DESIGN METHODOLOGY MANAGEMENT IN DESIGN FRAMEWORKS

por

Flávio Rech Wagner

RP nº 166 NOVEMBRO/91

"Trabalho realizado com o apoio do CNPq".

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Av. Bento Gonçalves, 9500 - Agronomia
91501 - Porto Alegre - RS - BRASIL
Telefones: (0512) 36-8399/39-1355 - Ramal 6161
Telex: (051) 2680 - CCUF - BR
FAX: (0512) 24-4164
E-mail: PGCC@VORTEX.UFRGS.BR
Correspondência: UFRGS-CPGCC
Caixa Postal 15064
91501 - Porto Alegre - RS - BRASIL

Editor: Ricardo Augusto da Luz Reis (interino)

UFRGS

Reitor: Prof. TUISKON DICK
Pró-Reitor de Pesquisa e Pós-Graduação: Prof. ABÍLIO BAETA NEVES
Coordenador do CPGCC: Prof. Ricardo A. da L. Reis
Comissão Coordenadora do CPGCC: Prof. Carlos Alberto Heuser
Prof. Clesio Saraiva dos Santos
Profa. Ingrid Jansch Pôrto
Prof. José Mauro V. de Castilho
Prof. Ricardo A. da L. Reis
Prof. Sergio Bampi
Bibliotecária CPGCC/II: Margarida Buchmann

Design methodology management in design frameworks

Abstract

This paper is a tutorial on design methodology management (*dmm*) in design frameworks for VLSI systems and other complex electronic systems. The motivation for such functionality in design frameworks is discussed. Several models and mechanisms for dmm are reviewed and compared, according to a proposed taxonomy for classifying the different approaches. The taxonomy is based on the two main aspects of methodology management: the control of the task flow and the control of the object representations created during the design process. Also considered are the framework tool integration capabilities.

Keywords
Design frameworks. Design methodology management. Task flow management. Tool integration.

Resumo

Este trabalho é um tutorial sobre gerência de metodologias de projeto em ambientes de projeto de circuitos VLSI e outros sistemas eletrônicos complexos. A motivação para esta funcionalidade em ambientes de projeto é discutida. Vários modelos e mecanismos de gerência de metodologias de projeto são revistos e comparados, seguindo uma taxonomia proposta para a classificação das diferentes abordagens. Esta taxonomia é baseada nos dois aspectos principais da gerência de metodologias: o controle do fluxo de tarefas e o controle das representações de objetos criadas ao longo do processo de projeto. Também são considerados os recursos de integração de ferramentas dos ambientes.

Palavras-chave
Ambientes de projeto. Gerência de metodologias de projeto. Gerência de fluxo de tarefas. Integração de ferramentas.

1 Introduction

The main objective of frameworks for the design of VLSI circuits and other complex electronic systems is to provide means for building specific environments that are oriented towards different architectures, technologies, and design methodologies. A framework must allow the integration of tools from different sources, aiming at design data consistency and user interface uniformity. A framework must also provide other services, such as data management (data sharing, access control, version control, and long transaction mechanism), intertool communication, and design methodology management. Examples of frameworks that partially or totally support these goals are Oct [11], from Berkeley, Cadweld [3], from Carnegie-Mellon, and CWS [8], from the Cadlab in Germany, as well as some recent comercial products (e.g. the Open Framework from Cadence and the ValidFrame from Valid).

The main feature of a design framework is the provision of a uniform data model for design data representation [19], which supports the representation of digital systems as complex objects, taking into account aspects like composition of sub-objects, hierarchy, and instantiation of objects.

A design framework must allow for multiple representations for a design object. In the scope of this tutorial, we will designate the organization of these multiple representations as the object *control structure* [20]. Different representations for a design object can correspond to

- *design alternatives* (e.g. a standard-cell or a gate-array approach)
- *design views*, i.e. representations of the same object at different abstraction levels (algorithmic, RT, logic, layout, etc)
- *design revisions*, i.e. consecutive refinements or improvements of the same object.

Most systems offer a single sequential representation for the version evolution for handling alternatives, views, and revisions. Some systems, however, permit the distinction between the different situations, either in some restricted way [11,15] or in a general, flexible way [20], where the control structure is an integral part of the data representation model supported by the framework.

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, both meeting design constraints and achieving design goals. 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 can be followed from a given design state, as well as criteria for deciding between the possible design paths (again, these crieria 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 design process, so that it conforms to the established rules. It can be achieved by controlling the task flow and/or the design object representations created during the design process.

The rest of this tutorial is organized as follows. Section 2 discusses design methodology management and its main functions. This section provides the basis for classifying design methodology management approaches into three different classes. These classes are then analyzed in sections 3 thru 5. Examples of design methodology management approaches for these classes are also reviewed and discussed. Section 6 discusses the tool encapsulation issue, which is strongly related to the design methodology management. Final remarks are given in Section 7.

2 Fundamentals of design methodology management

Design methodology management is the control of the design process, so that the desired object representations are created, the design constraints are met, and the design goals are achieved. It can be realized by controlling either the task flow or the various design object representations created during the design process. In the former case, the framework capabilities are related to design guidance, while in the latter case design methdology management is achieved mainly by automatically maintaining methodology-related data consistency. In the following, we describe design methodology management features according to each of these approaches.

2.1 Task flow

A design methodology contains a set of tasks that must be executed in order that the desired objects are obtained. Each design methodology may need a particular set of tasks, as the following examples illustrate.

Example 1 In the case of the layout design of a data path in a standard-cell approach, the following tasks must be executed: mapping from an initial structural description into the cells of a library, partitioning of cells into bands, positioning of cells inside the bands, routing between bands, design rule checking, electrical parameter extraction followed by a timing evaluation, and netlist extraction followed by a netlist comparison with the structural description.

Example 2 In the layout design of a control part in a random logic approach, tasks include: multi-level logical minimization of an initial set of boolean equations, technology related optimization and mapping, manual layout generation, design rule

checking, electrical parameter extraction followed by a timing evaluation, and netlist extraction followed by a netlist comparison with the structural description obtained after the technology mapping.

Other layout design approaches, such as a data path design using a gate-array methodology or a control part design using a PLA strategy, would need other sets of tasks.

Many design environments allow the user to define the task sequencing either in an explicit or in an implicit way. The explicit task flow definition [2,5,6] follows an algorithmic (or equivalent model, such as a graph or a Petri net) description of the task sequencing, eventually offering the possibility of conditional branches, iterations, and parallel executions. In the implicit task flow definition, tasks are executed when certain input objects are available [17], when certain conditions hold [21,3], or according to rules that select the next task [1].

Fiduk et al [6] give a very extensive list of functionalities that can be performed by a tool that controls the task sequencing. These functions, that will not be detailed here, are totally or partially implemented by the systems covered by this tutorial.

Design guidance A mechanism for the management of the task flow is much more interesting when it offers also design guidance. This means that the system helps the designer to decide which is the best (or most promising) task flow to be followed. In order to give valuable help, the system must have knowledge about the design constraints to be met, the design goals to be achieved, the tool capabilities as related to the tasks to be executed, and about the design data itself. Design guidance is related to two main capabilities: automatic task and tool selection and automatic task backtracking.

For the automatic task selection, three capabilities can be offered:

- The system identifies alternative tasks that can be executed from the current design point, based on the knowledge about the conditions that must hold in order that each task is executed. Conditions may involve only the existence of certain design object representations, or more complex object properties.
- The system selects the most promising alternative task. The choice can be based either in knowledge about task capabilites or in result estimations.
- The system selects the best suited tool for the next task, if several tools, with different properties, can be chosen for the same task.

The following examples illustrate the functionalities related to automatic task selection.

Example 3 After a cell manual layout design, the system identifies three tasks to be executed: design rule checking, netlist extraction, and electrical parameter extraction. The condition for executing any of these tasks is the existence of the cell

layout representation. If the system could also rely on data qualities for identifying the next tasks, the nelist extraction and the electrical parameter extraction could depend on a "good" layout (a layout which has successfully passed the design rule checking).

Example 4 When a finite state machine description for a control part design is created, the system identifies two possible following tasks: two-level logic minimization, which would lead to a PLA-based design, and multi-level logic minimization, which would be the entry point for a random logic design. The choice could be based on a knowledge about the current design state (which is the available area for the control part, which is the maximum allowed delay, and how many terms and literals do the input equations contain), as well as about task capabilities (which are the expected area and delay for the given input equations in the case of the PLA and random logic approaches). It must be noted that in each of the possible approaches a completely different task sequence will be followed.

Example 5 From a behavioral, algorithmic description, the system must generate an RT structure. It must schedule operations in time frames and allocate registers, functional units, and interconnections. Three different tools can be called for executing this high-level synthesis task, each one based on a different synthesis strategy: *list scheduling*, where a hardware allocation constraint is specified and the algorithm attempts to minimize the total execution time; *force-directed scheduling*, where a global time constraint is specified and the algorithm attempts to minimize the resources; and *left edge algorithm*, which minimizes the number of registers under time constraints. The choice may be based on the particular design constraints, such as maximum expected operation time or pre-defined number of registers, and on the relative performance of the tools. As opposed to the previous example, the same task sequence will follow after this task, no matter what tool has been chosen for the high-level synthesis.

The other main capability which is related to design guidance is the automatic backtracking to a previous design point, either to restore design consistency, when design changes occur, or to analyze other alternatives, when constraints cannot be met or goals are not achieved. It is needed that the system restores (at least from the user viewpoint) the state of the design data base in that previous point. Automatic backtracking is illustrated by the two following examples.

Example 6 In a layout design, suppose that the user has already completed the design rule checking, the parameter extraction, and the timing evaluation. The user is not satisfied with the obtained timing performance, and decides to manually change the layout. Since the former extracted parameter values are no longer valid, and there is no certainty whether the new layout follows the design rules, the system decides to re-execute the design rule checking and the parameter extraction, restoring the

consistency between the layout and the parameter values.

Example 7 In the high-level synthesis situation, the *list scheduling* strategy, which gives minimum execution time, has been chosen. A topology evaluator indicates however that the final area will be too large, mainly due to an excessive number of registers. The system decides to backtrack the design process, and selects the *left edge algorithm*, maintaining the obtained execution time as a constraint, but searching for a minimum number of registers.

Automatic backtracking for the purpose of restoring the design consistency may be unnecessary if the designer uses already existing version and configuration control mechanisms. In the case of Example 6, the layout change would be commited as a new layout version. Since the parameter values were related to the old layout version, the design consistency is maintained: there is no parameter values yet obtained for the new layout version. Configuration control can be used as explained in the next example.

Example 8 The designer creates a structural composition of a data path, by selecting given representations for the ALU, the register file, the shifter, and so on. Since the simulation of this composition gives bad timing estimates, the designer decides to change the ALU. The structural composition is no longer valid, neither the obtained timing estimates. The system would have to backtrack in order to repeat these two steps. Inconsistency can be easily avoided by using version and configuration control. The structural data path composition contains instances of ALU, register file, and shifter object types. In the first try, the designer selects a given version of each of these object types and builds a possible data path configuration. In the second try, the designer only changes the version selection for the ALU, thus building a new data path configuration. Timing estimates are now consistent with their respective configurations.

2.2 Control of object representations

A design methodology intends to create design object representations that show desired properties. Because of their complexity, the desired objects must be designed in a modular way. Furthermore, the design must follow an hierarchy of abstraction levels. It is thus necessary to create various representations for the desired objects as well as for several auxiliary sub-objects. The two following examples are the counterpart of examples 1 and 2, respectively, when the system controls the design object representations, and not the tasks needed for this creation.

Example 9 For the standard-cell layout design of a data path, the following object representations must be created: an initial structural representation in terms of cells of a given library; a layout representation with cells partitioned into bands; a layout

representation with cells positioned inside the bands; a final layout representation after the routing between bands; an extracted netlist; and a structural representation with the timing parameters extracted and back-annotated from the layout.

Example 10 For the random logic design of a control part, the following object representations are needed: an initial set of boolean equations; the boolean equations after multi-level minimization; a structural representation after technology-related optimization and mapping; the manually generated layout; an extracted netlist; the structural representation with the timing parameters extracted and back-annotated from the layout.

Data consistency While the task flow control can be enhanced with design guidance, the creation of the design object representations can be controlled so that the objects maintain the desired consistency. Most systems support automatic data consistency by offering a unified data representation model. This model supports modularity, hierarchy, and instantiation, as well as multiple representations for the same object, corresponding to design alternatives, views, and revisions.

Data models offers various possible control structures. The Oct manager [11] organizes *cells* as groupings of *views*. Each view has a *contents facet*, where the basic interface as well as the actual cell view description are stored, and many *interface facets*, that define aspects that are externally visible for particular tasks. The DAMASCUS system [15], in turn, organizes *design objects* as groupings of *representations* at different abstraction levels. Each representation contains a set of design *alternatives*, and for each alternative there is a graph of *revisions*. Different control structures offer distinct capabilities with regard to the automatic data consistency they provide. Data models with a fixed control structure offer the same kind of data consistency for all applications running on the framework. This is illustrated by the following example.

Example 11 In the AMPLO data model [7], for each *agency* (a design object), there are many *alternatives*, corresponding to various possible interface definitions for the object. Many *versions* can be defined for each alternative, corresponding to views at different abstraction levels, different architectures or implementations for the object, or revisions of a given view / architecture / implementation. All representations under a given alternative share the same interface definition. When a new representation is created for an object (for instance an RT structural description is synthesized from a behavioral one), the system automatically checks if the interface definition has been maintained.

Flexible data models, in turn, allow each application to define the best suited control structure for each design object, as in the case of the CWS [8], NELSIS [18], and GARDEN [20] frameworks. The system can therefore support a methodology-specific data consistency. In the GARDEN data model, representations for a *Design*

object can be organized as a hierarchy of *ViewGroups*. Each ViewGroup can define *UserFields* (user- or methodology-defined attributes) and interface aspects that are automatically shared by all representations below. *Views* can be appended at any level of the ViewGroup hierarchy. The following example shows that a methodology-specific data consistency can be achieved by defining a suitable GARDEN control structure.

Example 12 An 32-bit operational block is designed in a full-custom approach as an array of 32 1-bit wide horizontal slices. Each slice is composed as an abutment of cells like an ALU, a shifter, and a register file. Two data busses must horizontally traverse the slices, crossing the left and right boundaries of all cells at exactly the same height (relative to the bottom of the slice). For each data bus, this height, as well as the implementation layer and the layout track width, have their values defined as UserFields of a ViewGroup which gathers all representations of an object OPSlice (corresponding to the 1-bit wide slices) that are related to its layout design. Layout Views under this ViewGroup must necessarily follow these design constraints, since they inherit these UserField values.

2.3 Classifying the approaches

A comparison between different approaches to design methodology management can be best understood by making it clear that *this framework function should not only provide a mechanism for sequencing the design tasks, but rather 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, according to the achievement of the above goals they provide:

- dmm through task flow control
- dmm through task flow control enhanced with design expertise
- dmm by controlling the object representations created during the design process
- dmm by coupling the task flow management with the control of the object representations.

These approaches will be detailed and exemplified in the next sections. Since the third approach corresponds to using a unified data model without any additional mechanisms for design methodology management, it will not be considered in the analysis.

Table 1 summarizes the main design methodology management features of the systems reviewed in this tutorial.

	APPROACH	TASK FLOW CONTROL	OBJECT CONTROL
Chiueh & Katz	task flow control	design history: - user-directed backtracking - interactive plan building	none
Ulysses	enhanced task flow control	rule-based: - centralized task scheduling - hierarchical tasks	file-based
Cadweld	enhanced task flow control	rule-based: - distributed task scheduling - hierarchical tasks	file-based
ADAM DPE	enhanced task flow control	rule-based plan building: - automatic pre- and post-estimation - automatic backtracking	file-based
FACE	coupling task flow control with version control	data-driven: - graph with dependencies between tasks and object views	single versioning of object views
STAR	coupling enhanced task flow control with control of object representations	condition-driven	methodology-specific control structures

Table 1: Design methodology management approaches

3 Task flow management

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. 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. Examples are the CFI approach and the design history manager from Berkeley, as well as commercial products, such

as the OpenFramework from Cadence and the ValidFrame from Valid.

CAD Framework Initiative Design methodology management with emphasis on the task sequencing is best represented by the CFI approach. According to the CFI – CAD Framework Initiative, a consortium of companys and research centers whose goal is to standardize the main aspects of design frameworks – a design methodology [6] is a "sequenced set of operations employed in the execution of a given function", while design methodology management "is related to the execution and control of methodologies used in the design process".

In the CFI approach, design methodology management can be decomposed into three main functions:

- tool integration – how to describe atomic tools (data requirements, argument definition, characterization of the command set for the user, CPU and memory requirements, etc)
- task flow – how to describe sequences of tasks (task data dependencies, hierarchical tasks, flow control through conditional branches, selections and iterations, concurrent execution of tasks)
- execution environment – how to execute task sequences, relating them to the invocation of atomic tools (selection of the best tool for a given task, automatic invocation of tools, storing of and queries to the history of tool executions, *backtracking* and error recovery).

This approach emphasizes an *operational* view of the execution of design methodologies, as well as the tool encapsulation aspect, which will be discussed in Section 6.

Berkeley Chiueh and Katz [2] present a model of *design history* that extends previous work at the University of Berkeley, the Oct manager [11] and a version server [12].

Design process management is defined as the "support to the creation of alternative design evolutions, while maintaining multiple and simultaneous design contexts, each one of them with its design objects and history of operation sequences".

An *activity* is a grouping of *tasks* (tool invocations), ordered in time as a tree, so that each task can have many succeeding tasks in the activity. A node in the tree defines a *design point*, that univocally identifies a consistent state of the design data base. A *cursor* indicates the current design point. The tree is built as the user executes new tasks. At any moment, the user can roll the cursor back to a previous design point, thus creating a new branch in the tree from that node. In this new current design point, the user can access only the data defined at that data base state.

This system allows the user to create, modify, and reuse design methodologies in an interactive way during the design process.

The design methodology concept in this system is somehow related to design data management [19]. The "design points" offer a functionality that is obtained in other systems by "configurations" (in Oct [16]) or "TimeStamps" (in GARDEN [4]), that also allow recovering a consistent data base state from a previous time frame. However, no relationship with control structures associated to the design objects is established (in other words, design data management is realized without the explicit use of control structures), and the system does not give any direct help for achieving the desired design qualities.

4 Enhanced task flow management

In the second approach, which is followed by the Ulysses [1] and Cadweld [3] systems, as well as in the ADAM Design Planning Engine [13], the task flow control is enhanced with knowledge about the design constraints, goals, tools, and data. This knowledge may be used for the purposes of automatic tool selection and automatic backtracking, as already discussed in Section 2.

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.

Ulysses Ulysses [1], developed at the Carnegie-Mellon University, was the first environment to cope with design methodology management. The system is based on a *blackboard* model, a data base storing design data and parameters for tool scheduling. Each tool is a *knowledge source* (KS) for the blackboard, asynchronously activated each time that the data in the blackboard match certain rules, that are also stored in the same data base. Tools communicate with each other through files posted to and retrieved from the blackboard. The blackboard also contains rules that express design consistency checks to be permanently verified. The *scheduler* is a special KS, implemented as an expert system, which monitors the blackboard, verifies which rules match the data, and chooses the next tool to be activated, following *conflict resolution* parameters when several KSs are potentially executable. Tasks may be hierarchically described as compositions of other tasks / tools, by using commands such as do-while, if-then-else and for-each, as well as parallel commands. The RPOL (*Rating Policy Module*) is another KS that calculates ratings for *design points* along the design process always when the blackboard is updated. The RPOL compares these ratings periodically, switching the design process to the most promising point.

All KSs, including the scheduler and the RPOL, as well as the rules and the scheduling parameters, are described through a specialized high-level language, called *scripts*.

Cadweld The Cadweld environment [3] was also developed at the Carnegie-Mellon University, as an evolution of Ulysses, still using the blackboard concept. Cadweld,

however, does not have a centralized scheduler. For each tool a CTKO (*CAD Tool Knowledge Object*) is built, containing information about functionalities and capabilities of the tool, as well as sets of activation patterns to which the tool can respond. Each CTKO standly monitors the blackboard. When an activation pattern matches the data in the blackboard, the tool candidates for execution. Tasks can be hierarchically described. A task can post requests to the blackboard, to which other tasks or tools may volunteer. The candidate tasks / tools are inspected regarding their capabilities, and the most promising one is selected. After the execution, the task which requested the service may inspect the resulting objects to determine if the design constraints have been met. If not, the task may decide to backtrack the design. If the execution has succeeded, the resulting files are posted to the blackboard.

ADAM The DPE (*Design Planning Engine*) [13] is a design methodology manager developed for the ADAM environment [9], from the University of Southern California. DPE is an expert system that builds, evaluates, and dinamically executes *design plans.* From an initial state, and by using knowledge about the tools that can be applied to this state, DPE builds a possible plan. From evaluations that are calculated for each alternative path in the plan, obtained by fast evaluators, the DPE chooses a path to be followed. Later, more precise evaluators will either confirm the choice or let the DPE return to a previous point and choose a new path. From the new obtained design state, the DPE repeats the process of building a design plan.

Similarly to the scheduler of the Ulysses environment, it is the DPE itself that selects the next tool to be executed, by using information that is available to it. While in Ulysses this information is generated by a single specialized tool that is periodically executed, the DPE automatically activates evaluation tools that are specific for each state of the design plan in order to obtain the desired information. The DPE creates a design plan upon which backtracking can be executed, as in the Berkeley work. In that case, however, it is the user that decides the return to a previous point, while the DPE perform this function automatically, according to evaluations that are fired by the system itself.

5 Coupling task flow with object representation management

Instead of specifying which and when tools must be executed, the system can control the consistency of the objects to be created. This can be done through a unified data model, which handles composition relationships, configuration management, hierarchies of alternatives, common properties of representations, and other user- or methodology-defined integrity constraints. Tool integration becomes clearly more complex, since a mapping between the tool data and the data model objects is needed. Although the objects thus automatically hold the desired consistency, the system does not give user guidance to obtain these objects, unless the data model is coupled with an enhanced task flow management. With this coupling, the system achieves 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 of systems that couple a task flow management mechanism with data management features for the purpose of obtaining design methodology management capabilities are the FACE Core Environment and the STAR framework.

FACE In the FACE Core Environment [17], developed at GE, the functions of the design methodology management are "to control all object *views* that are created in the design environment, to propagate changes to the data so as to recognize if data are updated, and to plan the sequence of tool executions that are needed for building a given object view". A design methodology defines rules for the execution of tools and for the construction of views, so that an "object evolution is restricted by the methodology, that serves as a template" for the construction of the object.

A design methodology is represented by a directed acyclic graph, where the nodes are tool invocations and design object views. The graph shows which views are necessary for the execution of a tool and which are created by the tool. Parameters that are needed for a particular tool invocation can be appended to the corresponding node. The user can define methodologies through a graphical-interactive facility.

Although the FACE environment emphasizes the design methodology as responsible for the definition of the control structure that is associated to an object, the system in fact does not provide any design guidance nor automatic data consistency. It is based on very simple version and task flow control techniques.

STAR The STAR framework [21] offers a design methodology management mechanism based on three principles: the definition of a conceptual scheme for the application, the specification of the task flow, and the hierarchical definition of methodologies. It offers a methodology-oriented data consistency and a condition-driven task flow.

The STAR framework supports a unified, flexible data model based on the GARDEN data model. For each application, one can define a *conceptual scheme*, containing various control structures that are specialized for different design objects, *auxiliary objects*, such as stimuli files for simulation purposes and testability measures, and *correlations*, that are general purpose relationships between object representations. Control structure specialization includes defining the overall topology of ViewGroups and Views, defining UserFields, and attaching interface aspects to the various nodes of the control structure.

The tasks are specified through a condition-driven approach. Each task is a 5-uple {*task name, tool name, input conditions, output objects, task goals*}. The *input conditions* specify data qualities that must exist in order that the task is executed, such as the existence of an object representation or a complex expression involving

UserFields of any objects. The *output objects* are the representations, auxiliary objects, and correlations that are created by the task. The *task goals* specify conditions expected after the task execution. If they are not achieved, the task *fails*, though the output objects are still created. A design methodology succeeds if none of its tasks is marked as "failed".

Design methodologies can be derived in a hierarchical way. Derivation includes specialization of already existing control structures, definition of control structures for new objects, and specification of new tasks.

Since 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.

6 Tool encapsulation

Tool encapsulation is a generic denomination including issues that are related to the tool invocation and to the mapping between the tool data and the unified data model. Tool encapsulation has been generally presented as a feature of design methodology management [6]. 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. These kind of facilities should be therefore considered as an independent feature of design frameworks.

Data mapping Frameworks that are based on an underlying unified data model have a more complex tool integration process, since there must be a mapping between the data handled by the tools and the objects of the data model. This mapping can be of course avoided, if the tool is written (or modified) for directly operating upon the data model. Although this implementation would give better efficiency, it needs modifications to the tool code. Furthermore, it is not compatible with the invocation of the tool within different environments built upon the framework, as in the case of the hierarchically derived design methodologies in the STAR approach. Many users seem to prefer a "plug-and-play" approach to the tool integration, where the tool is "encapsulated" by a mapping function. This kind of encapsulation is generally obtained by an *extension language*, such as the TIDL - Tool Integration Description Language in the CWS framework [10]. This language also allows the definition of conceptual schemes, mapping from the tool user interface to unified user interface resources, data base set-up (initial loading of objects), and access management aspects (such as user and group rights). The *scripts* language of the Ulysses environment is also an extension language.

The external data mapping, executed only *before* and *after* the tool, has its shortcomings, however. Besides the lower efficiency, this approach is not possible when there must be an interaction between the tool and the data base during the tool execution, as in the case of an interactive layout editor, which allows the interactive

creation and immediate re-use of cells.

Kraft [14] presents an alternative approach to the tool integration, where the data mapping is described externally to the tool, but the mapping function is embedded in the underlying operating system, so that each access the tool executes to its data is automatically converted by the operating system to an access to the unified data base.

Tool invocation Besides data mapping, tool encapsulation also includes abstracting tool invocation details from the user. The extension language allows the encapsulation of aspects such as the path to the tool, the arguments to be passed to it, and the exact syntax of the command line, so that the user can call a "high level" abstract task.

7 Final remarks

This tutorial has shown that design methodology management should not be restricted to a "syntactical" task flow control, where the system only offers to the user the possibility of defining and executing task sequences, as proposed in the CFI approach, for instance. This sequencing may be enriched with branchings, iterations, and concurrent executions, and the system may allow backtracking to previous design points. However, this approach does not offer a "semantic" control of the task flow, where the system helps the user to choose the most promising task sequences.

A first improvement to design methodology management consists therefore in enhancing the task flow control with knowledge about design constraints, goals, tools, and data, so that the system offers design guidance, as in the Ulysses, Cadweld, and ADAM environments. The system may contain resources for automatic plan building, task and tool selection, and backtracking, all of them based on the knowledge about the state of the design process. This solution still has its shortcomings, since tasks operate on isolated files, so that the system does not guarantee that the generated objects maintain a desired consistency, except if this consistency could be integrated into the system knowledge, an approach that would be either inneficient or incomplete.

A further improvement must be therefore directed into coupling the enhanced task flow management with a unified data model, such as the model offered by the Oct manager. A unified data model automatically holds various integrity constraints that relate object representations to each other, such as composition and equivalence relationships. An even better approach can benefit from a data model that allows the definition of methodology-specific conceptual schemes, as in the STAR framework.

References

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

[2] 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.

[3] 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.

[4] E.B. de la Quintana, G.O. Annarumma, and P. Molinari Neto. *GARDEN – the Design Data Interface.* Technical Report CCR-107, IBM Rio Scientific Center, Rio de Janeiro, 1990.

[5] A. Di Janni. A monitor for complex CAD systems. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[6] 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.

[7] L.G. Golendziner and F.R. Wagner. Modeling digital systems as complex objects. In *9th International Symposium on Computer Hardware Description Languages and their Applications*, IFIP, 1989.

[8] 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.

[9] J. Granacki, D. Knapp, and A. Parker. The ADAM Advanced Design Automation System: overview, planner, and natural language interface. In *22nd Design Automation Conference*, ACM/IEEE, 1985.

[10] K. Groening et al. From tool encapsulation to tool integration. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

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

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

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

[14] N. Kraft. Embedded tool encapsulations. In F.J. Rammig and R. Waxman, editors, *2nd IFIP International Workshop on Electronic Design Automation Frameworks*, North-Holland, 1991.

[15] J.A. Mulle, K.R. Dittrich, and A.M. Kotz. Design management support by advanced database facilities. In F.J. Rammig, editor, *IFIP Workshop on Tool Integration and Design Environments*, North-Holland, 1988.

[16] M. Silva, D. Gedye, R.H. Katz, and A.R. Newton. Protection and versioning for Oct. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[17] 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.

[18] P. van der Wolf et al. Data management for VLSI design: conceptual modeling, tool integration, and user interface. In F.J. Rammig, editor, *IFIP Workshop on Tool Integration and Design Environments*, North-Holland, 1988.

[19] F.R. Wagner. *Modelos de Representação e Gerência de Dados em Ambientes de Projeto de Sistemas Digitais.* Technical Report CCR-121, IBM Rio Scientific Center, Rio de Janeiro, 1991.

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

[21] F.R. Wagner, A.H. Viegas de Lima, L.G. Golendziner, and C. Iochpe. STAR: um ambiente para a integração de ferramentas de projeto de sistemas digitais. In *VI Congresso da Sociedade Brasileira de Microeletrônica*, SBMICRO, Belo Horizonte, 1991.

Relatórios de Pesquisa

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

RP-144: "Manual do Sistema TRAMO Projeto TRANCA versão 1.0", janeiro 1991.
M.A. SOTILLE; C.E.S. SOUZA; M.G.R. ARAUJO; M. LUBASZEWSKI; R.A.L. REIS

RP-143: "PILCHA: Projeto e Implementação de um Sistema Digital Discreto dedicado ao controle de acesso direto a memória", janeiro 1991.
F. AZEREDO; L. ROISENBERG; D.A.C. BARONE

RP-142: "Ambiente para Estudo de Fractais - Relatório de Projeto", janeiro 1991.
S.D. OLABARRIAGA; F.S. MONTENEGRO