A Tightly Coupled Approach to Design and Data Management

Flávio R. Wagner, Lia G. Golendziner, Miguel R. Fornari

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

Abstract

This paper describes the tight coupling between design and data modeling and management facilities in the STAR EDA framework. STAR implements an innovative and flexible data model that allows the user to define, for each object type, a schema of the design alternatives and views to be created during the design process. Alternatives and views can be hierarchically related through an inheritance mechanism in order to establish arbitrary data integrity constraints that must be followed when new object descriptions are generated. On-the-fly extensions and modifications to each object schema are supported by a schema evolution mechanism. The evolutionary schemata of alternatives and views and the integrity constraints that are established through the object schemata build the basis for a design methodology management mechanism.

1 Introduction

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

A problem found in most approaches to data and design methodology management is that they have a pure *syntactical* nature. This means that, although they help the user to organize meta-objects such as versions, configurations, and tasks, they have no knowledge about the *semantics* of the relationships between these meta-objects. As an example, many version management systems do not consider the fact that different versions must share common, arbitrary data representation properties in order to be semantically meaningful. A semantically rich data and design management, in turn, can support the automatic verification of design integrity constraints, thus guiding and/or enforcing the user towards meaningful results.

The semantics of the design process is related to the data contents of object versions and configurations that are the inputs and outputs to/from the tasks. Therefore, a more powerful support to data and design management can be offered if the version, configuration, and design methodology managers are tightly coupled to the data representation mechanism. This is a two-way coupling. Data and design managers must have knowledge about the design data in order to guide the user in the establishment of semantically meaningful relationships among meta-objects. As a result, they can conduct the design process (i.e. the derivation of new design objects) so as to guarantee the fulfillment of the design integrity constraints that are established as properties of the meta-objects or of the relationships among them. In the other direction, the data representation model must be defined so as to consider an adequate support to the various data and design management services to be integrated into the framework.

Although the literature describes semantic approaches to data and design management services, very few of them consider all services in an integrated way, and none of them is based on a tight coupling between the management services and the data representation model.

The STAR framework follows this approach. It implements an innovative and flexible data model that allows the user to define, for each object type, a schema of the design alternatives and views to be created during the design process. Alternatives and views can be hierarchically related through an attribute inheritance mechanism. This model allows the designer or project manager to establish arbitrary data integrity constraints that reflect a sequence of design decisions to be taken and that must be followed when new object descriptions are generated. On-the-fly extensions and modifications to each object schema are supported by a schema evolution mechanism. The evolutionary schemata of alternatives and views and the integrity constraints that are established through the object schemata build the basis for a design methodology management mechanism.

The remainder of this paper is organized as follows. Section 2 presents the main features of the STAR data model. The version management mechanisms are then discussed in Section 3. Schema evolution facilities are considered in Section 4. Section 5 discusses the STAR approach to design methodology management. Section 6 concludes with final remarks and describes future developments.

2 The STAR data model

The STAR data model is strongly based on the GARDEN data model [1], developed at the IBM Rio Scientific Center. Besides representing complex electronic design objects, it provides a flexible way for

managing the various representations created along the various dimensions of the design evolution (*alternatives*, *views*, and *revisions*). This feature allows the system to implement, according to user- or methodology-defined criteria, many different conceptual schemata for representing the design evolution.

In the STAR data model, shown in Figure 1, each *Design* object gathers an arbitrary number of *ViewGroups* and *Views*. The ViewGroups may in turn gather, according to user- or methodology-defined criteria, any number of other ViewGroups and Views, building a tree-like hierarchical *object schema*. The fact that Views may be defined at any level of the object schema offers an unlimited number of ways for organizing the different representations of the Design. Since the system does not enforce any grouping criterion, it is left to the user or to the design methodology to decide how Views will be organized.

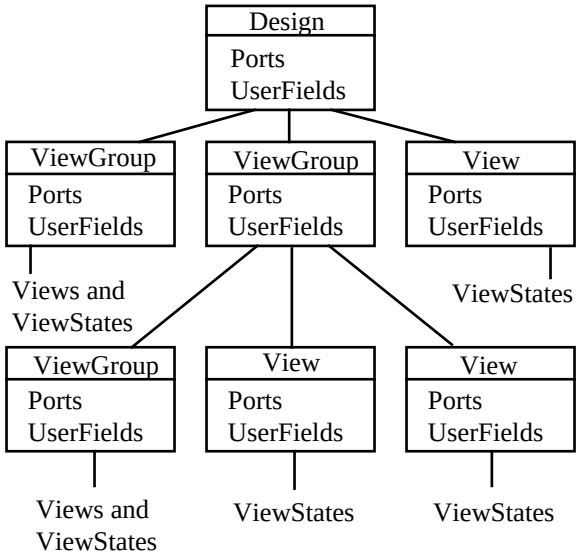

Figure 1: The STAR data model

Views are the leaves of the tree structure. STAR supports three kinds of Views: *HDL Views*, for behavioral representations, *Layout Views*, for physical descriptions, and *MHD Views* (Modular Hierarchical Description Views), for purely structural representations. Although all three View types may contain sub-objects that are instances of other object types, for HDL and Layout Views the data model does not handle the exact interconnections between the sub-objects, as it does for the MHD Views. As opposed to the MHD Views, HDL and Layout Views contain a file where specific design data, handled as a bit string by the STAR data management system, are stored. User-defined View types may be specified, so that MHD Views may be for instance of types "gate-level", "RT-level", and so on.

The object schema resembles a generalization hierarchy, in that each node is an abstraction of the subtree below it. Each node has some properties that may optionally be inherited by its descendant nodes. This is an *instance inheritance* relationship [2]: not only the existence of an attribute is transferred to the descendant nodes, but also its value, when defined. Inheritance may be *by default*, when descendant nodes may redefine attributes to more specialized domains and modify attribute values, or *strict*, when redefinition at descendant nodes is not possible.

There are three types of attributes for each node of the schema: *UserFields* are user-defined object attributes; *Ports* are interface signals and may contain in turn their own user-defined attributes; and *Parameters* allow the user to build generic, parameterized objects. The inheritance is mandatory with regard to the existence of Ports and Parameters.

Real design data, such as structural decompositions, HDL descriptions, and layout masks, are in fact contained in the *ViewStates*, that are the revisions created for each of the Views. The purpose of Design, ViewGroup, and View nodes of the object schema is to organize the various representations of a design object and to guarantee the consistency of the common attributes through the inheritance mechanism. Therefore, these nodes contain only the attributes to be shared by the representations they gather.

The data model supports other features such as the attachment of manufacturing process information (*Process* objects) to the design object representations, the specification of *Auxiliary Objects*, like test vectors and testability measures, and the establishment of general relationships among object representations (*Correlations*).

3 Version management

In the STAR context, *version* is a general designation for three dimensions of the evolution process: views, alternatives, and revisions. Version management is thus supported by two different mechanisms [3]: a) user- or methodology-controlled, *conceptual* versioning; and b) automatic revision control. Both mechanisms are strongly related to the data representation mechanisms. Revision control (for more details see [3]) is applied to all nodes of the object schema, i.e., to all versions of the conceptual level.

At the conceptual level, the user or the design methodology may define a particular object schema for each design object so as to organize views and alternatives according to a given strategy. This allows the user to apply a methodology control which is highly tuned for the design of each object. At this conceptual level, each design object has a small number of versions, that are created strictly under user- or methodology-control.

A conceptual schema for an object restricts / specializes the generic object schema presented in the previous section by: a) defining the overall topology of the schema, i.e., which are the ViewGroups and Views, how they are related to each other, which are the View types; b) specifying at which nodes Ports and Processes are to be attached; and c) defining attributes (both inherited and non-inherited) in the various nodes.

ViewGroups have a very general semantics. They can be used, for instance, to build a hierarchy of design decisions, where alternatives from a given design state

are appended to the previous alternative that has originated this state. This hierarchy of design decisions, if desired, can be close to a hierarchy of design abstraction levels. Furthermore, ViewGroups store the attributes that are common to all representations that correspond to a given design decision. The data model thus helps the designer guarantee that all representations related to a given design alternative have these common properties.

The application of these conceptual versioning mechanisms to the definition of an object schema which is specially oriented to the design of a microprocessor control block is illustrated in Figure 2. The schema defines the various representations that correspond to the control block behavior (a symbolic FSM before and after state assignment) and implementation, in both random logic and PLA design approaches. For each of these approaches, representations are organized under a distinct ViewGroup. For the random logic approach, representations include boolean equations after multi-level minimization, a gate-level structure after technology mapping, and a gate-level structure annotated with timing extracted from the layout. Layout related representations corresponding to the random logic approach are organized under an additional ViewGroup and include a layout view and an extracted logical netlist. This layout representations add two Ports (VDD and GND) to the control lines already defined at the root of the schema. Further representations for the PLA design are not shown. All random logic related representations share as a common attribute the maximum propagation delay, stored as a UserField in the ViewGroup VG-RandomLogic.

A configuration manager, which is highly integrated to the version manager, is also available. It supports *static*, *dynamic*, and *open* configurations and is implemented as an interactive tool that offers several facilities for building, storing, and re-using configurations. Version selection may be performed in a manual, automatic, or semi-automatic way. In the automatic selection, pre-defined criteria, such as current version or most recent version, are used. The semi-automatic selection is based on user-defined configuration expressions, involving object attributes.

In order to compare version models supported by other systems to the STAR approach, we may classify them into two main categories. The first category corresponds to general-purpose approaches, such as those proposed by Kemper et al [4] and Katz et al [5]. Kemper et al proposed a version model where an *abstract object* holds the properties that are common to all its versions. *Version hierarchies* of any depth can be built, using the instance inheritance mechanism. Furthermore, for each design object a different organization for the version set can be defined, using *version graphs* (ordering of versions by time relations or development history) and *partitions* (classification of versions according to the design status). This general model allows for the definition of any versioning schemata. It could be used as a platform for building application-oriented version models, such as the model in the STAR framework. The Version Server of Katz et al also supports a very general versioning

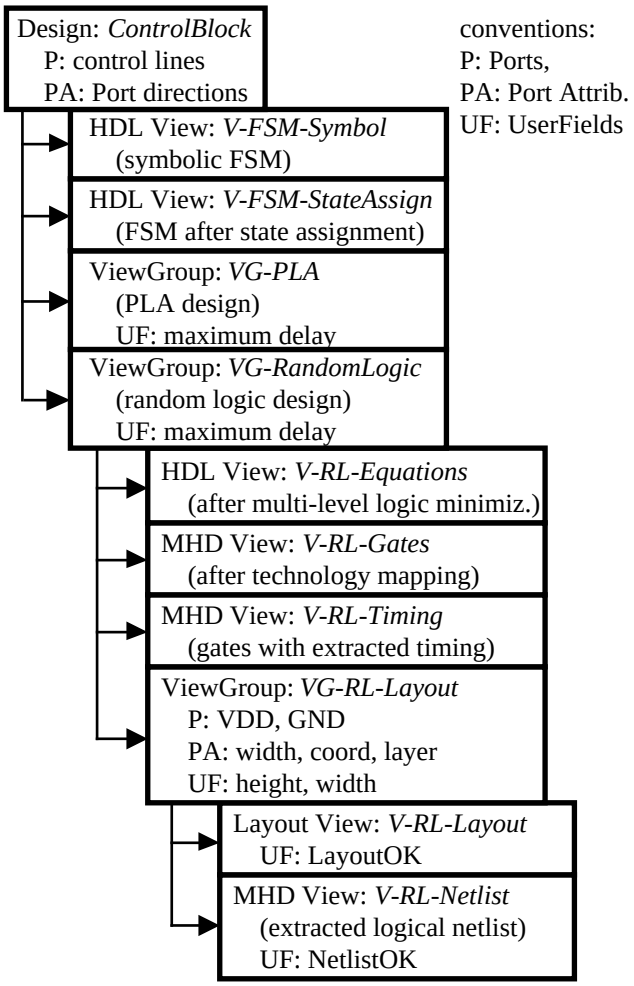

Figure 2: An example of an object schema

schema. While offering an interesting model for resolving dynamic configurations and a powerful change propagation mechanism, a distinction between alternatives and views cannot be established. The model does not support separate interface and contents portions of design objects and makes an unclear distinction between object definitions and object instances.

The second category corresponds to application-specific version models and includes for instance the DAMASCUS [6] and Oct [7] systems. The Oct manager organizes representations of a *cell* in *views*. A view has any number of *facets*. A special facet, named *contents*, contains the actual design data for the view and its basic interface definition. Other *interface facets* define externally visible information for specialized tools. Each facet can have many versions. Cells are grouped into *workspaces*, and a configuration mechanism allows the automatic selection of versions for facets in a workspace. The Oct manager does not support a distinction between design alternatives and views. The DAMASCUS system defines a design object through a 4-stage hierarchy consisting of *represen-*

tations (similar to views), *alternatives*, *revisions*, and *design stages*. Each alternative may define a different object interface, which is then inherited by all its revisions. Revisions may add new interface attributes. While considering all dimensions of the version space, this versioning schema is fixed.

These version management approaches are thus either too general, offering a weak semantic support for electronic design automation, or too specific, implementing a particular, fixed versioning schema. Some of them do not consider all dimensions of the versioning space [7], or are in fact restricted to revision control [5]. The STAR data model, on the other hand, handles revisions, alternatives, and views in an integrated approach. Instance inheritance exists from the Design node down to the ViewStates. It combines adequate semantic expressiveness (ViewGroups, Views, interface and attribute inheritance) with the support for the implementation of schemata according to user-or methodology-defined strategies. Its flexibility and completeness allows the modeling of a variety of versioning schemata [1], as well as the definition of schemata that are not supported by other systems.

4 Schema evolution

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

A mechanism for the definition and evolution of object schemata was thus developed for the STAR framework. As in other systems [8, 9], this mechanism is based on *schema invariants*, that are basic conditions that must be always fulfilled in order to assure that the schema is in a consistent state. Evolution operations include: creation and removal of Design, ViewGroup, and View nodes; creation, removal, and modification of attribute domain and value; migration of an attribute to a descendant; and modification of attribute characteristics (versionable / non versionable, inheritable / non-inheritable, strict / default inheritance).

The set of invariants assures that the object schema is in a consistent state. However, sometimes a great number of operations is necessary to modify the object schema from its actual state to another consistent state. In order to perform this modification, the user has to open a *schema evolution transaction*, suspending checking of invariants until the transaction is committed, except when an operation for stabilizing or consolidating a version is contained within the transaction. This is important to assure that further selections of versions will result in a set of consistent versions. These transactions will be incorporated into a more general long transaction mechanism to be developed in the future, since their duration is much longer than conventional transactions.

When a new node is created, its name and its immediate ascendant must be informed. The name of the node must be unique, according to the schema invariants. When a node is removed, all its descendant nodes are removed too. For working versions, the design data are really removed. For stable versions, the data are maintained in the database, but just historical queries can be done on them. Consolidated versions cannot be removed.

Attributes can be inserted, removed and copied at any time. If the current version is not a working one, a new version is derived from it, and the attribute modification is effective in this new version. If an inheritable attribute is redefined, some rules must be verified. If this attribute redefines another inherited attribute, then the inheritance mode must be by default, and the redefined domain must be a subset of or equal to the inherited domain. Furthermore, if there is a redefinition of this attribute in descendant nodes, this redefinition must also follow the two previous rules. When the domain of a UserField is changed, its value can be changed in the descendant nodes to keep them consistent. It is also possible to define a special function which automatically maps the old values to the new domain.

5 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: a) tasks that must be executed when the design process reaches a given state (this state can be expressed for instance in terms of design object properties); b) alternative design approaches that must be followed from a given design state, as well as criteria for deciding between the possible design paths; and c) design representations that must be created under given conditions, e.g., a representation at a more detailed abstraction level or alternatives that must be evaluated according to design requirements. *Design methodology management* is the control of the creation of design object representations and of the execution of required tasks so that objects and tasks conform to the established rules.

The definition of a design methodology in the STAR framework is based on three main principles: 1) the definition of object schemata for design objects; 2) the specification of the task flow; and 3) the hierarchization of design strategies.

A design methodology is primarily based on object schemata that organize all representations that can be created for the design objects of a given application. Each design object can have its own object schema, depending on the design strategy to be applied to it. Section 3 already discussed the many possible specializations of an object schema.

Task flow is expressed through a condition-driven model. A task is eligible for execution when its *input conditions* hold true. These conditions can express the existence of objects or qualities of them, explicitly modeled as attributes in the object schemata. *Task goals* describe properties expected from objects after a task is executed. If they are not achieved, the task "fails", though new object representations may have

been created. It is left to the user to select among many enabled tasks. A methodology succeeds when all its tasks have succeeded. Tasks may be executed stand-alone or within a *design strategy* (a collection of tasks).

Design methodologies can be organized in a hierarchical way. A new design methodology can be derived from a previous one: a) by specializing (either by extending or restricting) the object schemata of the previous methodology; b) by defining new Design objects (and their object schemata), Auxiliary Objects, and Correlations; and c) by defining new tasks or strategies. A design task is defined at a given level of this methodology hierarchy. The task definition must be consistent with the object schemata that are known to this methodology. Users are also constrained by a methodology to the execution of a given set of tasks and to the manipulation of a given set of design object representations. STAR does not make a distinction between "designers" and "project managers". Each user is potentially also a project manager and has the possibility of deriving new methodologies by extending/restricting the object schemata to which he/she has access.

A comparison to other approaches can be better understood by making it clear that a design methodology manager must not only provide a mechanism for sequencing the design tasks. Instead, it must also guide or enforce the design process in order to meet the design constraints and to achieve the desired goals, while maintaining the overall data consistency. From this broader perspective, four basic approaches to design methodology management can be identified in the literature.

In the first approach, there is only a mechanism for sequencing the tasks. The system may have capabilities for the definition and automatic execution of task sequences, including conditional branchings and iterations, and for storing and repeating user-defined ad-hoc sequences. The CFI approach [10] and the history model of Chiueh and Katz [11] follow this reasoning.

In the second approach, which is followed by the Ulysses [12] and Cadweld [13] systems, as well as in the ADAM Design Planning Engine [14], the task flow control is enhanced with knowledge about design constraints, goals, tools, and data. This knowledge provides two basic capabilities: a) automatic tool selection, by identifying alternative tasks that can be executed from the current design point and selecting the most promising one (e.g. based on tool result estimations, as in the ADAM DPE); and b) automatic backtracking to previous design points, either to restore 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 easier tool integration, the support for automatic data consistency depends entirely on the task flow control, and the quality of the design depends on the completeness of the knowledge representation.

In the third approach, instead of specifying which and when tools must be executed, the system only controls the consistency of the objects to be created. Although objects thus automatically hold the desired consistency, the system does not give user guidance to obtain them. Although data consistency management is normally considered as a data management feature, it is clear that it provides support for design methodology management too. There are two alternatives for controlling the consistency of the objects. The first one is to ensure consistency *by construction* through a unified data model, which can handle user- or methodology-defined integrity constraints. Tools in this case become obviously more complex, since they must implement the verification of the integrity constraints expressed through the data model. As an alternative, consistency can also be obtained through a *post-verification* approach, as in the Valkyrie validation system [15], which verifies if objects to be checked-in into the data base are consistent with previously existing ones (consistency is in fact restricted in this system to *equivalences* among objects). The checking is performed by verifying if the appropriate design steps have been applied to the objects.

Finally, a task flow control enhanced with design knowledge can be combined with mechanisms for controlling the creation of object representations that must fulfill integrity constraints. In this case, both automatic data consistency and methodology-oriented user guidance are obtained. Design qualities are achieved partially by the data model and partially by the task flow control, in such a way that each of these mechanisms responds for the consistency management that is most natural to it. In particular, the modeling of the design knowledge, 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 schemata. The STAR design methodology management model is the only one that implements such a combination of features.

6 Final remarks

The STAR framework establishes a strong coupling relating data representation mechanisms, versioning capabilities, and design methodology control. This paper highlighted these relationships, and the most important ones are summarized in the following.

The data model has highly interleaved data representation and management (conceptual versioning) aspects, supporting the inheritance of design data (UserFields, Ports, and Parameters) among design alternatives and views and the definition of application-specific schemata for each design object.

The object schemata are an integral part of a design methodology definition, since they specify all object representations to be created during the design process, as well as the attributes and Correlations that are needed for controlling the task flow. The creation of new object representations is restricted by the constraints imposed by the inheritance of design attributes in object representations. Schema evolution is an essential feature for supporting design methodology management in an evolving environment, where new design strategies may impose the incorporation of new features (representations, attributes, and Correlations) to the object schemata.

As a very general feature, it may be concluded

that data and design management aspects should not be supported as completely distinct framework services, as in most other approaches. The consistent and tightly integrated design of mechanisms for data representation, version management in all dimensions of the design evolution (alternatives, views, and revisions), configuration management, and design methodology management should be regarded as an essential requirement in the design and implementation of a framework. Furthermore, all these services must take into account that every design environment is a living entity, so that they must support evolution together.

STAR follows an innovative approach to version management which simultaneously a) gives dedicated support to all dimensions of the design evolution process, b) allows the definition of versioning schemata that are particular for each design object, according to user- or methodology-defined strategies, and c) is strongly oriented towards EDA.

The STAR data model (including conceptual versioning and schema evolution features) is mapped to the KRISYS knowledge base management system [16], developed at the University of Kaiserslautern, Germany. KRISYS supports the abstraction concepts that are needed for modeling complex, versioned objects. The basic data management layers (data representation, version and configuration management, schema evolution) are now available. The mapping from the STAR data model to KRISYS, implemented through a set of LISP functions, is described in other paper [3]. Other layers of the STAR framework (managers and application tools) are developed in the C programming language and communicate with the LISP functions. Tools access the services of the various framework managers through application programming interfaces. The STAR *cockpit* gives the user access to all framework managers. Application tools may be invoked either in the context of user-defined design methodologies or stand-alone. The user may browse through the data representation and management information that is expressed in the data model, by means of complex, interactive textual and graphical queries. Object-sharing mechanisms are supported by a cooperation manager [17].

As a next step, the task flow control mechanism will be implemented. As another further development, a SQL-like language will be defined and implemented. This language will be incorporated in a host language, like C, to allow object schema definition and evolution at run-time. Finally, a graphical interface, for defining and manipulating object schemata, will be added to the STAR cockpit.

Acknowledgments

The authors acknowledge the valuable support of other people who are or were also involved in the design and implementation of the STAR framework, in particular C.Iochpe, H.Grazziotin Ribeiro, R.S.Mello, M.A.C.Livi, A.Lemos, and J.Lacombe.

References

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

[2] W.Wilkes. Instance inheritance mechanisms for object-oriented databases. In K.R.Dittrich, ed., *Advances in Object Oriented Database Systems*. Springer, 1988.

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

[4] F.Kemper, W.Wilkes, and G.Schlageter. Basic mechanisms to support versioning in the database component of a CAD framework. In *2nd IFIP Intern. Workshop on EDA Frameworks*. North-Holland, 1991.

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

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

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

[8] J.Banerjee et al. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems*, January 1987.

[9] D.J.Penney and J.Stein. Class modification in the GemStone object-oriented DBMS. *SIGPLAN Notices*, December 1987. (Proceedings of the OOPSLA-87)

[10] K.Fiduk et al. Design methodology management - a CAD Framework Initiative perspective. In *27th Design Automation Conference*. ACM/IEEE, 1990.

[11] T.Chiueh and R.H.Katz. A history model for managing the VLSI design process. In *International Conference on Computer-Aided Design*. IEEE, 1990.

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

[13] 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.

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

[15] R.Bhateja and R.H.Katz. VALKYRIE: A validation subsystem of a version server for computer-aided design data. In *24th Design Automation Conference*. ACM/IEEE, 1987.

[16] N.Mattos. *An Approach to Knowledge Base Management*. Springer-Verlag, 1991. (Lecture Notes in Artifical Intelligence)

[17] M.A.C.Livi and C.Iochpe. Design and cooperation management through design steps and object characteristics. In *XVIII Conferencia Latinoamericana de Informatica*. CLEI, 1992.