A Formal Basis for Design Process Planning and Management*

Margarida F. Jacome
Electrical and Computer Engineering Dept.
University of Texas at Austin
Austin, TX 78712

Stephen W. Director
Electrical and Computer Engineering Dept.
Carnegie Mellon University
Pittsburgh, 15213

ABSTRACT

In this paper we present a design formalism that allows for a complete and general characterization of design disciplines and for a unified representation of arbitrarily complex design processes. This formalism has been used as the basis for the development of several prototype CAD meta-tools that offer effective design process planning and management services.

1 Introduction

CAD tools should not simply aid designers in solving specific synthesis, analysis, and/or optimization design (sub)problems but, in addition, they should aid designers in planning and managing the increasingly complex design process. In order to achieve this goal we need a means of representing designs more abstractly than is required for detailed design. More specifically, we need a design formalism that explicitly represents the fundamental intentions, strategies, and mechanisms in design, i.e., the *content* or the *semantics* of design. (As opposed to having such an information "buried" or hidden in the syntactical idiosyncrasies of the design *"form".)* Observe that through the realization of such a syntactically transparent and semantically rich design formalism, it becomes possible to articulate the essential concepts of design and model both, the *design of artifacts* and the *process of design*, across the design disciplines, in a coherent and precise way. A design formalism with the above characteristics is immediately useful in guiding the development of general purpose, highly effective design process planning and managements *meta-tools*. Observe that such meta-tools, by being capable of capturing the fundamental strategies for controlling complexity embed in traditional design methodologies, and by intelligently making use of such strategies throughout the design process, allow for the creation of a new generation of powerful CAD environments.

In this paper we present such a formalism of design. Our formalism allows for a complete, and general characterization of design disciplines and for a unified representation of design processes taking place in the context of these disciplines. [1] This formalism has been used as the basis for the development of several prototype CAD meta-tools, such as Minerva [2] and Clio [3], that offer highly effective design process planning, management, and decision support services.

The remainder of this paper is organized as follows. First we discuss the main issues involved in properly planning and managing complex design processes. Then, in Sections 3 to 7, we introduce our formal characterization of design processes. Some conclusions are given in Section 8.

2 The Design Process Planning and Management Problem

The design process is fundamentally a search process that takes place in a "solution space" for an object that meets a desired "initial specification." Since the solution space associated with most design problems is of high dimension, and since the effort for generating and evaluating a possible solution to a design problem can be extremely large, the search process must be efficient. To achieve such efficiency, problem decomposition is often employed to derive sub-problems, each of which is less complex than the original problem. This decrease in complexity translates into a problem that has a lower dimensional solution space, and is presumably easier to search. However, these subproblems cannot always be solved independently of each other [4], since design decisions made during the solution of one sub-problem may impact decisions that need to be made while solving other subproblems. As a consequence, there is information that must be shared among sub-problems and consistency checks must be made during their solution. Choosing which methods and strategies to use for problem decomposition during active problem solving, and then properly applying them for the solution of a design problem, while guaranteeing that consistency is preserved, constitutes the essence of what we call *design process planning and management.*

* This work is supported in part by the Engineering Design Research Center, Carnegie Mellon University, under contract no. EEC-8943164.

©1994 ACM 0-89791-690-5/94/0011/0516 $3.50

In the process of solving each of the sub-problems that result from problem decomposition, heuristics are typically employed to further prune the associated search space. Since such heuristics do not always work, designers sometimes reach an impasse. In such a situation designers may need to *backtrack* to (i.e., to revisit) previous design states in order to reconsider those design decisions that proved to be inadequate. Another possibility is that the designer may conclude that a given set of requirements for the object under design is infeasible and may have to backtrack to a previous design state in order to consider alternative trade-offs for such requirements.

3 The Design Space

Development of a design process planning and management meta-tool that effectively addresses the needs outlined above requires a syntactically transparent, problem-based, representation of the design process. Such a representation would unify, at an adequate conceptual level, all of the different levels and stages of a design process. It would also allow a uniform and direct representation of problem decomposition and dependencies among the resulting sub-problem, as well as implementation of both local and global, backtracking mechanisms. (Global backtracking mechanisms are those that impact different phases and/or stages of the same design.) In order to provide such a unified, problem based design process representation with the necessary *semantic content*, we also need to be able to characterize the **design disciplines** in which the design processes take place. The formalism which we now describe achieves these goals.

We begin the discussion of the formal representation of the design process by defining the ***design space***, which is the *problem space* where design takes place. The design space contains a **knowledge** component and a **data** component. The knowledge component explicitly characterizes the design discipline and the data component characterizes the design process.

Space precludes us from presenting a detailed discussion about the two fundamental components of the design space. (A complete definition of the design space can be found in [thesis].) However, we will introduce and briefly discuss some of the key concepts and fundamental abstractions in these two components. As will be shown below, design disciplines, and thus the knowledge component of the design space, are characterized in terms of *abstract classes of design objects, consistency constraints, design objectives,* and *operators*. Abstract classes of design objects may be organized in a *discipline hierarchy* and are characterized by *specifications*. As will also be shown, the design process, and thus the data component of the design space, can be fully characterized in terms of the *design state* and the *design history,* i.e., the sequence of design states that led to the current design state.

4 Characterizing Design Objects

A **design object** is an abstraction of a physical device or process and is characterized by a set of inputs, a set of outputs, and a set of **properties** that describe it.

An **abstract class of design objects** is a set of design objects characterized by the same set of **abstract properties**. Each abstract property specifies a particular feature that can be described in characterizing a particular design object of the set. The **abstract specification** for an abstract class of design objects is thus the set of all abstract properties which are (or may be) necessary for characterizing each design object in the particular abstract class. As an example, consider the abstract class of design objects known as *adders*. An abstract property for this class is "**<*adder*> _has_ _word-length_ _equal_to_ <*val*>**".

A **property instance**, or simply a **property**, is a statement derived from an abstract property. It describes the feature specified by the abstract property for the object under design. An example of a property derived from the previous abstract property, and used for characterizing the design object "MY_ADDER" is "**MY_ADDER _has_ _word-length_equal- to_ 16 bit**". Finally, a **specification** is as a set of properties characterizing a particular design object.

4.1 Design Object Properties

The abstract specification of an abstract class of design objects (and also the specification of a design object), can be partitioned into two general categories: *behavioral sub-specifications* and *structural sub-specifications*.[1] Behavioral specifications contain the properties that define how the design object should "behave" or function. Structural specifications, on the other hand, define how the design object should actually be 'realized'. Each of these specifications may be further partitioned into three categories: *requirements, restrictions, and descriptions*.

Behavioral and structural **requirements** are properties that define the "givens" of the design problem, in other words these are the properties that the object under design must meet when design is complete. Behavioral requirements are frequently the initial specification for a given system, either when the intended behavior for the system has "ideal characteristics"[2] or when the complete

1. For simplicity purposes, in what follows we will refer to these sub-specifications simply as specifications.

behavioral description of such a system is too complex to be directly managed, at least during the initial stages of the design process. Slew rate is an example of a behavioral requirement for operational amplifiers. Examples of structural requirements are area and dissipated power.

Behavioral and structural **restrictions** are properties that are employed by the designer to prune the design space in order to reduce design complexity. Designers typically derive restrictions during the **conceptual design** phase of the design process. Examples of structural restrictions are fabrication technology, layout style, and circuit topology.

Finally, behavioral and structural **descriptions** are properties that, respectively, *fully* define the behavior and the structure of a given design object. Specifically, **behavioral descriptions** indicate, via mathematical equations or behavioral description languages, how a design object reacts, or should react, to specific sets of stimuli applied to its inputs. Behavioral descriptions can be specified using the fundamental connectives and/or constructs of these mathematical formalisms and/or behavioral description languages or, alternatively, a behavioral description can be specified in terms of a set of *assumed behavioral sub-descriptions*. A behavioral sub-description is said to be "assumed" in a particular behavioral description, if such a sub-description is *not explicitly represented* in the behavioral description (in terms of the fundamental connectives and/or constructs of the mathematical formalisms and/or behavioral description languages that characterize the particular abstraction level of the design discipline -- see Section 4.3). Each of the assumed behavioral (sub)descriptions can, thus, be seen as a "non-primitive behavioral building block".

A **structural description**, on the other hand, is an interconnection of a number of structural building blocks. Such building blocks can be *primitive building blocks*, or *non-primitive building blocks*. A **primitive building block** cannot be expressed in terms of other, simpler, building blocks defined at the same abstraction level. Before the design process can commence, structural primitive building blocks must be available for each abstraction level of the specific design discipline (see Section 4.3). Furthermore, a behavioral description -- called a **model** -- must exist for each structural primitive building block. An atomic physical device or process (and thus the primitive building block that represents it) may have different models depending on the operational range in which it will be used and/or the required accuracy with which we want to reproduce the real physical process.

4.2 Design Object Decompositions

A description (behavioral or structural) expresses the (complex) behavior or structure of an entire design object in terms of less complex (primitive or non-primitive) behavioral or structural "building blocks", respectively. A behavioral description is said to constitute a **fundamental functional decomposition** if it contains only primitive behavioral building blocks, otherwise, is said to constitute a **non-fundamental functional decomposition.** Similarly, a structural description is said to constitute a **fundamental structural decomposition** if it only uses structural primitive building blocks. Otherwise, if the structural description uses at least one structural non-primitive building block, it is said to constitute a **non-fundamental structural decomposition**. For instance, at the circuit level of abstraction, the structural description of an OPAMP may be defined in terms of transistors and capacitors, which are among the primitive building blocks of the circuit level of abstraction for the VLSI digital design discipline. This structure would, thus, constitute a fundamental structural decomposition. On the other hand, the structure of the OPAMP could alternatively be represented, at the same abstraction level, using non-primitive building blocks, such as current sources, differential amplifier stages, and voltage gain amplifiers, which constitutes a non-fundamental structural decomposition.

4.3 The Discipline Hierarchy

Typically, for each given discipline, it is possible to identify a number of different abstract classes of design objects. As an example, for the discipline of VLSI Digital Circuits, we may design ALUs, Multipliers, and Adders. In this section we introduce the notion of a *discipline hierarchy* which organizes the entire set of abstract classes of design objects associated with a given discipline.

The discipline hierarchy, denoted by Δ in Figure 1, is defined as a two dimensional structure of ordered sets of **facets**, denoted by δ_i, i = 1, 2,..., each of which are sets of abstract properties from specific abstract classes of design objects. As illustrated in Figure 1, the vertical dimension of the discipline hierarchy is referred to as the *abstraction dimension*, and the horizontal dimension is referred to as the *specialization dimension.*

The abstraction dimension organizes the design hierarchy into **abstraction layers,** denoted by Δ_i^A, where "i" identifies the particular abstraction layer. More specifically, the abstraction dimension *partitions* the abstract

2. "Ideal" in the sense that it has characteristics that are known to be physically or technologically impossible to achieve or implement, but can be approximated by a real design object

specification of each individual abstract class of design object, creating an abstract sub-specification for each abstraction level Δ_i^A, i= 1, 2,.... The resulting sub-specifications constitute the facets of the abstract class of design objects. The design abstraction layer that corresponds to the least abstract level of Δ is called the **ground abstraction level**, or simply the **ground level**, and is denoted by Δ_G^A in Figure 1. (Note that the least abstract layer has the most detail associated with it).[5] In Figure 1, the direction in which the level of abstraction increases is indicated.

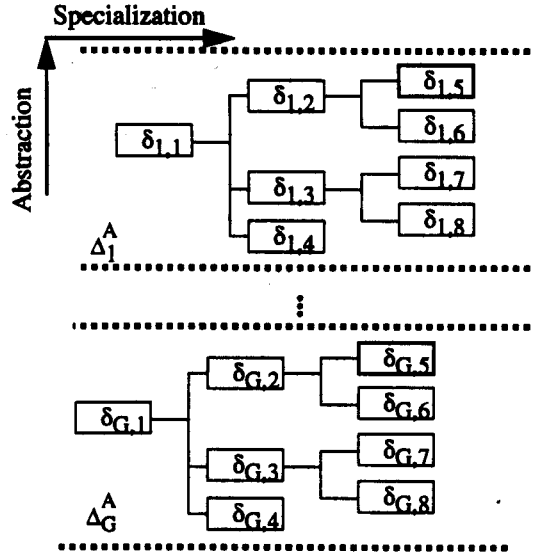

Figure 1 The Discipline Hierarchy Δ

The specialization dimension (or horizontal dimension) of the discipline hierarchy serves a dual purpose. First, it discriminates between the different abstract classes of design objects which may coexist for the design discipline, and second it allows for the expression of commonalties among different abstract classes of design objects defined for a particular design discipline. In other words, it provides for the representation of specialization (or conversely, generalization) by allowing different abstract classes of design objects to share abstract properties. An example of a possible design hierarchy for VLSI digital circuits can be found in [2].

5 Consistency Constraints

Values of properties that belong to the same, or different, abstraction levels, and to the same, or different, abstract classes of design objects, may be constrained by arbitrarily complex relations. Examples of relations among properties that may be defined, for instance, for a CMOS inverter, at the transistor level of abstraction would be (assuming Vss = 0):

$V_{IL} = (3\ V_{DD} + 3\ V_{TP} + 5\ V_{TN})/8$; and

$V_{IH} = (5\ V_{DD} + 5V_{TP} + 3\ V_{TN})/8$;

where V_{IH} and V_{IL} denote the high and low logic thresholds, respectively; V_{DD} denotes the drain voltage source; and V_{TP} and V_{TN} denote the threshold voltages for the inverter's p-channel and n-channel MOSFETs, respectively.

Abstract consistency constraints represent such dependencies among property values. An **abstract consistency constraint** is thus defined by an *independent abstract sub-specification*, a *dependent abstract sub-specification*, and a *relation* involving the abstract properties contained in both sub-specifications. For instance, for the consistency constraints shown above, the set of independent abstract properties could be {Vdd, V_{TP}, V_{TN}}, while the set of dependent properties could be {V_{IL}}, for the first case, and {V_{IH}}, for the second case. Observe that, given a particular relation, defining which abstract properties belong to which group may be design methodology dependent. Note also that an abstract property can be a member of an arbitrary number of consistency constraints and may be listed as a member of the independent abstract sub-specification for some of these abstract consistency constraints, and as a member of the dependent abstract sub-specification for the remaining abstract consistency constraints.

An **active consistency constraint**, is defined by a *relation* and by an *independent sub-specification* and a *dependent sub-specification*.[1] For each active consistency constraint, a predicate can be derived, denoted by "**VERIFY**(constraint)" whose value is "true" if the independent and dependent sub-specifications verify the relation defined in the consistency constraint and "false" otherwise.

6 Organization of Design Objects in a Design Process

A hierarchy[2] of design objects is created during a design process. We define the **design process hierarchy** as a two dimensional organization of such design objects. The vertical dimension of this hierarchy is the same as the hierarchy of abstraction levels defined in the discipline hierarchy and is also called the **abstraction dimension**.

Functional and structural decompositions, together, define the horizontal dimension of the design process hierarchy, called the **decomposition dimension**.

1. In other words, abstract consistency constraints are "templates" while active consistency constraints are instances of such templates in a particular design process.
2. Not to be confused with the discipline hierarchy, Δ, defined earlier. Δ is a knowledge-level structure while the design process hierarchy is a data-level structure.

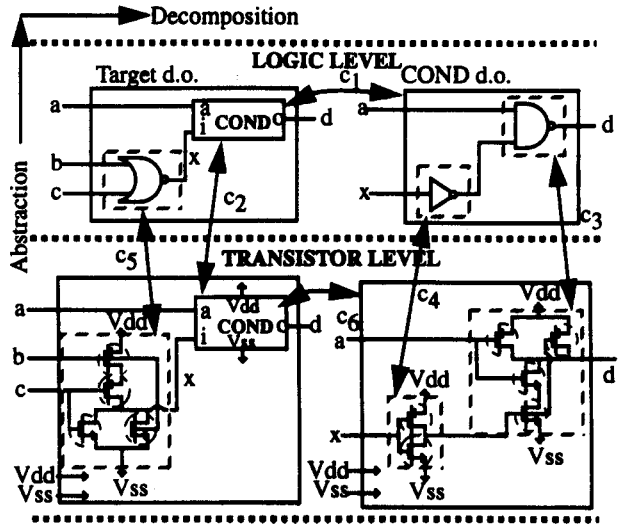

Figure 2 Illustrating the design process hierarchy

Observe that structural and functional decompositions are the only allowed mechanisms for creating new design objects and adding them to the design process hierarchy. When new design objects are incorporated into a design process hierarchy, at a given abstraction layer, a set of active consistency constraints, representing the relations among these new design objects are also instantiated in the design process. Such constraints relate the specification of the original design object, or *parent design object*, to the specifications of the *component design objects*, also called *descendant design objects*.

Figure 2 shows a snapshot of a design process hierarchy created while implementing the behavioral description given by d = COND(a, NOR(b, c)).[1] The structural primitive building blocks for the logic level are shown within a square. Observe that the structural description for the target design object constitutes a non-fundamental structural decomposition, since it contains the non-primitive structural building block "COND".[2] The structural primitive building blocks at the circuit level of abstraction are represented within a circle. As it can be seen in Figure 2, the structural description for the target design object, at the transistor level, constitutes also a non-fundamental structural decomposition, since the structural building block "COND", at the transistor level, is also a non-primitive building block. Note also the active consistency constraints, denoted c_1 through c_6, relating properties from the same and from different abstraction levels.

1. For simplicity we represent the design process hierarchy in terms of the structure description property, yet the concepts illustrated for this particular property are directly applicable to all types of properties.
2. Note that this is just a possible structural description realizing the above behavior.

7 Characterizing Design Activity

Based on the above discussion, we may view **design** as the process of deriving a complete ground level specification for an object, starting with an **initial specification**, i.e., a set of values for *some* of the properties of the design object. Let us now formalize the design activity, once an initial specification has been provided to the designer.

7.1 Design Objectives and Operators

In general, the consistency constraints among the properties that characterize the object under design are highly complex. This complexity precludes the possibility of completing the design in a single design step. The design process typically consists of a sequence of *generation design steps* and *test design steps*.

A **generation design step** typically involves the selection of a particular sub-specification (among those that have been instantiated in the design process hierarchy), the selection of any relevant active consistency constraints, and either *synthesis* or *optimization*. **Synthesis** is the generation of values for those properties that still do not have specified values, while **optimization** is the generation of new values for properties that already have a specified value. Observe that in order to further overcome the high complexity associated with some consistency constraints, generation design steps may only take into account a subset of the relevant active consistency constraints or may assume a simplified version of such consistency constraints. In these situations *inconsistent* property values, i.e. property values that violate some of the active consistency constraints, may be generated. **Test steps** are intended to detect such inconsistencies. In particular, test steps consist of selecting a particular sub-specification in which all properties have specified values, selecting all active consistency constraints associated with this sub-specification, and checking for violation of consistency constraints. When one or more consistency constraints are violated, *backtracking* or *optimization* may be considered.

Each design step is a particular solution for a design (sub)problem. These design (sub)problems are defined in terms of general **design objectives**, such as synthesis, optimization and test. (Design objectives may be further specialized until the *target properties* for the design problem become uniquely determined.) Each design objective also defines the control knowledge necessary for solving the specific design problem. This knowledge contains such pieces of information as how

the particular design objective (and thus design problem) should be decomposed into sub-objectives (i.e., design sub-problems), in case its complexity is not directly manageable. Finally, design objectives may also have their own abstract properties, aimed at conveying problem-specific additional information. Examples of objective related properties are stimuli for test (simulation) objectives and stopping criteria for optimization objectives.

An **active design objective** is a specific instance of a design objective in a particular design process. Since active design objectives are ultimately responsible for controlling the solution process of the problem they define, whenever design objective decomposition occurs the parent design objective has to supervise the set of active design sub-objectives that were generated from its own decomposition. Thus, in a design process, active design objectives are organized in a "parent-to-descendent" tree-like hierarchy.

Finally, **design operators** are design functions, implemented by means of algorithms and/or procedures, aimed at actually solving the design problems defined by the active design objectives. In other words, design operators perform the design steps.[1]

7.2 Design State and Design History

A **design state** is defined in terms of: (1) the design process hierarchy, i.e., the set of all design objects instantiated so far in the design process; (2) the set of all active design objectives currently instantiated in the design process; and (3) the set of all active consistency constraints (defined for the properties contained in the design process hierarchy and in the set of active design objectives).

Since each active design objective in the design state uniquely defines a design problem, the design state is an organized collection of design problems that mirrors the active design objective hierarchy. Any new problem (or active objective) added to the design state will, in principle, remain in the design state until the end of the design process, eventually reaching "achieved" status. Backtracking is the only way of removing a problem from the design state. A design step is thus a transition from one design state to another. If a design step is successful, the particular active design objective defining the problem solved by the design step becomes "achieved."

Finally, the **history of a design process** includes the ordered sequence of all design states visited so far in the design process, together with the operators used to modify such design states. For the complete formal definition of the design space see [1].

1. Most conventional CAD tools are examples of design operators.

8 Conclusions

We have presented a formal basis for explicit characterization of a design discipline and a unified, problem based representation of the design process. This formalism has proven useful in guiding the development of the Minerva and Clio meta-tools and in helping designers to reason about and evaluate the effectiveness of their design methodologies (for instance, in determining optimal problem decompositions and thus optimal CAD tool granularity). Moreover, this formalism provides the necessary conceptual infrastructure for the development of a new generation of meta-tools that offer highly advanced services, such as automatic learning and automatic archiving. We note in closing, that while our motivation for developing this formalism was to specifically address VLSI design problems, what has resulted is applicable to design processes in general.

9 Bibliography

[1] M.F. Jacome. *Design Process Planning and Management for CAD Frameworks.* PhD thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, September 1993.

[2] M.F. Jacome, and S.W.Director. Design Process Management for CAD Frameworks. In Proceedings of *29th ACM/IEEE Design Automation Conference.* ACM Press, 1992.

[3] J.C. Lopez, M.F. Jacome, and S.W. Director. Design Assistance for CAD Frameworks. In Proceedings of *First GI/ACM/IEEE/IFIP European Design Automation Conference.* ACM Press, 1992.

[4] H.A. Simon. *The Sciences of the Artificial.* The MIT Press, 1981.

[5] E. D. Sacerdoci. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115-135, 1974.