

A Design Process Management Language

Peter R. Sutton and Stephen W. Director

Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh PA 15213

Abstract

A language for describing design discipline characteristics such as abstraction levels, design object classifications and decompositions, design objectives, and design methodologies is proposed. The language, DDDL, has led to the development of Minerva II, a design process management tool, which, when configured with a design discipline description, can be used to aid teams of designers in the solution of complex multi-disciplinary design problems. This paper describes the design discipline characteristics which DDDL can capture and gives an example of how Minerva II can be used to help designers.

1 Introduction

Electronic design (CAD) frameworks have evolved from *design tool and data management* [2] to *design flow or methodology management* (e.g., [6], [7],[8] and [9]). The services provided by these frameworks, however, are no longer sufficient for today's complex design processes due to the trends towards multi-disciplinary concurrent design and towards carrying out design at higher levels of abstraction. It is necessary to better manage data interactions between levels of abstraction, e.g. reflecting delay information determined at the layout level of abstraction back into behavioral synthesis, and between designers working in different disciplines, who may be using disparate framework environments. For these reasons, the design management services offered by CAD frameworks must move to a higher level of abstraction. We call this level the *design process management* level [1]. Design process management provides support for, and control of, all aspects of the solution of design problems including conceptual (or exploratory) design, problem decomposition, backtracking, constraint propagation and management, and design history management. It also encompasses support for existing design management services provided at lower levels of abstraction, including data, tool, flow and methodology management.

The provision of these design process management services requires some *knowledge* of design discipline characteristics and how they interact. We have developed a language called DDDL (for **D**esign **D**iscipline **D**escription **L**anguage) to capture these design discipline characteristics. DDDL has grown out of the work by Jacome [3],[4] which, in proposing a formal theory of design, established that design processes can be described in terms of these characteristics. The Minerva Design Process Manager [5] developed by Jacome, however, did not provide a mechanism for capturing design discipline knowledge and was not able to be configured for arbitrary design disciplines. DDDL was developed to overcome this deficiency. Minerva II, which implements a DDDL parser, is easily configured with design discipline information and can manage design problems in that discipline (or disciplines). DDDL has also allowed for the realization of a more intuitive user interface in Minerva II. The new user interface allows for simpler management of design problems and improves the efficiency of the design cycle.

Before proceeding, it is useful to discuss how our work relates to other work in this field. As stated above, design process management is a level of service above that provided by current commercial and research CAD frameworks. These frameworks (e.g. [7], [8] and [9]) provide flow-based execution management and are able to manage simple methodologies but do not provide the higher level management of decomposition, backtracking, conceptual design and constraint propagation that is essential for more efficient design cycles. The lower level services provided by these systems are still important though, as it is through these services that a design process manager is able to manage tool executions and data.

The remainder of this paper is organized as follows. In Section 2 we present the design discipline characteristics that can be described using DDDL. Unfortunately, space precludes us from describing details of the language itself. In Section 3 we describe how Minerva II can help the designer solve a design problem. Finally, in Section 4, we draw some conclusions.

2 Design Discipline Characteristics

In this section we describe the design discipline characteristics which DDDL can capture. First, we define what we mean by a design discipline: a *design discipline* is a “field of design” encompassing all design objects of some broad type. For example, the “Electronics System” design discipline encompasses the design of all types of electronic systems.

A **design domain** is a design object classification or type. For example, “Hardware System” and “Processor” are design domains. Design objects in a given design domain are characterized by having certain types of *properties*. For example, design objects in the “Processor” design domain may have the properties “instruction set”, “addressing modes”, “timing specifications” and “unit cost”. The design domains in a design discipline can be arranged in a *design domain hierarchy*, where the top level design domain corresponds to the design discipline itself, and lower level design domains inherit the properties of the higher levels. For example, design objects in the design domain “Digital Signal Processor” may have all of the properties of the “Processor” design domain, along with the additional properties “FFT speed” and “Maximum sample rate”. We say that the design domain “Digital Signal Processor” is a *specialization* or *subtype* of the design domain “Processor”.

A **design objective** is a design function to be carried out. For example, “Synthesize”, “Verify”, and “Optimize” are all design objectives. Design objectives are context dependent, that is, the design domain, target abstraction level and starting point all impact the meaning of a particular objective. This will become clearer below in our discussion of a design problem.

A **design problem** can be specified in terms of a design objective for a target design domain at a target abstraction level for some given input properties. The *target* design domain is the design object type to be created (or operated on), and the *target* abstraction level is the abstraction level at which we wish to represent our final design object. An example of a design problem is: “Design a hardware system at the layout level having a given behavioral description (at the behavioral level of abstraction), using a particular 0.5 μ m CMOS process and having a required area no larger than 1cm²”.

Design problem decomposition is the act of breaking up a design problem into smaller, easier-to-solve problems. Design problem decomposition can be of two forms - design domain decomposition or design objective decomposition. *Design domain decomposition* is the breaking up of the design object into sub-objects with the same design objective being applied to each. For example, the design problem “design a hardware system” can be decomposed into the sub-problems “design a processor system” and “design a memory system”. *Design objective decomposition* involves breaking up the design goal into sub-goals which apply to the same whole design object. For example, the design problem “design a hardware system” could be decomposed into the sub-problems “specify a hardware system”, “synthesize a hardware system” and “verify a hardware system”.

When decomposing a problem, **design constraints** are often generated. Design constraints relate the different parts of a design object. For example, when decomposing a hardware system into a processor system and a memory system, several constraints may be generated. One constraint might apply to the interface between the two parts, ensuring that the two sub-systems will work together. Another constraint might relate the area of the two parts to that of the whole, ensuring that the restriction on the overall hardware system’s area is not unknowingly violated.

A **design methodology** is a specific method, approach and/or set of rules to be followed when solving a given design problem. Typically a company or group will specify a methodology for designers to follow so as to ensure consistent design results. DDDL is able to capture several types of methodology information, including the design problems the methodology is available for; the particular design domain and design objective decompositions which are to be used; default design options (such as fabrication technology); and the tools which must be used (if available) for certain tasks.

3 A Design Process Management User Interface

The development of DDDL has led to the development of Minerva II, a new version of the Minerva Design Process Manager [5] with a completely revised user interface. In this section, we illustrate, through user interface screen-shots, how Minerva II, configured with a DDDL design discipline description, is able to manage a design process.

It is important to note that Minerva II operates at the *design process level* - a level of abstraction above that of today’s CAD frameworks. Minerva II, however, can use the executive and data management services of most

existing CAD frameworks. This is accomplished through a “framework encapsulation” interface enabling Minerva II to use the services which the executive provides. A prototype framework encapsulation of the Hercules Task Manager [7] has been implemented. We forego a discussion of this type of encapsulation so as to concentrate on the designer’s view of design process management.

To solve a problem using Minerva II, the designer must first define the problem. This involves the selection of a target design domain and a design objective. This is achieved through a problem definition window as shown on the left in Fig. 1. On the left of the window, the hierarchy of design domains declared for the design discipline is shown. This hierarchy is directly derived from the DDDL description used to configure Minerva II. On the right side of the window is a listing of all the design objectives defined for the design discipline. Once a design domain and design objective have been selected, the target abstraction level may be specified (as shown on the right in Fig. 1) or, by default, chosen to be the lowest abstraction level for that design domain/design objective combination. It is also possible for the designer to specify a design methodology to follow. If a design methodology is specified, any restrictions contained within the methodology will be enforced by the system during the problem solving process.

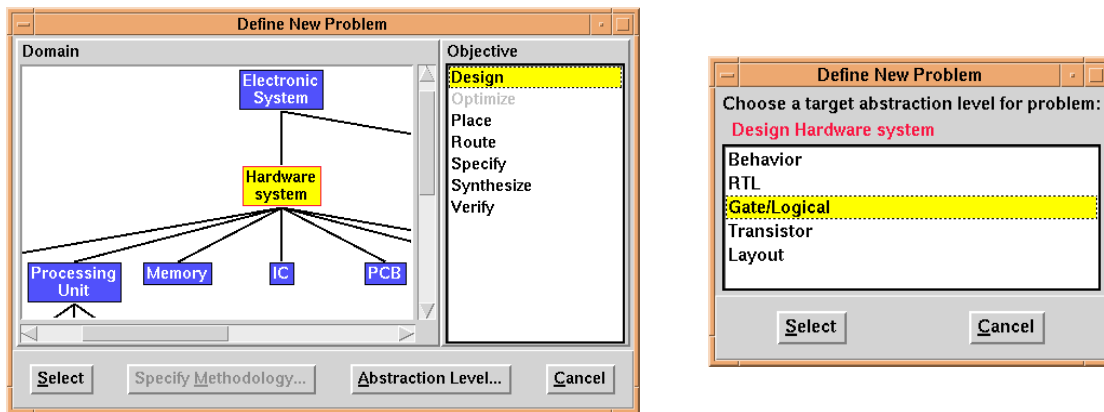
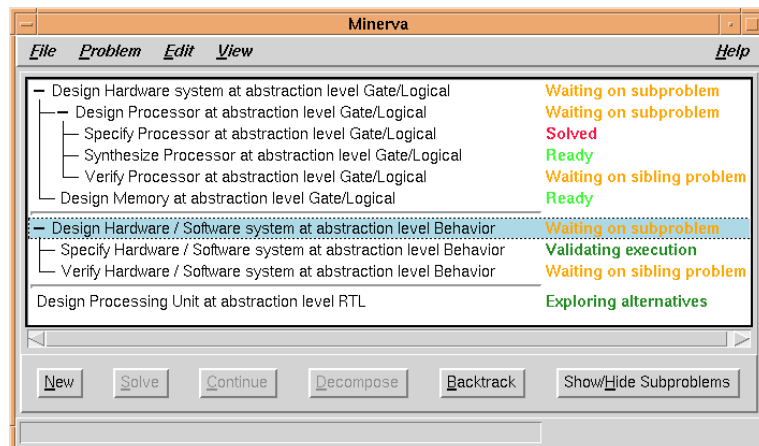


Fig. 1. Problem definition involves the selection of a design objective, a target domain and optionally a target abstraction level. The figure on the left shows the selection of the design domain “Hardware system” and design objective “Design”. The figure on the right shows the selection of the “Gate/Logical” abstraction level as the target abstraction level for this problem.

Minerva II’s problem status and selection window, shown in Fig. 2, is used to select a problem to solve (or to join a solution process in progress). All active problems are displayed and sub-problems of any given problem can optionally be displayed. Adjacent to each problem description is the current status of that problem. Any problem that is ready to be addressed can be selected for solution and any problem whose solution is in progress can be continued. Multiple problems (or sub-problems) can be addressed concurrently (by the same or different users) if desired.

Fig. 2. Minerva II’s Problem Status and Selection Window. All active design problems (and their sub-problems) in the system are shown, with their status.



After problem selection, five problem solving steps take place. The first step is *alternatives exploration* where the designer explores possible trade-offs for the design problem at hand and specifies any restrictions on the problem (e.g. maximum power and area). Once this step is finished, *design plan generation* occurs. It is during this step that the framework executive system is queried as to which tools are available to directly solve the design problem under consideration. If appropriate tools are available, then Minerva II performs *plan validation* by querying the designer about which of the generated design plans are acceptable. If any are acceptable, Minerva II instructs the framework executive system to perform *plan execution*. After execution, *execution result validation* takes place.

If at any time an impasse occurs (for example, no tools are available to solve the design problem directly, or the user rejects the generated design plans or execution result), Minerva II facilitates either decomposition of the design problem into design sub-problems, or backtracking to an earlier design state. We discuss each of these in the next two paragraphs.

Design problem decomposition can be achieved either through design domain decomposition or design objective decomposition. When opting to decompose, the user is shown all available decompositions. Possible decompositions are determined from the DDDL design discipline description that is used to configure Minerva II, with the choice being limited if a methodology was specified during design problem definition. When a design domain decomposition is chosen, the sub-problems created appear in the Minerva II problem status and selection window, and may be selected for solution when appropriate.

When an impasse occurs (or at any time if desired), the designer may backtrack to a previous design state. When backtracking, the designer chooses a problem to backtrack to. Any design problem in the hierarchy above the current design problem may be backtracked to - in which case the system returns to that design problem and makes it ready to be addressed again. If the current design problem is chosen to backtrack to, the designer may return to any previous problem solving step for that design problem (e.g. alternatives exploration).

4 Conclusions

We have proposed a language called DDDL and described the design discipline characteristics which it can capture. DDDL allows the capture of design discipline knowledge into a simple human and machine understandable form. The description, once captured, can be used by design process management software to help designers manage the solution of their design problems. Support can be provided for managing constraints, backtracking and problem decomposition among other things. As design problems become more complex, these capabilities will be increasingly important for the success of electronic design processes. We have also illustrated how the Minerva II Design Process Manager, with an improved user interface, can be used to manage the problem solving process.

References

- [1] J. B. Brockman, T. F. Cobourn, M. F. Jacome, and S. W. Director, "The Odyssey CAD Framework," *IEEE DATC Newsletter on Design Automation*, Spring 1992.
- [2] D. S. Harrison, A. R. Newton, R. L. Spickelmier, and T. J. Barnes, "Electronic CAD Frameworks," *Proceedings of the IEEE*, vol. 78, no. 2, Feb. 1990.
- [3] M. F. Jacome, *Design Process Planning and Management for CAD Frameworks*. Ph.D. Thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, Nov. 1993.
- [4] M. F. Jacome and S. W. Director, "A formal basis for design process planning and management," in *Proceedings of 1994 IEEE International Conference on Computer Aided Design*, Nov. 1994.
- [5] M. F. Jacome and S. W. Director, "Design process management for CAD frameworks," in *Proceedings of 29th ACM/IEEE Design Automation Conference*, June 1992.
- [6] S. Kleinfeldt, M. Guiney, J. K. Miller, and M. Barnes, "Design Methodology Management," *Proceedings of the IEEE*, vol. 82, no. 2, Feb. 1994.
- [7] P. R. Sutton, J. B. Brockman, and S. W. Director, "Design Management Using Dynamically Defined Flows," in *Proceedings of the ACM/IEEE Design Automation Conference*, June 1993.
- [8] K. O. ten Bosch, P. Bingley, and P. van der Wolf, "Design Flow Management in the NELSIS CAD Framework," in *Proceedings of 28th ACM/IEEE Design Automation Conference*, June 1991.
- [9] P. van den Hamer and M. A. Treffers, "A data flow based architecture for CAD Frameworks," in *Proceedings of 1990 IEEE International Conference on Computer-Aided Design*, Nov. 1990.