3

Configuration Management in the STAR Framework *

Helena G. Ribeiro, Flávio R. Wagner, Lia G. Golendziner

Universidade Federal do Rio Grande do Sul, Instituto de Informática
Caixa Postal 15064, 91501-970 Porto Alegre RS, Brazil
e-mail {hgraz,flavio,lia}@inf.ufrgs.br

Abstract

Configuration management is an essential service to be provided by electronic design automation frameworks. Besides conventional static and dynamic configurations, normally offered by most systems, the STAR framework supports specialized facilities, such as open configurations (as in VHDL), the automatic resolution of open and dynamic configuration through expressions on object properties, and the manual resolution of these configurations by means of a graphical-interactive database browser. Unlike other systems, the STAR configuration manager is a separate framework module which the final user can directly reach through the main user interface. Furthermore, the STAR configuration management mechanisms respond to novel requirements, imposed by powerful versioning services.

1 Introduction

Configuration management is an important service that must be available in electronic design automation frameworks. Design objects to be handled in these frameworks have a complex and hierarchical structure, with objects composed of other sub-objects. The design of an object is an evolutive process, and in this context a set of versions can represent the same object in different moments of the process. When a single representation of a composite object is necessary to be submited to a design tool, a selection of versions for its sub-objects must be performed. This set of selected versions that define a given representation of the object is called an object configuration. The process of selecting versions for the sub-objects can be very long and difficult, because the sub-objects can be composed in turn of other sub-objects, and each sub-object in this hierarchy can have many versions. The user must select versions that are compatible with each other and

*This work was partially supported by CNPq and CAPES.

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

that best fit to given requirements. A configuration manager must provide resources to help designers establish and maintain object configurations.

Three types of configurations are defined in the literature: *static* configurations [1] associate a complete reference (object and version) to each sub-object; *dynamic* configurations [1] associate partial, incomplete references to the sub-objects; and *open* configurations [2] leave the references completely undefined. Open and dynamic configurations must be *resolved* to a complete reference before the object may be used by certain design tools.

There are many proposals of configuration management mechanisms in the literature (for instance [1, 3, 4, 5, 6]). These mechanisms support static, dynamic, and open configurations in different ways and offer different resources for the resolution of dynamic and open configurations.

This paper presents a configuration manager for the STAR framework [7], which is under development at the University of Rio Grande do Sul at Porto Alegre, Brazil. The STAR framework is based on a semantic data model that offers specialized, combined mechanisms to represent concrete design data and to manage the various representations created for each design object along the design evolution (alternatives, views and revisions). On top of the data model sits a version manager, which is responsible for revision control. The STAR configuration manager [8] supports static, dynamic, and open configurations. The hierarchical organization of versions in the STAR data model imposes some very specific requirements on the definition and resolution of configurations. Dynamic and open configurations can be resolved in a manual, automatic, or semi-automatic way, whereby versions may be selected according to desired object qualities. An interactive database browser and a special selection language are available for supporting version selection. After defined, configurations can be stored as database objects that persist in the framework and thus reused.

This paper is organized as follows. Section 2 reviews the main features of the STAR data model and of the STAR version management mechanisms. Section 3 defines configurations in the context of the STAR framework, while Section 4 presents the configuration management mechanisms. A comparison between the STAR mechanisms and those available on other systems can be found in Section 5. Section 6 concludes with final remarks.

2 The STAR framework

The STAR data model [9] provides a flexible management of the various representations created along the various dimensions of the design evolution (alternatives, views and revisions). This feature allows the system to implement, according to user- or methodology-defined criteria, conceptual schemata that are specialized for representing the design evolution of each object type.

As shown in Figure 1, each *Design* object gathers an arbitrary number of *ViewGroups* and *Views*. The ViewGroups may in turn gather, according to application-defined criteria, any number of other ViewGroups and Views, building a tree-like hierarchical *object schema*. Three types of Views are supported: *HDL Views*, for behavioral descriptions, *MHD Views* (Modular Hierarchical Description), for structural descriptions, and *Layout*

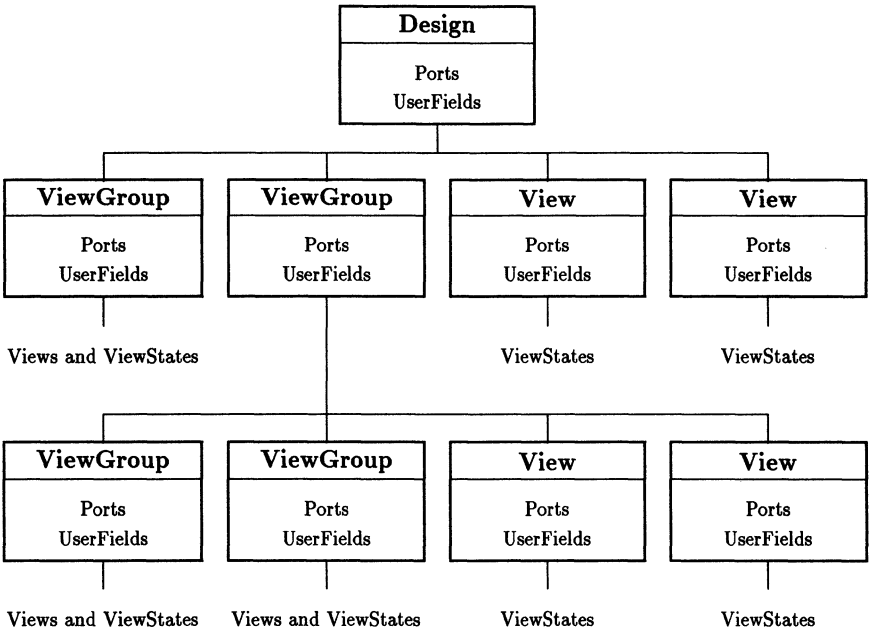

Figure 1: The STAR data model

Views, for geometric descriptions. In all View types, objects can be described as a composition of sub-objects that are instances of other objects, but only MHD Views handle the exact interconnections among the sub-objects.

ViewGroups can be used, for instance, to build a hierarchy of design decisions, where alternatives from a given design state are appended to the ViewGroup which corresponds to this state. The advantages and generality of this schema are stressed elsewhere [9, 10].

The object schema is a generalization hierarchy. Each node has properties that may be inherited by its descendant nodes. Not only the existence of an attribute is transferred to the descendant nodes, but also its value, when defined. *Ports* and *UserFields* (user-defined attributes) are among these inherited attributes. Therefore, they may be defined at any level of the object schema hierarchy.

Version management in the STAR framework [9] is supported by two different mechanisms, at two different levels. At a conceptual level, the user or the design methodology may define a particular object schema for each design object so as to organize design views and alternatives according to a given strategy. This allows the user to apply a methodology control which is highly tuned to the design of each object [7].

At a lower level, the system offers automatic mechanisms for handling the various revisions that are created for each design representation along the time axis. There are two revision mechanisms. Firstly, to each View (i.e., each leaf of the object schema) an acyclic graph of ViewStates is appended. ViewStates contain the real design data

that correspond to the various design representations, such as layouts, HDL descriptions, structural decompositions, and so on. ViewStates have an associated status, representing their design stage. Possible status values are *in progress*, *stable*, and *consolidated.* Another mechanism allows the sequential versioning of the other nodes of the object schema (Design, ViewGroup, and View), according to changes made to attributes (Ports and UserFields) defined at these nodes. The system maintains the correspondence between ViewStates and versions of their ascendant nodes, thus linking each ViewState to the inherited attributes that were valid at the time of its creation.

3 Configurations in the STAR framework

In the STAR framework, a *configuration* is defined for each ViewState as a selection of a particular version of a particular design object for each sub-object within this ViewState. The three basic types of configurations already defined in the introduction (static, dynamic, and open configurations) are supported by the framework. Open and dynamic configurations must be *resolved* to a complete reference before the ViewState is used by certain design tools.

In the STAR context, the general definitions of static, dynamic, and open configurations must be refined. A static configuration selects, for each sub-object, a complete reference Design - ViewGroup - ... - ViewGroup - View - ViewState. Dynamic configurations select partial paths, containing at least a reference to a Design, but not reaching the ViewState level of the schemata of the referenced objects. The references may thus reach Design, ViewGroup, or View nodes of the schemata. In fact, in a general sense STAR configurations are always intrinsically dynamic, since the system always selects the *current* version of the Design, ViewGroup, and View nodes of the object schemata.

In order to implement open configurations, the STAR framework supports the definition of local *Components* within a given ViewState, in an approach which is similar to the VHDL language. Components are defined through their interface and parameters, and may be instantiated as sub-objects (*DesignInstances*, in the STAR terminology). Binding of Components to other design objects may be done later, when the configuration is resolved, although a reference (a complete or partial one) may be already established within the ViewState which contains the Component. Components are a generalization of DesignInstances: when there are many sub-objects of the same type, they make reference to one Component, which instantiates the common object. The same Component can be referenced by many DesignInstances.

A configuration of a design object X is presented in Figure 2. The configuration is attached to the representation defined by the path ViewGroup XVG1 – View XV1 – ViewState XVS2 in the object schema of X. XVS2 is composed of two sub-objects, the DesignInstances DI1 and DI2. DI1 references the design object MD, and the user selected the path MD - MV1 - MVS1 in the object schema of MD, while DI2 references the design object N, and the user selected the path ND – NVG1 – NV1 – NVS1 in its object schema. For each selected node (Design, ViewGroup, View) in these paths, the system automatically chooses the current revision. This is represented in the figure by hachured objects over white objects.

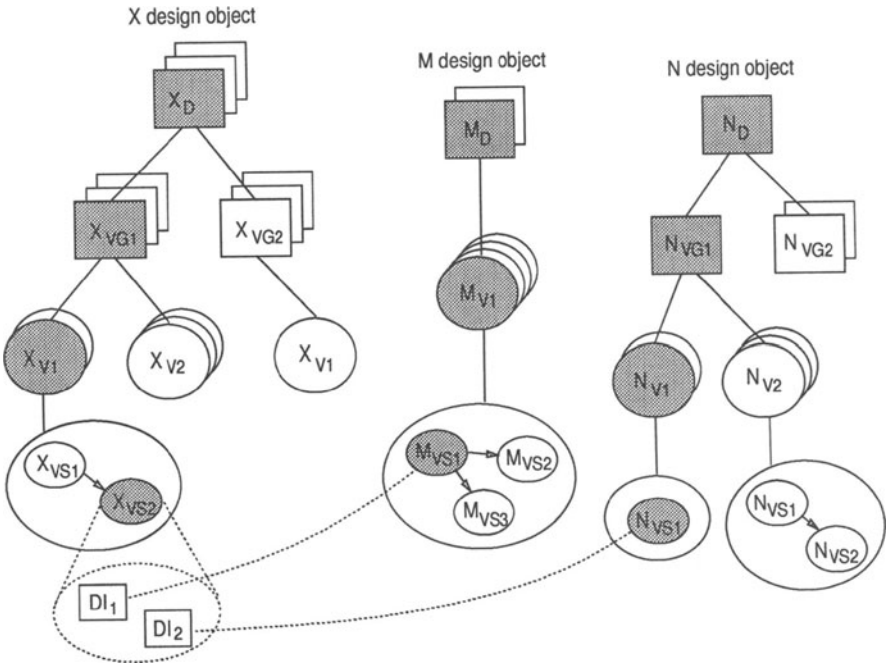

Figure 2: An example of a configuration

In VHDL, components are a mandatory support for the definition of sub-objects, even in the case of static configurations. The STAR framework offers two options: DesignInstances may instantiate either Components or directly other design objects (reaching the level of Designs, ViewGroups, Views, or ViewStates). The second alternative is more convenient for static configurations, when the object contains a small number of similar sub-objects, something that is likely to occur at more abstract design levels.

Each DesignInstance and Component has a *reference attribute* to instantiate design objects. This attribute can be defined (partial or complete references) or left undefined (no references). As in VHDL, in STAR an open or dynamic configuration may be resolved through a separate database object, called *ConfigurationBody*, which is non-versionable. ConfigurationBodies are bound to ViewStates, and one ViewState can have many ConfigurationBodies attached to it.

A ConfigurationBody completes the partial references down to ViewStates of the schemata of the objects that have been selected for the sub-objects of an object X. ConfigurationBodies may also reference already established ConfigurationBodies of these ViewStates, in a hierarchical fashion. ConfigurationBodies have a status (in progress, stable, or consolidated) which depends on the status of the referenced ViewStates.

ConfigurationBodies do not need to complete the references of all sub-objects of the

object X. They may be use to specify configurations for a subset of these sub-objects, so that a complete configuration of X may result from a combination of various ConfigurationBodies.

4 Configuration manager

The *configuration manager* is a special module of the STAR framework. It is built on top of the basic data handling system and of the version manager and may be directly accessed through the framework cockpit [11]. Its goal is to support the resolution of dynamic and open configurations both through an interactive user interface and an application programming interface. The configuration manager allows the user to combine various partial ConfigurationBodies, that cover a subset of the sub-objects of the target design object, with interactive selections for the remaining sub-objects.

The configuration manager offers a repertory of operations to make possible the definition, manipulation, and resolution of configurations:

- resolve a configuration and optionally save the result in a ConfigurationBody;
- create a ConfigurationBody;
- modify a ConfigurationBody, by changing, adding, or deleting referenced objects;
- remove a ConfigurationBody, if it is not being referenced by another configuration;
- search for the existing ConfigurationsBodies for a given ViewState; and
- copy a ConfigurationBody, either to modify the copy without affecting the original ConfigurationBody or to define the same configuration for a new ViewState.

The configuration manager supports three different resolution modes: *manual*, *automatic*, and *semi-automatic*.

The manual mode is supported by an interactive database browser. The user initially selects a ViewState for which the configuration is to be resolved. For each DesignInstance of this ViewState, the user interactively selects a particular leaf of the object schema of the referenced object. This process is repetead for each ViewState selected along the object composition hierarchy. If there are ConfigurationBodies bound to a given ViewState in this hierarchy, the user can select one of them in order to avoid building a complete configuration for the whole sub-hierarchy below this ViewState.

In the automatic mode, the user chooses both the initial ViewState and one of two possible selection criteria: *current* or *most recent* ViewState. This criterion is used by the configuration manager to automatically select ViewStates for the sub-objects in the object compostion hierarchy. The current or most recent ViewState of each design object is pointed by a cursor which can be interactively moved via the browser. The automatic mode with selection of the current ViewState is the default resolution mode.

In the semi-automatic mode, the resolution is supported by a selection language that permits the expression of user-defined criteria based on desired object qualities.

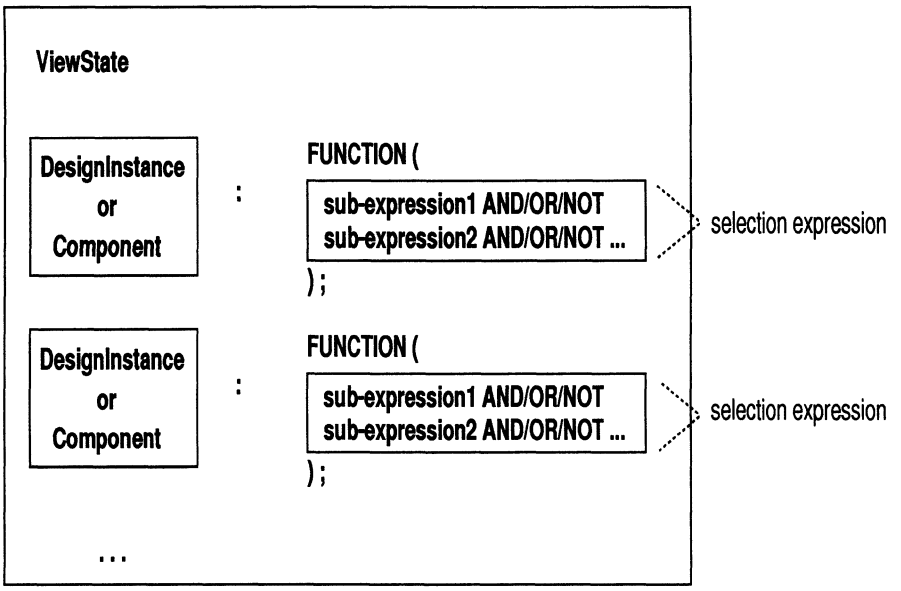

Figure 3: An example of use of the selection language

The selection language (see example in Figure 3) allows the user to specify *configuration expressions* for the Components (or DesignInstances) of the ViewState for which a dynamic configuration is to be resolved. The expressions may be attached to each Component (or DesignInstance), to groups of Components (or DesignInstances), or to all Components (or DesignInstances) of the ViewState.

An expression is composed of a set of sub-expressions. Each sub-expression specifies conditions on properties the ViewStates must show in order to be selected. The evaluation of each sub-expression restricts the set of candidate ViewStates. The simultaneous evaluation of all sub-expressions results in a final set of candidate ViewStates that show all the desired properties. In order to restrict this final set to a single ViewState, the user must either define a conflict resolution function (which specifies the selection of either the current or the most recent ViewState), or switch to the manual selection mode, or redefine the configuration expression. If the final set is empty, the resolution fails.

The sub-expressions are built as logical connections (through AND, OR, and NOT operators) of *logical factors*. Factors are expressed by the following special operators:

- *relational operators*, which compare object attributes to constant values;
- MAX (or MIN), which selects the version with the maximum (or minimum) value of an attribute;
- *existencial operators*, which ask for the existence of attributes (possibly inherited ones);

- LAST (or FIRST), which selects the last (or first) version of a Design, ViewGroup, or View; and
- CURRENT, which selects all current ViewStates associated with a Design, ViewGroup, or View.

5 Related work and comparison

As in the SDE environment [5, 12], the STAR configuration manager offers configuration expressions as a means of specifying user-defined constraints to solve dynamic and open configurations. The SDE environment supports four types of constraints:

- *performance* constraints specify restrictions on attributes like size, delay, power consumption, and latency;
- *selection* constraints are restrictions on the selection of cells as components;
- *environment* constraints specify the environment where a cell can be used as a component of a larger design (e.g related to a cell fanout); and
- *relativity* constraints are dependencies of cells on other cells;

Other systems, such as the OCT manager [3] and the Version Server [1] use *workspaces* (called *layers* in the Version Server) to organize versions that may be used together in a meaningful configuration. In these systems, the user is directly responsible for assigning versions to workspaces (or layers). These mechanisms are directly comparable to the *selection* and *relativity* constraints of the SDE environment. They do not allow the expression of complex queries to solve configurations.

The *selection language* of the STAR framework, in turn, allows for the specification of expressions that are directly comparable to the *performance* and *environment* constraints of the SDE environment. The assignment of adequate attributes to the various nodes of the STAR object schemata allows for an "emulation" of the SDE *selection* and *relativity* constraint types.

In the SDE environment, constraints are embedded into a VHDL extension and are thus expressed within the design object descriptions. In the STAR framework, the configuration manager is a separate module offering a general-purpose service for resolving dynamic configurations. Design tools and descriptions are not affected by this service. If desired, design tools may access it through the API of the configuration manager.

Configuration management in the STAR framework has very particular requirements. The hierarchical object schemata and the possibility of assigning Ports and other general-purpose attributes to any of their nodes make dynamic configurations and their resolution more complex, but give an extra modeling flexibility.

6 Final remarks

The configuration manager is a separate module of the STAR framework, which offers integrated, extensive, and flexible mechanisms to define, solve, and handle design object

configurations. These mechanisms are implemented through both an interactive user interface and an application programming interface. The main features of the configuration manager are:

- support to static, dynamic and open configurations, so as to define configurations according to the current development of the design object or to the abstraction level which is desired for a given object representation;
- the definition and storage of ConfigurationBodies and its reutilization, also in a hierarchical way, within other configurations;
- the resolution of dynamic and open configurations in one of three different modes (manual, automatic, and semi-automatic);
- a selection language that allows the definition of configuration expressions to select objects according to object features, so as to establish different semantic configuration constraints; and
- an interactive selection of object representations, by means of a navigation in the object schemata via an interactive database browser.

References

[1] R.H.Katz et al. Design version management. *IEEE Design & Test of Computers*, February 1987.

[2] IEEE. *IEEE Standard VHDL Language Reference Manual*, 1988.

[3] M.Silva et al. Protection and versioning for OCT. In *26th Design Automation Conference*. ACM/IEEE, 1989.

[4] E.Siepmann. A data management interface as part of the framework of an integrated VLSI design system. In *Intern.l Conference on Computer Design*. IEEE, 1989.

[5] M.J.Chung and S.Kim. The configuration management for version control in an object-oriented VHDL design environment. In *Intern. Conference on Computer Aided Design*. IEEE, 1991.

[6] S.Banks et al. A configuration management system in a data management framework. In *28th Design Automation Conference*. ACM/IEEE, 1991.

[7] F.R.Wagner, L.G.Golendziner, and M.R.Fornari. A tightly coupled approach to design and data management. In *EURO-DAC*. 1994.

[8] H.G.Ribeiro. A Configuration Manager for the STAR Framework. Master thesis, UFRGS, Porto Alegre, 1993.

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

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

[11] R.Mello, L.G.Golendziner, and F.R.Wagner. The visual interface of the STAR framework. Also presented at the *4th IFIP Intern. Working Conference on EDA Frameworks*, 1994.

[12] S.Kim and M.J.Chung. A constraint-driven approach to configuration binding in an object-oriented VHDL-CAD system. In *10th IFIP International Symposium on Computer Hardware Description Languages and Their Applications.* North-Holland, 1991.