have to be generated, reflecting the modification in ascendant nodes. This method of object schema definition results in an important decrease of design development time.

3.1 Schema Invariants

Invariants for the schema evolution mechanism were defined to assure database integrity and object schema correctness. The invariants for the STAR schema evolution are[3]:

[3]For simplicity, object means Design, ViewGroup or View. Instance means ViewState and attribute means UserField, Port or Parameter.

- Each object inherits properties from only one other object (single inheritance). This restriction comes from the definition of the object schema as a tree-like generalization hierarchy.
- All descendant nodes from an object have a unique name. The complete name of an object is composed by the Design name and all the descendant object names that are in the path to the mentioned object in the object schema.
- All attributes of an object have unique names. This restriction guarantees attribute identification.
- All inheritable attributes of an ascendant object are inherited. Since the first invariant assures that multiple inheritance does not exist, name conflicts do not occur.
- *UserFields* inherited by default can be redefined in descendant nodes: the domain can be redefined to more specialized domains or values can be modified. Strict inherited attributes (some *UserFields*, all *Parameters* and *Ports*) can not be redefined.
- All referenced objects in the schema are present in the database. This is the referential integrity of relational databases adapted to the STAR framework.

This invariant set assures the integrity of the database objects. However, sometimes a sequence of operations is needed to go from a consistent database state to another one. Then, a modeling transaction can be started, disabling the invariant checking until the transaction is committed. One exception is the stabilization of a version. When this operation is required, the hierarchy where the object version is included must be verified to assure that only correct versions are stabilized. This modeling transaction is typically a long transaction and must be incorporated in the mechanism of long transactions provided in the framework.

3.2 Operations

A complete set of operations for schema modification is defined to allow an easy data modeling and evolution in the STAR framework. These operations are the basis for a higher level object schema definition language and are listed below. The name of the object is in bold, its properties are in italic and the possible operations on an object are cited. Brackets indicate an option to be taken. Square brackets indicate an optional value. If not specified, a NULL value is assumed.

- **Library** *(Name)*
  Operations: Create; Delete.
- **Design** *(Name, Library)*
  Operations: Create; Delete.
- **ViewGroup** *(Name, {Design, ViewGroup}, [Criterion])*
  Operations: Create; Delete; Modify ascendant object or criterion.
- **View** *(Name, {Design, ViewGroup}, Type)*
  Operations: Create; Delete; Modify ascendant object.

- **Parameter** *(Name, Domain, Inheritable, Versionable, Object)*
  Operations: Create; Delete; Modify domain, versionable characteristic and/or inheritable characteristic.
- **UserField** *(Name, Domain, Inheritable, Versionable, Inheritance Type, Object, [Value])*
  Operations: Create; Delete; Modify domain, inheritable characteristic, versionable characteristic and/or inheritance type; Modify value; Move the *UserField* to another object.
- **Port** *(Name, Type, Object, Versionable, Direction, [Number of wires], [Domain])*
  Operations: Create; Delete; Modify versionable characteristic, direction, number of wires and/or domain; Move the *Port* to another object.
- **Correlation** *(LeftObject, RightObject, Direction, [Mode], [Criterion])*
  Operations: Create; Delete; Modify objects, direction, mode and/or criterion.

When a new node is created, its name and its immediate ascendant must be informed. The name of the node must be unique, according to the schema invariants. When a node is removed, all its descendant nodes are removed too. For in progress versions, the design data are really removed. For stable versions, the data are maintained in the database, but just historical queries can be done on them. Consolidated versions cannot be removed. A node can change its place in the object hierarchy, moving to another ascendant object. This operation is semantically equivalent to a combination of a removal from the original node and an insertion in the new one. All descendant nodes, if they exist, are moved together.

Attributes can be inserted, removed and copied at any time. If the current version status is not in progress, a new version is derived from it, and the attribute modification is effective in this new version.

If an inheritable attribute is redefined, the inheritance mode must be verified. If this attribute redefines another inherited attribute, then the inheritance mode should have been defined as *by default*, and the redefined domain must be a subset of or equal to the inherited domain.

When the domain of a *UserField* is modified, its value should be changed to keep consistency. The user can define a special function that automatically maps the old values to values in the new domain.

The modification of an attribute from versionable to non-versionable can be done at any moment. This modification alters only the semantics of value modification of an object and does not change the object schema.

Modifications in the number of wires in a *Port* are possible just for *PortBundles*. In case of reduction of *PortWires* it is necessary to indicate *PortWires* to be removed. If the number of wires is increased, a list of new *PortWires* has to be indicated. When the direction of a *PortBundle* is modified, the direction of all *PortWires* that composed it must be modified too. Designers receive a list of *Nets* that are affected by the modification, to allow a manual correction.

ViewStates are not considered in the schema evolution manager because they are instance objects. Creation, removal and other operations on *ViewStates* are directly controlled by the data manipulation language (DML). All modifications in the object schema are reflected in the instances, i.e., the *ViewStates*. The automatic revision control generates a new version when some characteristics are modified, in a consistent way.

Correlations can be freely modified because changes on them do not violate any invariant, just modify the semantics of the delete operation.

Copy of a node is an essential operation to allow the designer to reuse a well tested and approved object schema in the development of a new design. The designer can copy a well-established design to a new area and make some modifications to obtain the necessary conceptual schema for the new design. The designer can copy just one node or the node and all its descendants.

A great number of schema evolution operations can affect the correction of already defined tasks. For example, an attribute used to express a pre-condition of a task, if removed, turns the tasks' definition incorrect. When such situation occurs, a list of incorrectly defined tasks is returned to the designer, who is responsible for making the necessary modifications for correcting the affected tasks.

4 Comparison

In the STAR framework, both the final user and the application programmer have full access to the schema evolution facilities, including removal and redefinition of nodes and attributes.

In the CADLAB framework [Gott88], for instance, the final user may only extend existing schemata, by using the TIDL language [Groe91] and recompiling the schema definition. The application programmer, in turn, may also delete attributes and object types for which no instances exist.

In the NELSIS framework [Wolf88], the database system offers a semantic data model (called OTO-D), versions, a graphical query interface, tool activation, support for design transaction and physical distribution, but there is no schema evolution facility.

The power of the schema evolution mechanism of the STAR framework can be compared to those present in object-oriented databases, considering the large number of available operations.

Invariants are used by ORION [KiCh88], O_2 [Deux91] and GemStone [PeSt87]. These invariants guarantee the structural correctness of schemata, but do not include behavioral aspects. In STAR, the mechanism that returns to the designer a list of affected tasks is similar to the mechanism that controls method modifications in O_2 and is called behavioral consistency [Zica91].

The possibility of combining versions of nodes and schema evolution presents similarities with the proposals by Kim & Chou [KiCh88] and Skarra & Zdonik [SkZd86]. However, in [KiCh88], versions of the whole schema are generated after a schema modification. In [SkZd86], versioning is done for a single class. Modifications that impact stored objects (for example, changing a domain of an attribute) must be managed by *handlers* provided by the user. In the STAR mechanism, versions are created for any node in the object schema. Version nodes are connected so that it is possible to return to previous versions of any node, keeping the correspondence among all versions of the schema and the design data (represented by the *ViewStates*).

In ISIS-V [DaZd86], at each transaction commit, a new version of the entire database is generated, defining a linear sequence of database states. All changes made during a transaction, schema or instance modifications, are stored in the new database version. Returning to previous definitions implies returning the whole database to a previous state.

5 Final remarks

This paper described a schema evolution mechanism considering the STAR data model. The proposed mechanism allows the definition of a schema and its modification in several ways, either adding or removing nodes/attributes/relationships between nodes. The mechanism is extremely flexible and capable of retaining the system's development history, based on versions of objects.

The implementation of the STAR database is being made using KRISYS [Matt91], a knowledge base management system that provides object-oriented concepts. The version layer is implemented. All operations here described have been completely specified and are being implemented as part of the data definition language.

Bibliography

[Atki89] M.Atkinson et al. *The object-oriented database system - manifesto.* Rapport Technique Altaïr 30-89, 21 août 1989.

[Bert91] E.Bertino. Object-oriented database management systems: concepts and issues. *Computer*, Los Alamitos-CA, v.24, n.4, p.33-47, Apr. 1991.

[DaZd86] J.W.Davison, S.B.Zdonik. A visual interface for a database with version management. *ACM Trans. on Office Information Systems*, v.4, n.3, July 1986.

[Deux91] O.Deux. The O2 system. *Communications of ACM*, v.34, n.10, Oct. 1991.

[Gott88] K.Gottheil et al. The Cadlab Workstation CWS - an open, generic system for tool integration. In: F.J.Rammig(ed.). *IFIP Workshop on tool integration and design environments.* North-Holland, 1988.

[Groe91] K.Groening et al. From tool encapsulation to tool integration. In: F.J.Rammig, R.Waxman(eds.). *Eletronic Design Automation Frameworks.* Elsevier Science Publishers, 1991.

[KiCh88] W.Kim, H-T.Chou. Version of schema for object-oriented databases. In: *VLDB Conference*, 1988.

[Matt91] Mattos, N.M. *An approach to knowledge base management.* Springer-Verlag, 1991. (Lectures Notes in computer Science, 513).

[PeSt87] D.J.Penney and J.Stein. Class modification in the GemStone object-oriented DBMS. In: *Sigplan Notices*, December 1987.

[SkZd86] A.H.Skarra, S.B.Zdonik. The management of changing types in an object-oriented database. *Sigplan Notices*, v.21, n.11, 1986. (OOPSLA-86)

[Wagn92] F.R.Wagner et al. Design version management in the STAR framework. In: *3rd IFIP International Workshop on EDA Frameworks.* North-Holland, 1992.

[WaVi91] F.R.Wagner and A.H.Viegas de Lima. Design version management in the GARDEN framework. In: *28th. ACM/IEEE Design Automation Conference*, June 1991.

[WGF94] F.R.Wagner, L.G.Golendziner, M.R.Fornari. A tightly coupled approach to design and data management. In: *EURO-DAC 94.*

[Wilk88] W.Wilkes. Instance inheritance mechanisms for object-oriented databases. In: K.R. Dittrich(ed.). *Advances in Object-Oriented Database systems.* Springer-Verlag, 1988.

[Wolf88] P. van der Wolf et al. Data management for VLSI design: conceptual modeling, tool integration & user interface. In: F.J.Rammig(ed.). *IFIP Workshop on tool integration and design environments*. North-Holland, 1988.

[Zica91] R.Zicari. A framework for schema updates in an object-oriented database system. In: *International conference on Data Engineering*, 1991.