16

Planning and Managing Multi-disciplinary and Concurrent Design Processes

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, PA 15213

1 Introduction

It is time to move beyond the development of CAD tools that only aid designers in solving specific synthesis, analysis, and/or optimization design problems, but to develop tools that aid designers in planning and managing the increasingly complex design process itself. Specifically, we need to develop meta-tools capable of capturing design methodologies, and of intelligently making use of such methodologies while planning and managing the design process.

The more complex the design process is, the more useful can be a design process planning and management meta-tool capable of dynamically and coherently implementing an adequate design methodology throughout the entire design process. For instance, some design methodologies include mechanisms for controlling iterative design processes to ensure that complex, interdependent design constraints are satisfied. If, during the course of the design process, a constraint i s not satisfied, such design methodologies prescribe a backtracking strategy (i.e., indicate what design decisions should be re-accessed), based on the specific output produced by the analysis tools, and on the current coverage of the solution space (i.e., on the solutions already tried). In order to effectively explore methodologies embodying complex decision making, throughout the entire design process, meta-tools capable of creating an integrated representation of the complete design process are required. In this paper we will discuss the design process planning and management meta-tool Minerva, [1] [2], which is capable of capturing and dynamically implementing arbitrarily complex design methodologies.

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

This paper is organized as follows. In the next section we discuss hardware/software codesign, an area in which we feel that design planning and management is essential. Then, in Section 3, we point out the particular aspects of such a methodology that make it almost intractable, if only conventional CAD tools are available in the design environment. In Section 4 we present the formalism of design that constitutes the basis for Minerva. Then, in Section 5, we provide some details on Minerva's implementation, and illustrate some of its planning and managing services when supporting the codesign methodology described in Section 2. We finish with some conclusions in Section 6.

2 Example: Hardware-Software Codesign

In this section we review the h/s codesign methodology described in [3] to motivate the need for meta-tools for design process and planning.

Assume that we are given a behavioral description of a system we wish to realize. Such a description is typically a collection of processes written in a hardware description language such as Verilog, or a programming language such as C. The first step of the h/s codesign process is *behavioral partitioning, which* is aimed at determining which chunks of the original behavioral description, also called *tasks*, should be implemented in hardware and which chunks of behavioral description should be implemented in software. Observe that determining the granularity of the tasks themselves, i.e., the number and size of the hardware and software components, is also part of the behavioral partitioning problem. In general, partitioning is a highly *combinatorial* optimization problem. Moreover, even if a *satisfycing partition* (as opposed to an optimal one) is found that meets all requirements (e.g. speed and cost), these requirements can only be broadly *estimated* at the moment the partitioning itself takes place. This is so because at this point in the design process there is no concrete mapping of the resulting partition onto a specific design object. Thus behavioral partitioning is an *ill-defined* problem and it is very difficult, if not impossible, to develop *exact methods* for determining the best partitioning. Rather, *heuristic methods* have been developed to for determining an adequate partitioning in acceptable time. These methods work by dramatically pruning the search space but do not guarantee that a satisfycing partition, if one exists, will be found.

Given a suitable partitioning, the next step in h/s codesign is to group the resulting set of tasks into a new set of processes — those to be implemented in hardware and those to be implemented in software. This step is called *style selection.* As was the case with partitioning, style selection can only be achieved in a reasonable amount of time if heuristics are used to chose an adequate set of hardware and software processes.

After style selection, a *behavioral synthesis* step is used to realize the set of hardware components. This step, as with previous steps, is highly complex — recall that synthesizing an RTL implementation of an algorithm meeting speed, cost, and path requirements is an NP complete problem. Again, heuristic methods are used to overcome this difficulty.

To summarize then, to guarantee that a codesign process will be completed in an acceptable amount of time, heuristics must be employed. Unfortunately, heuristic methods cannot guarantee that satisfycing solutions, if they exist, will indeed be found. And certainly heuristics cannot guarantee that such a solution will be found on the first try.

Clearly the earlier a non-satisfycing solution is detected, the better. *Analysis* and/or *verification* tools can be used at various stages of the codesign process to assess the potential of a particular candidate solution to meet the design requirements. If the design is found to be non-satisfycing, the designer must *backtrack* (i.e., return to a previous design state), in order to try to adequately modify the design in order to bring it closer to a satisfycing one. Since the need for backtracking may occur repeatedly during a regular h/s codesign process, the h/s codesign process is iterative in nature.

When a non-satisfycing solution is detected, the designer must choose a *backtracking strategy* to determine which previous design state to return to, and then determine what should be modified. Properly choosing a backtracking strategy can be as critical to the efficiency and success of a h/s codesign process as is properly performing any of the "active" design steps (such as behavioral partitioning, or style selection). An example will illustrate this point.

Suppose that while performing behavioral synthesis for a particular hardware component the designer realizes that the resulting RTL description does not have the potential to meet one of the requirements. There are three backtracking alternatives: (1) re-synthesize the specific component, trying to impose adequate restrictions on the behavioral synthesis process; (2) modify the behavior for the particular component under design, by partially re-doing style selection; or (3) modify the particular tasks, by partially re-doing partitioning.

Observe that the nature of the requirements that are not satisfied, and the way these requirements relate to one another, determine which backtracking strategy to pursue. For instance, it is important to check if some trade-offs are possible among the candidate solution's current unsatisfied requirements. If such trade-offs are possible, this means that the current candidate solution may be quite close to a satisfycing one. Other-

wise, it may be that the particular region of the solution space being explored does not hold any acceptable solutions. Observe that partially re-doing partitioning allows the designer to generate significantly different solutions, while re-synthesizing just the specific hardware component only allows the designer to move very locally in this space. Yet re-synthesizing is much less costly than partially re-doing style selection, which in its turn is much less costly than re-partitioning. Clearly, a more costly backtracking step should only be tried if the immediately less costly one proves to be incapable of generating a potentially satisfyc-ing solution. Moreover, the solution space must be explored in a *controlled* and *coherent* fashion, such that, for instance, if a solution does not exist for a particular problem the designer is capable of identifying the situation in reasonable time.

3 Advanced Plan and Management Services

Let us highlight some key aspects of the codesign methodology described above. First of all, h/s codesign involves solving a significant number of complex design sub-problems, some of which may be addressed concurrently. Adequately planning a h/s codesign process thus involves choosing which methods and strat-egies to use for problem decomposition, during active problem solving, and then properly applying these for the solution of the codesign problem, while guaranteeing that consistency is preserved. As discussed below, maintaining consistency ultimately requires the ability to adequately characterize and handle sub-problem dependencies throughout the design process. Additionally, h/s codesign involves manipulation of quite different types of design objects, and is thus multidiscipinary in nature, thereby requiring a capability for handling, in an integrated form, the design of fundamentally distinct objects. Finally, h/s codesign is an iterative process. Adequately deciding on which backtracking strategy to pursue, given a specific failure, requires reasoning about the current overall state of the design process and, particularly, about the current coverage of the solutions space. It also requires, again, an adequate handling of sub-problem interactions, in the sense that such sub-problem interactions establish a causal chain between the various design representa-tions and decisions created during the course of the design process. Such causal chains would significantly help in tracing back the design decisions that may have lead to a particular failure.

Summarizing, then, the effective application of h/s codesign methodologies requires a *unified* view of the design process, naturally integrating the design of objects of different nature, and the various kinds of prob-lem decomposition mentioned in the previous section. Such a unified view is also needed to support deci-sion making about backtracking strategies, and the actual implementation of such strategies. (Recall that the some of the strategies under consideration in h/s codesign processes may traverse several phases of the codesign process). Finally, guaranteeing consistency during active problem solving, as well as supporting

reliable backtracking, requires the capture of sub-problem dependencies in this "unified" design process representation.

We have just characterized the basic infrastructure that is necessary for an effective implementation of the h/s codesign methodology described in Section 2. As will be shown in the next two sections, the Minerva design process planning and management meta-tool realizes such an infrastructure making it capable of fully incorporating and applying the h/s codesign methodology described in Section 2.

4 Minerva's Design Formalism

Minerva is based on a formalism of design that allows for a complete and general characterization of design disciplines, and for a unified, problem-based representation of design processes. Space precludes us from describing this formalism in detail. We will however introduce the principal concepts. Readers interested in a more detailed description should consult references [4] and [1].

Figure 1 illustrates the basic elements of Minerva's design process representation: design objects, design objectives, operators and consistency constraints. Objects under design, also called *design objects*, are represented and characterized by a collection of *properties* (or features). Such properties are partitioned by abstraction levels, creating one or more *facets* of the design object. Figure 1 shows a design object of the class "processing unit", with two facets: "algorithmic level," and "register-transfer level." The properties characterizing this design object (i.e., 'stored' in its two facets), are: "cost", "delay", "path", and "behavioral description." *Design objectives* define general classes of design problems. A general objective "synthesis" is shown in Figure 1.

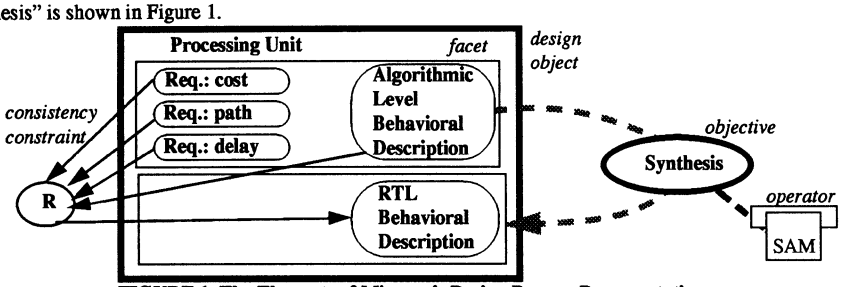

FIGURE 1. The Elements of Minerva's Design Process Representation

A design *problem* is defined in terms of a set of design objects (or, more specifically, a set of properties among those characterizing these design objects) and a design objective (viz. the problem to be solved within the context of the previous properties). Design problems are solved by *operators*, i.e., CAD tools.

So, in Figure 1 the CAD tool SAM [5] will be used to solve the behavioral synthesis problem defined by the design objective "synthesis" associated with a design object of the class "processing element" represented at "algorithmic level."

Problems can be directly solved if a suitable operator is available, and if the operator succeeds. Otherwise, an *impasse* has occurred. According to the nature of the impasse, it maybe resolved either by *problem decomposition* or by *backtracking*. Problem decompositions that take place during the design process generate a hierarchy of design problems where "parent" problems directly control their sub-problems. This problem hierarchy completely characterizes a particular design process. Figure 2 illustrates the fist level of problem decomposition for a h/s codesign process using the methodology described in Section 2. The control hierarchy is symbolically represented in Figure 2 by the arrows connecting "parent" objectives to their sub-objectives. (Observe how problem decomposition strategies can be easily captured and implemented in the high level, semantically rich problem-based representation of design processes.)

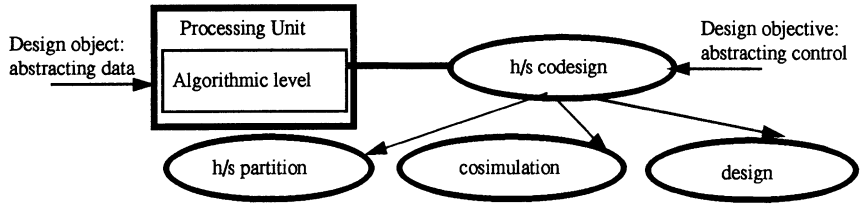

FIGURE 2. Hierarchy of Design Problems

Finally, *consistency constraints* express relations among properties, as symbolically illustrated in Figure 1. They constitute the basis for expressing sub-problem dependencies, as will be explained next. Observe that one of the forms by which problem decompositions make sub-problems treatable is by "artificially" assuming static values for certain properties (in the context of particular sub-problems), when indeed such values may yet not be static at all, or may even be impacted (to a certain extent) by the very decisions that will be made in the context of the particular sub-problem. This is the reason why, sometimes, sub-problem solutions may be inconsistent -- because the assumptions on the values of properties excluded from these sub-problem proved, later on, to be incorrect. Let us give an example. While implementing a particular chip component, the designer takes only into consideration the specific requirements stated for that particular component, such as the area reserved for it during layout planning. However, if the team of designers actually fails in generating all of the remaining chip components meeting their individual requirements, the design effort for the first component might end up being useless. We may say then that there is a "second

order" dependency among some of such component requirements -- for instance, among the area of each chip component and the areas of the remaining components of the same chip. The properties "temperature" and "dissipated power" are another example of such a decomposition strategy -- their values tend also to be treated as independent (while addressing different design sub-problems) when in fact they clearly depend on each other.

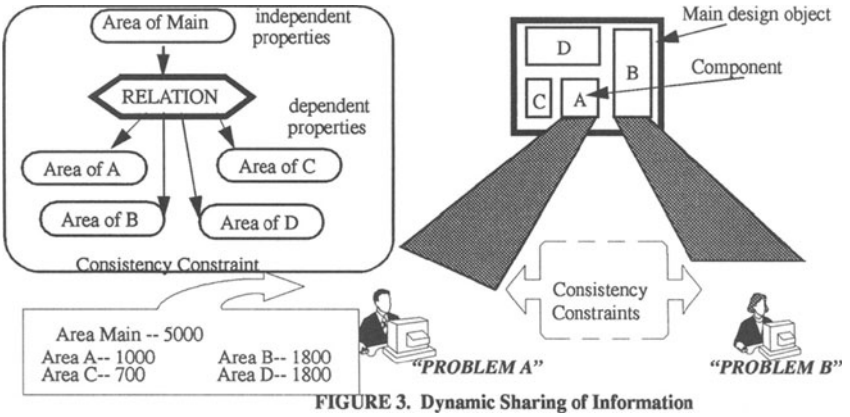

FIGURE 3. Dynamic Sharing of Information

Consistency constraints capture such "second order" relations among properties, and thus express sub-problem dependencies. Figure 3 shows a consistency constraint representing the relation between the areas of the components of a chip and the area of the chip itself. The semantics is as follows. Given a set of independent property values, and a relation, the set of dependent values must be chosen such that the relation stated in the constraint is verified. Otherwise, we say that the constraint was violated. Note that the later such inconsistencies are detected, the higher is the cost of backtracking (i.e., returning to a previous design state) in order to restore consistency, i.e., to adopt adequate assumptions on property values.

5 Minerva

In this section we discuss how Minerva's general capabilities adequately respond to the problem of planning and managing complex h/s codesign processes. We start by briefly discussing Minerva's general organization. As indicated in Figure 4,[1] Minerva has a set of basic components that serve all of its *sessions*.

1. Since space preclude us from presenting an exhaustive description of such an organization, in this paper we only address the fundamental aspects of it and remit interested readers to [2]

Such components include a *knowledge base*, that characterizes the design (sub)disciplines that are relevant to the particular design environment, and also an interface to an existing *executive meta-tool*, such as the one described in [6], that is responsible for executing CAD tools, storing design data, etc.

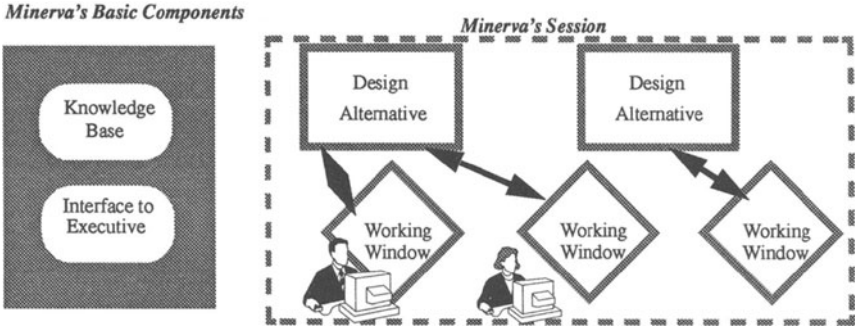

FIGURE 4. Minerva's General Organization

A session in Minerva, on the other hand, encompasses all of the *design alternatives* simultaneously under development for a given, central, design problem, as indicated in Figure4. Such a design problem can be, for instance, the design of a processing unit implementing a particular algorithm. Each design project ongoing in a particular design environment should thus have one Minerva session assigned to it.

Support of concurrent design requires guaranteeing that each design alternatives be *asynchronously* accessed by each of the designers assigned to the session's project. Figure 4 shows the general organization of a Minerva session -- in such an organization, one independent design alternative process exists for each solution being concurrently developed. Designers interact with these design alternatives through a dedicated agent called a *working window*. Figure 4 indicates that the design alternative depicted in the left is being concurrently accessed by two designers. "Working windows" and "design alternatives" cooperate (in a form that is transparent to the designer) in order to provide all of the synchronism features needed for supporting such concurrent design activity.

Figure 5 is a symbolic view of a hardware/software codesign process, as it would be shown to the designer through a working window. In this view, objectives already achieved are symbolically represented as dotted bubbles, while objectives not yet achieved are represented as solid bubbles. Objectives that can be concurrently addressed, at the current point in time, are shadowed. The symbolic view shown in Figure 5 corresponds to a snapshot of an advanced stage of a codesign process at which designers are already developing

the set of hardware components that will implement the "hardware" tasks derived during the h/s partitioning (see Section 2).[2] Each of the shadowed "design" sub-objectives, together with their associated tasks

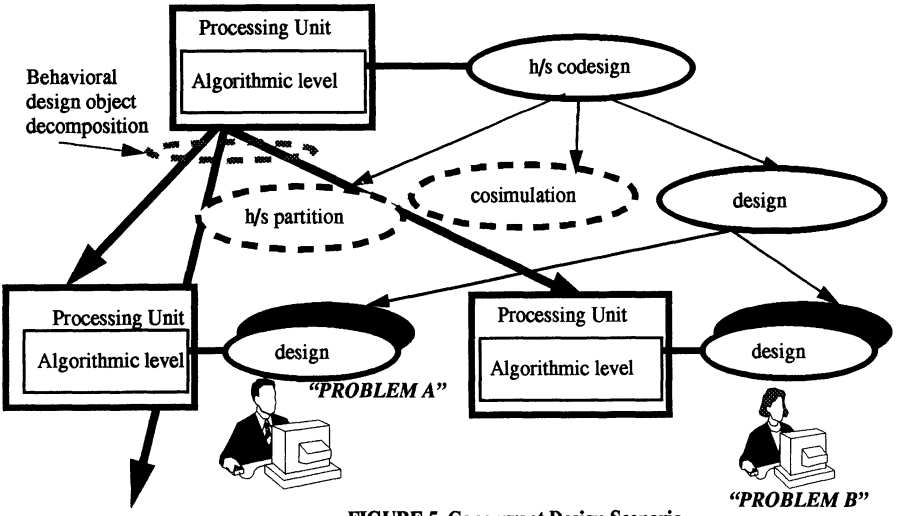

FIGURE 5. Concurrent Design Scenario

(stored within the processing units coupled to the sub-objectives), constitutes a design sub-problem that can be addressed concurrently at this stage of the codesign process -- in Figure 5 these sub-problems are labeled "Problem A" and "Problem B".

During active problem solving, when the designer selects a particular design problem to be addressed, the design information (i.e., set of properties) directly relevant for the accomplishment of the particular problem is delivered to the designer, through the working window. Additionally, Minerva's consistency constraints allow this same designer to dynamically monitor all of the properties that, not being directly relevant to the problem under consideration, may in certain circumstances have some impact on it, as discussed in Section 4. Figure 3illustrates this dynamic sharing of information that is provided in Minerva. Observe that the designer working on problem "A" is informed ("on-line") about the status of some of properties outside of the scope of this problem. Specifically, the designer is shown the amount of area being used by the designs worked on by other designers.

2. For simplicity purposes, a number of intermediate problem decompositions (mainly for the objectives h/s partition) are omitted from Figure 4.

During backtracking, consistency constraints are used by Minerva to show how dependent properties relate to their assumptions, i.e., independent properties. In particular, consistency constraints are used by Minerva to: (1) guide the selection of convenient backtracking points given specific failures; and (2) implement the backtracking steps themselves, i.e., propagating their effects throughout the design process. Let us give an example. Assume that in Figure 3the designer implementing "component A" concludes that the area originally planned for this component is impossible to achieve. The consistency constraint depicted in the figure, and all the others involving the particular design object failing to meet its requirements, will directly assist the designer in deciding: (1) what trade-offs can be tried -- for instance, what components could have their planned area reduced, in order to create space for Component A, or, (2) how the global problem can be redefined -- i.e. the requirements for the chip itself be changed. The concept illustrated in this example is generalizable to representing more complex constraints.

6 Conclusions

In this paper we demonstrated that progress in certain key design areas, such as those involving complex embedded systems, can only be significant if adequate design process planning and management meta-tools are available (together with the more conventional CAD tools). We then described Minerva, a meta-tool that is capable of capturing the fundaments of arbitrarily complex design methodologies, and of intelligently making use of such methodologies, while planning and managing the entire design process. We further discussed how Minerva naturally supports concurrent and multidisciplinary design, in the context of sequential or iterative design methodologies.

7 Bibliography

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

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

[3] D.E. Thomas, J.K. Adams and H. Schmit, "A Model and Methodology for Hardware-Software Codesign," *IEEE Design & Test of Computers*, September 1993, pp. 6-15, September 1993.

[4] M.F. Jacome and S.W. Director, "A Formal Basis for Design Process Planning and Management", *Proc. of ICCAD Conference*, Santa Clara, CA. November 1994.

[5] R. Cloutier, and D. Thomas, "Synthesis of Pipelined Instruction Set Processors" In Proceedings of the ACM/IEEE 30th Design Automation Conference, ACM, 1993 (sam)

[6] P.R. Sutton, J.B. Brockman., and S.W. Director, "Design Management Using Dynamically Defined Flows," in Proc. of *30th ACM/IEEE Design Automation Conference,* ACM Press, 1993.