59383-0

DESIGN METHODOLOGY MANAGEMENT IN THE STAR FRAMEWORK

por

Flávio Rech Wagner
Arnaldo H. Viegas de Lima
(IBM Brasil - Centro Científico Rio)

RP-170 Dezembro/91

Trabalho realizado com o apoio do CNPq e da IBM Brasil

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Av. Bento Gonçalves, 9500 - Agronomia
91501 - Porto Alegre - RS - BRASIL
Telefone: (0512) 36-8399/39-1355 - Ramal 6161
Telex: (051) 2680 - CCUF BR
FAX: (0512) 244164
E-MAIL: PGCC@VORTEX.UFRGS.BR

Correspondência: UFRGS-CPGCC
Caixa Postal 15064
91501 - Porto Alegre - RS - BRASIL

Editor: Ricardo Augusto da Luz Reis (interino)

Informática - SBU
CAD: Sistemas digitais
Ambiente: Projeto
Gerência: metodologia: Projeto
CNPq: 1.03.03.00-6

Nº CHAMADA FL 2151		Nº REG.: 36125 DATA: 24/02/92
ORIGEM: D	DATA: 14/01/92	PREÇO: Cr$ 40.009,00
FUNDO: CPGCC	FORN.: CPGCC	

Design Methodology Management in the STAR Framework

Abstract

The design methodology management (*dmm*) model of the STAR framework is presented. As opposed to other approaches, where *dmm* is based only on design flow control, in STAR it is achieved through a coupling of task flow control and design data representation features, thus obtaining both design guidance and automatic, methodology-oriented data consistency. The core of a STAR design methodology are *control structures* that hierarchically organize object alternatives, views, and revisions according to design strategies that are specific for each object. Design methodologies can be derived in a hierarchical way, through specialization of the control structures. The task flow management follows a condition-driven approach.

Keywords
Electronic design automation. Design frameworks. Design methodology management. Design flow management.

Resumo

O modelo de gerência de metodologias de projeto do ambiente STAR é apresentado. Ao contrário de outras abordagens, onde esta gerência é baseada apenas no controle do fluxo de tarefas, no STAR ela é alcançada através de uma combinação do controle do fluxo de tarefas com aspectos de representação de objetos de projeto, obtendo-se assim tanto condução do projeto como uma consistência de dados automática e orientada à metodologia. O núcleo de uma metodologia de projeto STAR são *estruturas de controle* que organizam hierarquicamente as alternativas, vistas e revisões dos objetos de projeto de acordo com estratégias que são específicas para cada objeto. Metodologias de projeto podem ser derivadas de um modo hierárquico, através da especialização das estruturas de controle. A gerência do fluxo de tarefas segue uma abordagem dirigida por condições.

Palavras-chave
Automação do projeto de sistemas eletrônicos. Ambientes de projeto. Gerência de metodologias de projeto. Gerência de fluxo de tarefas.

Design Methodology Management in the STAR Framework

1 Introduction

The main objective of CAD frameworks is to provide means for building specific environments that are oriented towards different architectures, technologies, or design methodologies. A framework must allow the integration of tools from distinct sources, aiming at design data consistency and user interface uniformity, while offering design methodology management and data management facilities.

One of the most important features expected from design frameworks is a unified data model for the representation of complex design objects. Most of the proposed or existing frameworks and data models allow a correct representation of complex objects [1,2,3,4]. However, they do not offer flexible mechanisms for representing the variety of possible relationships between all dimensions of the design evolution process, such as views at various abstraction levels, hierarchically related alternative design solutions, and revisions for each alternative/view.

A mechanism for design methodology management is among the most important resources to be offered by design frameworks. A design methodology is a set of design rules that either enforce or guide the design activities performed by the user, so as to obtain design objects with desired properties. Many papers have described alternative approaches for implementing design methodology management mechanisms in recent years [5,6,7,8,9]. These approaches mainly rely on the management of the task flow, offering facilities for task selection, task sequencing (with conditional branching and iteration mechanisms), and automatic task execution and backtracking. Some systems [5,6,7] enhance the task flow management with knowledge about design constraints, goals, tools, and data. However, tools operate on isolated files, so that these systems do not offer the automatic data consistency that may be achieved by a unified data model.

The STAR framework implements a powerful and flexible data model, based on the GARDEN model, that supports the management of the diverse representations created for the design objects during the design process [10]. The data model allows the organization of these representations in a hierarchical structure, which can adequately describe the relationships between the design evolution alternatives.

The flexibility of the STAR data model is the foundation for a design methodology management model which is based on a coupling between the task flow control and the management of the design representations and not only on the task flow control. The model relies on the possibility of defining, for each design object, a specific organization for its various design representations, reflecting both the descriptions created by the particular tasks to be executed for the object design and the relationships between them. STAR supports a condition-driven task flow model, based on the specification of input / output relationships between tasks and design object properties. The model also offers the possibility of specialization of design methodologies in a hierarchical way. The combination of these features, which is not found on other systems, offers not only design guidance, but also automatic methodology-oriented data consistency.

The rest of the paper is organized as follows. Section 2 reviews the STAR data model and briefly discusses its advantages regarding other systems. The model for design methodology management is described in detail in Section 3. An example which illustrates its main features is presented in Section 4. Section 5 compares the STAR concepts for design methodology management with other approaches. Finally, section 6 summarizes the main points of the STAR design methodology management model and draws some conclusions.

2 The STAR data model

One of the key features of the STAR framework is a powerful data model, that is not affected by the design methodology nor enforce a particular one. This model is strongly based on the GARDEN data model [10]. It provides a flexible way for supporting complex design objects and for managing the various representations created along the dimensions of the design evolution (*alternatives*, *views*, and *revisions*). This feature allows the system to implement, according to user- or methodology-defined criteria, many different conceptual schemes for representing the design evolution. A direct result of such strategy is the ability exhibited by the system in modeling and controlling different design methodologies in a natural and efficient way.

Control structures — In the STAR data model, all design objects are placed in a single **Repository**, divided into **Libraries** of **Design** objects. Each Design can contain an arbitrary number of **ViewGroups** and **Views**. The ViewGroups define simple grouping objects that may also contain any number of ViewGroups and Views, building a tree-like hierarchical *control structure*, as shown in Figure 1.

ViewGroups are nothing more than *groupings* of objects, gathering other ViewGroups and Views according to user- or methodology-defined criteria. The fact that Views can be defined at any level of the control structure offers an unlimited number of ways for organizing different descriptions of the Design. As the system does not enforce any grouping criterion, it is left to the user or to the design methodology to decide how the Views will be organized.

The actual descriptions of a design object are stored at the Views, that are the leaves of the tree structure. STAR supports three types of Views: **HDL**, for behavioral representations of the design object (e.g. in any hardware description language), **Layout**, used for symbolic, schematic, and mask level descriptions, and **MHD** (Modular Hierarchical Descriptions), for modular representations, composed by sub-objects that are instances of other objects. While HDL and Layout Views can also contain sub-objects, they do not handle the exact interconnections between them, as in the MHD Views. As opposed to the MHD Views, they contain a file where specific design data, handled as a bit string by the STAR DBMS, is stored.

The time evolution of the design object is expressed at the View level as an acyclic graph of **ViewStates**. STAR also provides a simple **TimeStamp** mechanism to control the

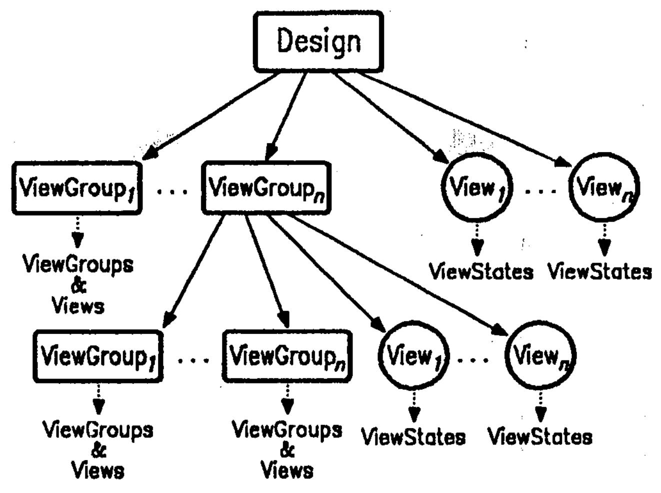

Figure 1: The STAR data model – primary control structure

evolution of objects placed outside the ViewState scope (e.g. interface pins and attributes defined at the Design and ViewGroup nodes).

Other features — **Processes**, that gather technological related information, can be attached to Libraries, Designs, ViewGroups, or Views. The attachment is inherited by all objects in the control structure that are descendants of the node where the Process is attached to.

Another characteristic of the model is the support for multiple interfaces for a design object. Instead of enforcing that interface **Ports** must be defined at a particular node of the control structure, such as Design or View, the model allows Port definitions at any level of the structure (Design, ViewGroup, View, or ViewState). Since the control structure can be depicted as a tree, the actual interface presented at any node is the composition of all Ports defined along the path from the root (Design) to the node. Port attributes can also be added at any node of the hierarchy. There is no inheritance of Port characteristics from ViewState to ViewState, only directly from the View to each ViewState.

DesignInstances (sub-modules inside a View) can be occurrences of **Components**, defined inside the same View. A reference to another object can be done through a Component, so that it is valid for all its DesignInstances, or through each DesignInstance. It is possible to reference Designs, ViewGroups, Views (*dynamic* configurations [11]), or ViewStates (*static* configurations). It is also possible to let Components and DesignInstances without references, thus building *open* configurations, as in VHDL [12]. Dynamic and open configurations must be later resolved through special configuration descriptions. A *configuration manager* allows the user to resolve configurations according to object attributes, as in [13], selecting specific ViewStates of certain Designs for the Components or DesignInstances.

Objects can also be grouped together according to user- or methodology-defined criteria using the **Correlation** concept. The Correlation can be viewed as an extension of the *equivalence* concept [11], with a generic grouping criterion and support for gathering heterogeneous objects such as Designs and ViewStates.

Each STAR object may contain an arbitrary number of **UserFields** (attributes), that can hold any type of information and can contain another collection of UserFields. UserFields defined at a particular node of the control structure can be optionally inheritable by the node descendants. STAR design objects can be also parameterized.

Auxiliary Objects, that are necessary for specific applications, may be also defined. Examples are stimuli and result files, for simulation, technology files, for design rule checking, and testability measures, for testability analyzers.

Model analysis — The ViewGroup concept allows the definition of hierarchical control structures that, combined with the Port inheritance mechanism, presents two important properties. First, ViewGroups can be thought as design alternatives, grouping all representations that correspond to a given design decision and storing the attributes and interface pins that are common to them. The data model thus guarantees that all representations related to a given design alternative hold these common properties. Second, ViewGroups can be used to build a hierarchy of design levels, so that a design alternative can be appended to the abstraction level where it makes sense.

The properties of ViewGroups and Views, combined with the Correlations and the Port and UserField inheritance, build a powerful mechanism for creating different organization strategies for the object representations (alternatives, views, and revisions). The flexibility achieved with these properties supports control structures that are not available with other data models. It also allows the mapping to STAR of a variety of control structures [10].

We call the general STAR data model, as shown in Figure 1, a **primary** control structure, while control structures built on top of it are **secondary** ones. For example, a VHDL-like secondary control structure is simply achieved by mapping *architectural bodies* to Views, by directly linking these Views to the Design (the root of the control structure), without using ViewGroups, and by storing the interface definition at the Design. An Oct-like secondary control structure can be obtained, as shown in Figure 2, by mapping an Oct-View OV_i into a STAR-ViewGroup VG_i, while the Contents Facet of OV_i is mapped into both a unique ViewGroup VG_{cf} under VG_i and a View V_{cf} under VG_{cf}. VG_{cf} stores the basic object interface definition, while V_{cf} stores the Facet contents. Other Interface Facets of OV_i are mapped into STAR-Views $V_{i,j}$ under VG_{cf} and can add new interface attributes.

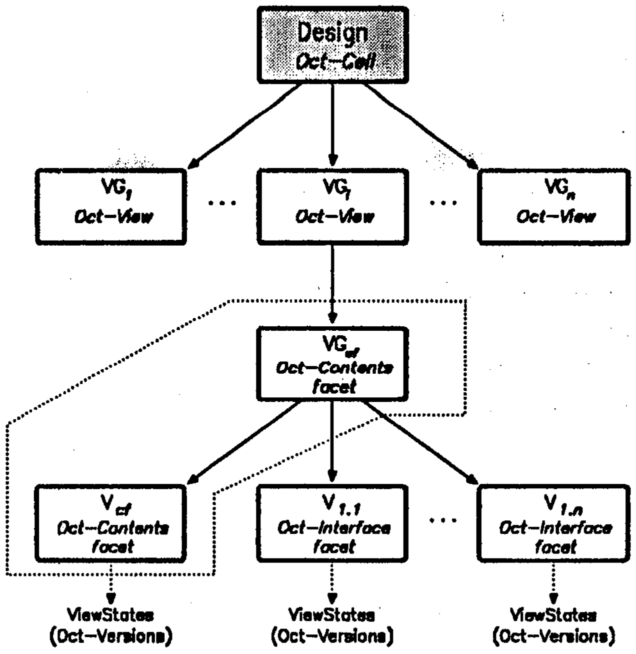

Figure 2: Mapping the Oct control structure to STAR

3 The STAR model for design methodology management

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

- tasks that must be executed when the design process arrives at a given state (this state can be for instance expressed in terms of some design object properties);
- alternative design approaches that must be followed from a given design state, as well as criteria for deciding between the possible design paths (again, these criteria can involve design object properties);
- design representations that must be created under given conditions, e.g. a representation at a more detailed design level or alternatives that must be compared according to some trade-offs.

Design methodology management is the control of the creation of the design object representations and of the execution of the required design tasks so that the objects and tasks conform to the established rules.

The definition of a design methodology in the STAR framework is based on three main principles: the definition of control structures for the design objects (a *conceptual scheme* for an application), the task specification, and the hierarchization of the design methodologies.

Conceptual scheme — A design methodology is primarily based on secondary control structures, that organize all representations that can be created for the design objects of a given application. Each design object can have a different control structure, depending on the particular design strategy to be applied to it.

The most important aspect of a control structure is its overall topology, i.e. how ViewGroups and Views are hierarchically organized. This topology is defined by criteria for organizing design representations, that can limit the number or types of ViewGroups and/or Views to be placed under a given ViewGroup or under the Design.

A conceptual scheme can include UserField specializations for the object control structures (area, power, geometric attributes, view types, etc), definitions of Auxiliary Objects (test vectors, testability measures, simulation results, etc), which in turn can also have their own UserFields, and Correlations (equivalence relationships, relationships between Auxiliary Objects and ViewStates from which they have been created, etc).

Restrictions can refer to Ports, configurations, and Processes. A methodology can decide at which nodes of the control structure Ports are to be defined. In the Oct-like control structure of Figure 2, for instance, Ports can be defined only at the Views V_{cf} and $V_{1.n}$. A methodology can also restrict the possible configurations of a DesignInstance, avoiding dynamic configurations, for instance. Finally, a methodology can specify a particular association of Processes to other objects. It can be decided, for instance, that Processes are always attached to Designs, so that all ViewGroups and Views of each Design have the same Process, but different Designs of a same Library may have different Processes.

Task flow — Task flow is expressed through a condition-driven model, where input-output relationships between tools and design data properties are specified. These relationships can involve "qualities" of the design objects, such as the values of their attributes. These qualities must be explicitly defined in the control structures for these objects. A tool is eligible for execution when its input data, with the desired qualities, is available. The choice among many executable tools is left to the user.

Each task is specified by a 5-uple { *task name*, *tool name*, *input conditions*, *output objects*, *task goals* }. The same tool can be used in many different tasks. The *input conditions* list the objects that must exist in order that the task is eligible for execution, eventually specifying properties that these objects must additionally hold. The *output objects* list object representations, Auxiliary Objects, and Correlations that are created by the task. *Task goals* describe the properties expected from the objects after the task execution. If the goals are not achieved, the task "fails", though the output objects are still created. A later execution of the same task can succeed, if the input objects have changed (either new versions have been created for them or new configurations have been selected) or if properties from other objects, used in the specification of the task goals, have changed due to the execution of other tasks. A design methodology has achieved its overall goals if all its tasks have been executed and none of them is marked as "failed".

This task flow model thus does not impose particular design sequences in order to meet design constraints and achieve design goals, but rather let to the user expertise the choice of the best solutions to particular problems faced during the design process.

Hierarchies of methodologies — Design methodologies can be organized in a hierarchical way. A new design methodology can be derived from a previous one:

- by specializing (either by extending or restricting) the control structures of the previous methodology, i.e. by building new secondary control structures from already existing ones;
- by defining new Design objects (and their control structures), Auxiliary Objects, and Correlations;
- by adding new tasks.

Once a control structure topology has been restricted by a given design methodology, a new secondary control structure to be derived from it cannot violate the restriction rules. It cannot neither remove already defined ViewGroups and Views, but it can add new ViewGroups, Views, and attributes.

A design task is defined at a given level of the methodology hierarchy. The task definition must be therefore consistent with all properties and restrictions of the design methodology to which it belongs, handling objects, object representations, and attributes defined by the methodology.

Users are also related to the methodology hierarchy. A user is constrained by a methodology to the execution of a given set of design tasks and to the manipulation of a given set of design object representations. However, he/she may have the power for defining new design methodologies by defining new objects, deriving new secondary control structures from the methodology in which he/she has been placed, defining new tasks, and integrating new tools. Because of this possibility, it does not make sense to distinguish "users" from "application managers" in the STAR framework. Each user is potentially also an application manager, which defines his/her particular design environment.

Tool integration — Although tool integration is not a function of design methodology management, both concepts are intimately related. When an already existing tool is integrated into a design environment, its design data must be mapped into the object representations and attributes of the control structures specified for that environment. The tool can in fact manipulate its design data in the most appropriate particular format, letting the mapping into the STAR objects to a separate module. However, a new tool can be already oriented towards a conceptual scheme implemented on the STAR framework, so that it makes direct access to the STAR objects and attributes.

A tool which directly handles a given set of STAR objects and attributes that are defined in a specific design methodology M_i can be used not only in tasks defined inside M_i, but also in other methodologies that have been derived from M_i. However, this tool cannot be used in tasks defined inside other methodologies where objects and attributes it handles are not known. The use of an external mapping allows a tool to be used in tasks defined in different design methodologies.

4 Example of a design methodology

We show the application of the STAR design methodology management model in the physical design of a 32-bit RISC microprocessor, which is composed of two main blocks, the Operational Block (OB) and the Control Block (CB). Full-custom and cell-based approaches are considered for the Operational Block, which is designed around two data busses. In the full-custom design, the Operational Block layout is designed as an array of 32 horizontal 1-bit wide slices. Each slice contains 1-bit wide cells for modules like the ALU, a register file (RFile), a barrel shifter (BS), the Program Counter (PC), etc.

In the layout design of the operational block slice, the ALU is the critical module, since it has the largest area and delay. Figure 3 shows the control structure for the ALUSlice object, whose main features are:

- The root contains Ports for the data busses A_Bus and B_Bus and for the control lines. All of them are 1-bit wide signals and have an attribute PortDirection.
- The structural representation as an interconnection of basic transistors and gates is stored in the MHD View V-ALUSlice-Struct.
- The ViewGroup VG-ALUSlice-LO gathers all layout related representations of ALU-Slice. It defines attributes Height, Width (cell dimensions), PowerBusWidth, PowerConsumption, and MaxDelay. The values of these attributes come from previous design steps and are used as design constraints for the ALU slice layout design. The ViewGroup also adds Ports for VDD and GND, defines an additional line A_Bus_not (the same for B_Bus), since the operational block layout is designed with two complementary bus lines, and adds new Port attributes for the layout design. Because of the overall topology, the data bus lines traverse the cell in the horizontal direction, and so have right and left coordinates, while the control lines, which traverse the cell in the vertical direction, have top and bottom coordinates. These Ports and their attributes are inherited by all representations under the ViewGroup.
- Two alternatives for the ALU slice layout design are compared: manual and automatic synthesis with a module generator. The representations for these alternatives are stored under the ViewGroups VG-ALUSlice-Manual and VG-ALUSlice-ModGener, respectively. The comparison is based on the area, power consumption, and maximal delay of the resulting layouts. VG-ALUSlice-LO has attributes Height and Width (of the cell layout), MaxDelay, PowerBusWidth, and PowerConsumption, that receive values coming from the winning solution.
- Views V-ALUSlice-Manual-LO and V-ALUSlice-Manual-ENL contain the layout and the extracted netlist descriptions for the manual design, respectively.
- The MHD View V-ALUSlice-Extracted is generated by back-annotating to V-ALUSlice-Struct the timing extracted from the layout.

- Equivalence relationships, shown as dotted lines in the figure, are established between ViewStates of representations obtained by synhtesis or extraction from each other.

The design of the ALU slice is composed of many tasks: layout generation, design rule checking, netlist extraction, netlist comparison, electrical parameter extraction and back-annotation, and timing evaluation. These tasks operate upon the ALUSlice representations already shown in Figure 3.

Figure 4 shows the detailed specification of the layout generation task in the manual design, according to the STAR condition-driven task model. Each time this task is executed, a new ViewState for V-ALUSlice-Manual-LO is created, as well as a Correlation between this ViewState and the ViewState of V-ALUSlice-Struct from which it is designed. The only condition for the task execution is the existence of a ViewState of V-ALUSlice-Struct. Eventually, the methodology could specify that a layout should be generated only if the selected ViewState of V-ALUSlice-Struct had some property, for instance an expected value for a testability measure. The task "fails" if the ALU slice height is greater than a desired height defined during the floorplanning process of the overall microprocessor. There is no mandatory task sequence for correcting this problem. A new execution of the task can succeed, either if the floorplan is re-evaluated (so that the design goal is changed) or if a new ViewState for the ALU slice layout achieves the desired height.

Figure 5 shows the control structure for the OB object. All its layout related representations are stored under the ViewGroup VG-OB-Layout. It redefines Ports and adds new Port attributes, as VG-ALUSlice-LO. Representations for the full-custom and cell-based alternatives are gathered in the ViewGroups VG-OB-FullCustom and VG-OB-CellBased, respectively, defined under VG-OB-Layout. VG-OB-FullCustom defines attributes Width, Height (both receive values that result from designing the layout as an abutment of 32 1-bit wide almost identical slices, each one containing modules ALU, RFile, BS, PC, etc), and PowerBusWidth. This is the width of the VDD and GND lines, initially set to a value to be used as a constraint by the slice layout design, but which can be re-evaluated after that.

Figure 6 shows the control structure for the Control Block (CB) object. It must be noted that it is significantly different from the control structure for the OB object, since the Control Block layout design follows a distinct design methodology, where tasks are for instance state assignment, logic minimization, technology mapping, and layout generation. Random logic and PLA design alternatives are considered. As opposed to the OB control structure, there is no ViewGroup gathering all layout representations for the CB object, because it was decided to group, for each of the alternatives, the behavioral, structural, and layout representations under a unique ViewGroup (VG-CB-RandomLogic or VG-CB-PLA, respectively), since they have common properties.

The complete microprocessor design can be defined as an hierarchy of design methodologies. A chief designer defines the design methodology (and so the control structure) for the overall microprocessor. First features of the OB and CB objects are also defined at this point, including Ports of their structural Views and main attributes. New methodologies, defined by the designers responsible for these two objects, can then refine the respective control structures. In the case of the OB design methodology, new objects are defined, such as the ALU slice. Some of their basic characteristics are also already defined, such as properties of their structural representations. A more refined control structure for the ALU slice layout design may belong to yet another design methodology. A hierarchy of methodologies is thus a natural way of thinking about the microprocessor design. Each one specializes already existing control structures and defines control structures for new objects.

From this example, we can conclude that:

- A hierarchy of ViewGroups (as for the ALU and the Control Block) can correctly express the evolution of the design decisions taken during the design process. Each ViewGroup adds UserFields, Ports, and Port attributes that are specific of the corresponding design decision. The decision hierarchy can be strongly related to a hierarchy of design abstraction levels.

- The design methodology management is closely related to the management of the design representations created during the design process.

- As a consequence of the previous point, secondary control structures that are specific for different design objects (such as Operational Block, ALU slice, and Control Block) are needed for specifying and controlling particular design methodologies for those objects.

- Design methodologies can be derived from already existing ones, as the designer specializes control structures for already defined objects, creates new design objects, decides their control structures, and specifies the tasks needed for their designs. These methodologies can be hardly completely defined "a priori", so that each designer must be considered as an application manager for a derived methodology.

- A "natural" task model specifies which are the conditions (object representations that must exist and/or object properties that must hold) that enable a task execution, without indicating mandatory task sequences.

5 Comparison with other approaches

A comparison between different approaches to design methodology management can be best understood by making it clear that this framework function must not only provide a mechanism for sequencing the design tasks, but rather must also guide or enforce the design process in order to meet the design constraints and to achieve the desired design goals, while maintaining the overall data consistency.

Four basic approaches to design methodology management can be identified:

- only task flow control

- task flow control enhanced with design expertise
- controlling the creation of design representations
- coupling the task flow management with the control of the creation of object representations

In the first approach, there is only a mechanism for sequencing the design tasks. It may contain capabilities for the definition and automatic execution of task sequences, including conditional branching and iteration, and for storing and repeating user-defined "ad-hoc" sequences. The CFI approach [14] follows this reasoning. Since a "pure" task flow control does not help meeting the design constraints and achieving the design goals, it should not in fact be classified as a "design methodology management" approach.

In the second approach, which is followed by the Ulysses [5] and Cadweld [7] systems, as well as in the ADAM Design Planning Engine [6], the task flow control is enhanced with knowledge about the design constraints, goals, tools, and data. This knowledge provides two basic capabilities:

- automatic tool selection, by identifying alternative tasks that can be executed from the current design point and by selecting both the most promising task and tool (if several tools, with different properties, can be chosen for the same task), e.g. based in tool result estimations, as in the ADAM DPE;
- automatic backtracking to previous design points, either to restore design consistency, when design changes occur, or to analyze other alternatives, when constraints cannot be met or goals are not achieved.

These systems are not based on an underlying unified data model, so that tools operate on isolated files. While this allows for an easier tool integration, it prevents the system from supporting an automatic data consistency. The quality of the design depends solely on the completeness of the knowledge representation.

In the third approach, instead of specifying which and when tools must be executed, the system controls the consistency of the objects to be created. This can be done through a unified data model, which can handle composition relationships, configuration management, hierarchies of alternatives, common properties of representations, and other user- or methodology-defined integrity constraints. Although the objects thus automatically hold the desired consistency, the system does not give user guidance to obtain these objects. Tool integration becomes clearly more complex, since a mapping between the tool data and the data model objects is needed.

Finally, an enhanced task flow management can be combined with mechanisms for controlling the creation of object representations, achieving both automatic data consistency and methodology-oriented user guidance. Design qualities are achieved partially by the data model and partially by the task flow control. The knowledge about expected design properties, modelled for the specific purpose of controlling the task flow, is released from the burdening of representing all design consistencies that are already maintained by the data model. Examples are the design manager of the FACE Core Environment [8] and the STAR model. The FACE manager, however, is based on very simple version and task flow control techniques, while the STAR framework supports a methodology-oriented data consistency and a condition-driven task flow.

It must be noted that the STAR approach does not aim at the automatic execution of tasks. Its condition-driven task flow model relies on user expertise, only indicating tasks that are eligible for execution. Therefore, it does not offer facilities for automatic tool evaluation, selection, and execution, neither for automatic task backtracking.

Tool encapsulation, which consists of a mapping between the tool data and the unified data model and an abstraction of tool invocation details, has been presented as a feature of design methodology management [14]. It is clear, however, that tools *still* have to be integrated into the framework even if it does not give any support for design methodology control. Tool encapsulation facilities should be therefore considered as an independent feature of design frameworks.

6 Conclusions

A model for design methodology management in the STAR framework has been presented. As opposed to other approaches, where design methodology management is based only on design flow control, in STAR it is achieved by coupling a task flow control mechanism with flexible and sophisticated features offered by the data model for handling design representations along the various dimensions of the design evolution. The combination of these techniques results in user guidance, while maintaining automatic, methodology-oriented data consistency.

The paper has shown:

- that in STAR a design methodology is mainly defined by a conceptual scheme of the application, which contains *control structures* for the design objects (a control structure organizes the design representations created along the design process);

- that specific *control structures* for different design objects, as well as a hierarchical representation of the design decisions taken for each design object, are necessary for a correct modelling of the design process and very useful in the design methodology management;

- that deriving methodologies in a hierarchical way is also a very useful feature in concrete design situations. The derivation is obtained by specializing *secondary* control structures for already existing objects, by defining control structures for new objects, and by specifying new design tasks.

STAR supports a condition-driven task flow model, where input/output relationships between tasks and design object properties are specified. It does not impose specific design sequences, but rather enable design tasks according to the achieved properties of the design objects, relying on the user expertise for the selection of the most promising task sequence. This task flow model is therefore simpler than models supported by other systems. It does not provide facilities for automatic tool evaluation, selection, and execution.

References

[1] D.S. Harrison et al. Data management and graphics editing in the Berkeley Design Environment. In *International Conference on Computer Aided Design*, IEEE, 1986.

[2] K. Gottheil et al. The CADLAB workstation CWS – an open, generic system for tool integration. In F.J. Rammig, editor, *IFIP Workshop on Tool Integration and Design Environments*, North-Holland, 1988.

[3] L. Claesen et al. Open framework of interactive and communicating CAD tools. In F.J. Rammig, editor, *IFIP Workshop on Tool Integration and Design Environments*, North-Holland, 1988.

[4] R.H. Katz and E. Chang. Managing change in a computer-aided design database. In *International Conference on Very Large Data Bases*, Brighton, 1987.

[5] M.L. Bushnell and S.W. Director. VLSI CAD tool integration using the Ulysses environment. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[6] D.W. Knapp and A.C. Parker. A design utility manager: the ADAM planning engine. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[7] J. Daniell and S.W. Director. An object-oriented approach to CAD tool control within a design framework. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[8] W.D. Smith et al. FACE Core Environment: the model and its application in CAE/CAD tool development. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[9] T. Chiueh and R.H. Katz. Managing the VLSI design process based on a structured history framework. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

[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.H. Katz et al. Design version management. *IEEE Design & Test*, February 1987.

[12] *IEEE Standard VHDL Language Reference Manual.* IEEE, New York, 1988.

[13] S. Kim and M.J. Chung. The constraint-driven design in an object-oriented VHDL CAD. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

[14] K.W. Fiduk, S. Kleinfeldt, M. Kosarchyn, and E.B. Perez. Design methodology management – a CAD Framework Initiative perspective –. In *27th Design Automation Conference*, ACM/IEEE, 1990.

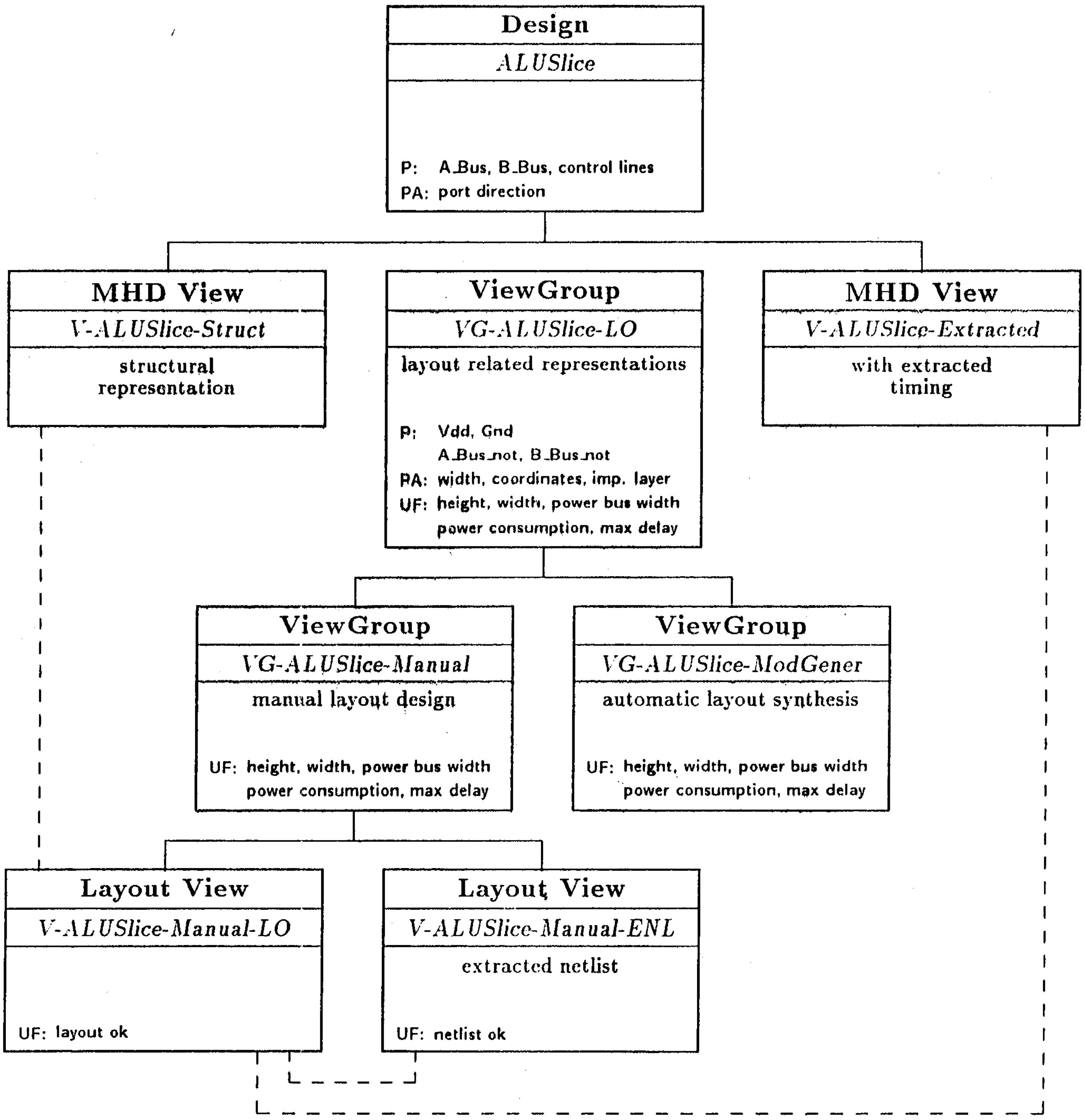

Figure 3: The control structure of the ALUSlice object

```
---------------------------------------------------------------------------
task: ALU slice layout generation
tool: mask-editor [ configuration = 1.5 micra ]
input: V-ALUSlice-Struct
output: ViewState for V-ALUSlice-Manual-LO
        Correlation: ViewState of V-ALUSlice-Manual-LO
                     manually designed from
                     ViewState of V-ALUSlice-Struct
goal: Height =< 1/32 V-Microprocessor-Layout . OB-Part1 . Height
      (OB-Part1 is a DesignInstance within the floorplanning
      description of the microprocessor, corresponding to OB)
---------------------------------------------------------------------------
```

Figure 4: Task specification for the ALU manual layout generation

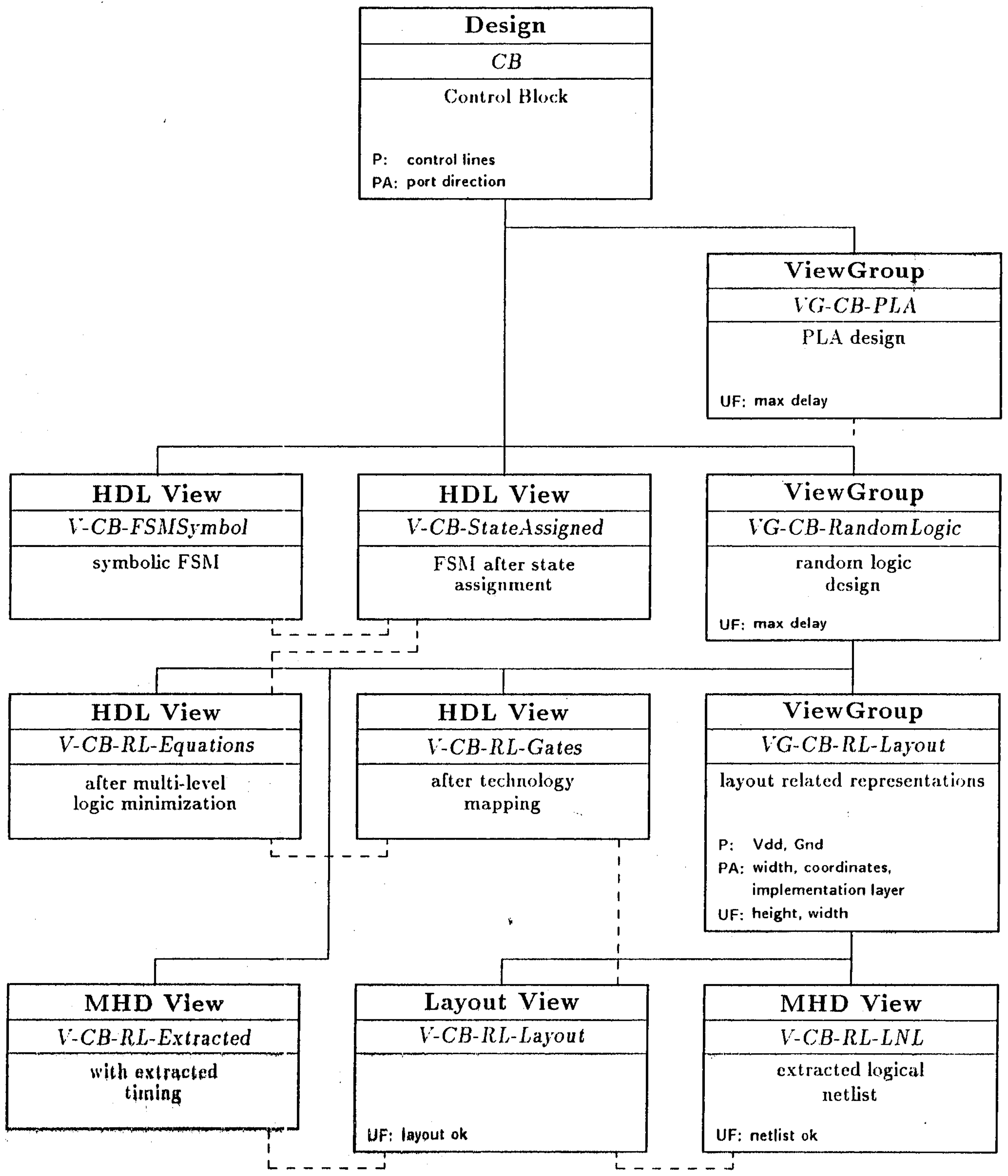

Figure 5: The control structure of the Operational Block object

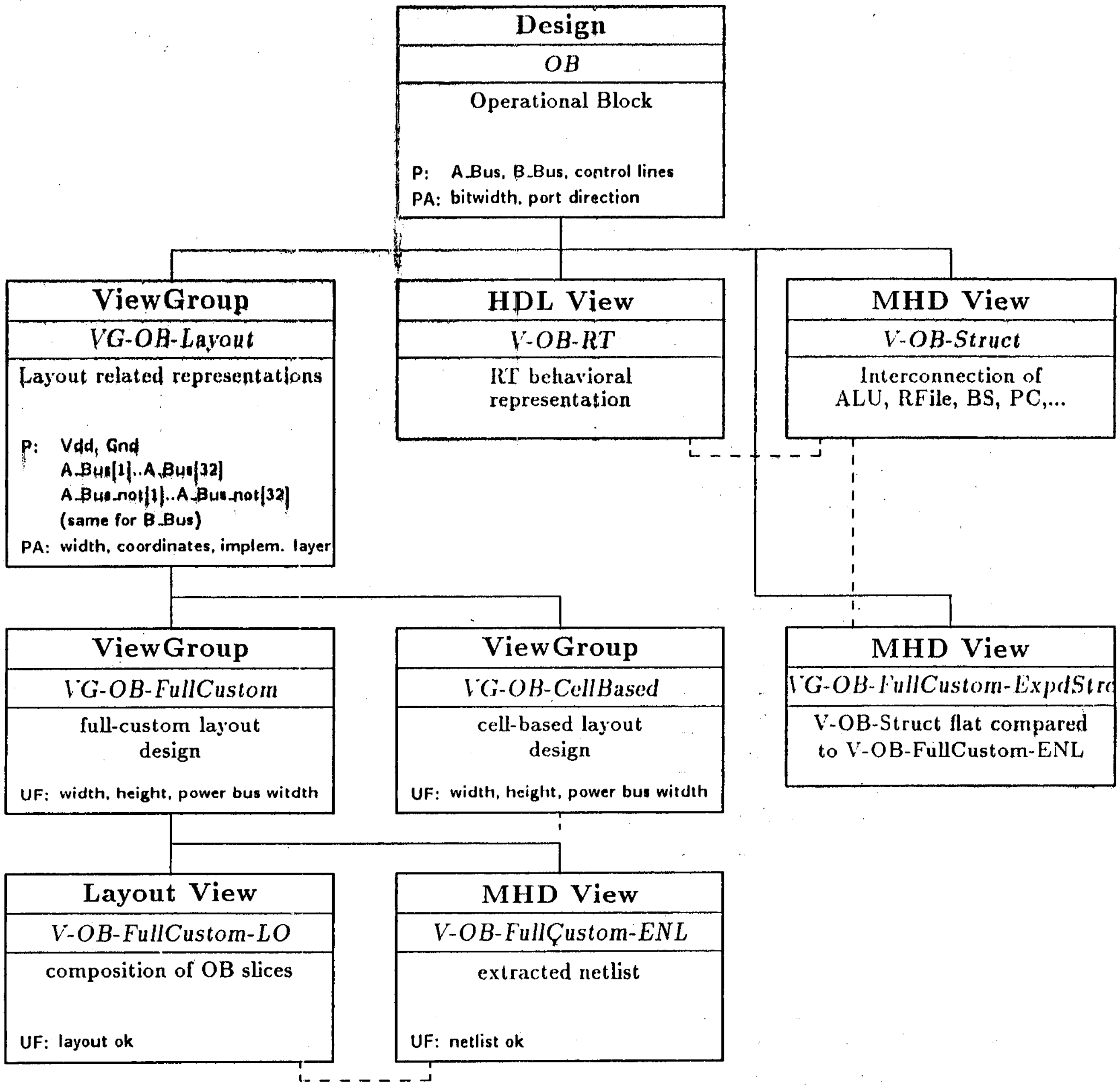
Design
OB
Operational Block
P: A_Bus, B_Bus, control lines
PA: bitwidth, port direction
ViewGroup
VG-OB-Layout
Layout related representations
P: Vdd, Gnd
A_Bus[1]..A_Bus[32]
A_Bus_not[1]..A_Bus_not[32]
(same for B_Bus)
PA: width, coordinates, implem. layer
HDL View
V-OB-RT
RT behavioral representation
MHD View
V-OB-Struct
Interconnection of ALU, RFile, BS, PC,...
ViewGroup
VG-OB-FullCustom
full-custom layout design
UF: width, height, power bus witdth
ViewGroup
VG-OB-CellBased
cell-based layout design
UF: width, height, power bus witdth
MHD View
VG-OB-FullCustom-ExpdStr
V-OB-Struct flat compared to V-OB-FullCustom-ENL
Layout View
V-OB-FullCustom-LO
composition of OB slices
UF: layout ok
MHD View
V-OB-FullCustom-ENL
extracted netlist
UF: netlist ok

Figure 6: The control structure of the Control Block object

Relatórios de Pesquisa

RP-170: "Design Methodology Management in the STAR Framework, dezembro, 1991.
F.R. WAGNER, A.H.V. DE LIMA

RP-169: "Lotos - Language of Temporal Ordering Specification, Novembro, 1991.
M.C. MÓRA; D.J. NUNES

RP-168: "Um Estudo Comparativo de Métodos Intervalares para Avaliação de Funções Racionais", setembro 1991.
A.R. COELHO; L.M. CADORE; G.P. DIMURO; D.M. CLAUDIO

RP-167: "The Data Model of the STAR Framework", novembro 1991.
F.R. WAGNER

RP-166: "Design Methodology Management in Design", novembro 1991.
F.R. WAGNER

RP-165: "Desenvolvimento de "Asa" e "Trama"", outubro, 1991.
S.D. OLABARRIAGA; E.M. CORRÊA; C. CALLIARI; A.L. POMPERMAYER

RP-164: "Estudo Topológico e Elétrico da Nova Célula de Base para CIs Gate Array - Tecnologia 1.2 um", outubro, 1991.
L.R. FROSI; G.V. PAIXÃO; J.L.G. CUNHA; D.A.C. BARONE

RP-163: "Representação de Conhecimento em Engenharia do Conhecimento", setembro, 1991.
N. EDELWEISS

RP-162: "Biblioteca de PADS Digitais CMOS 1.5 um - Versão 1", setembro 1991.
M.K. DOSSA

RP-161: "Versões Intervalares do Método de Newton", agosto 1991.
H. KORZENOWSKI; M. LEYSER; T.A. DIVERIO; D.M. CLAUDIO

RP-160: "Processamento Vetorial e Vetorização de Algoritmos na Máquina Convex C210", agosto 1991.
T.A. DIVERIO

RP-159: "Um estudo de técnicas de validação e de verificação de produtos de software", junho 1991.
N. EDELWEISS

RP-158: "Extensão das Ferramentas PIU/LINUS de especificação e controle de interfaces com o usuário", Junho 1991.
J.P. FIGUEIRÓ

RP-157:	"The Domain of Nets and the Semantic Bases of a Notation for Nets", Abril 1991. A.C.R. COSTA
RP-156:	"Continuous Predicates and Logical Reflexivity", Abril 1991. A.C.R. COSTA
RP-155:	"TENTOS - Gerenciador de Software para Microeletrônica", abril 1991. F.G. MORAES; R.A.L. REIS.
RP-154:	"Sistemas Especialistas para a Engenharia de Software", Abril 1991. H. AHLERT.
RP-153:	"Estudo Comparativo e Taxonomia de Ferramentas de Suporte à Construção de Sistemas, Abril, 1991. H. AHLERT.
RP-152:	"IMP-MAC - Emulador de Impressora Padrão Apple", Abril, 1991. C. DE ROSE; R.F. WEBER.
RP-151:	"Em direção a um modelo para representação de aplicações de escritórios baseadas em documentos", Abril, 1991. D.B.A. RUIZ.
RP-150:	"Implementação de Sistemas de Gerência de Banco de Dados", Abril, 1991. D.B.A. RUIZ.
RP-149:	"Integração de Ferramentas no Sistema AMPLO: Crítica e Proposta de Extensões", março, 1991. F.R. WAGNER.
RP-148:	"Interface de Entrada para Teclado e Mouse", março, 1991. J.M. DE SÁ.
RP-147:	"Servidores - Guia do Usuário - Edição 1", janeiro, 1991. A.R. TREVISAN, C. LEYEN, G. CAVALHEIRO, J.F.L. SCHRAMM, L.G. FERNANDES, P. FERNANDES, R.M. BARRETO, R. TEODOROWITSCH
RP-146:	"Biblioteca de Células TRANCA regras ECP15/1", janeiro, 1991. C. CRUSIUS, L. FICHMAN, M. KINDEL, C. MARCON, R. REIS
RP-145:	"Manual do Usuário do Projeto TRANCA; v 1.0", janeiro 1991. F.G. MORAES; M. LUBASZEWSKI, R.A.L. REIS