15

Design Flow Management: More than Convenient Tool Invocation *

Olav ten Bosch, Pieter van der Wolf and Alfred van der Hoeven

Delft University of Technology
Department of Electrical Engineering / DIMES
Mekelweg 4, 2628 CD Delft, The Netherlands

Abstract

The term design flow management is sometimes used for facilities that hardly offer more functionality than showing a graph of tool icons that describes the preferred order in which these tools should be executed. In this paper, we argue that a design flow management system (design flow system, for short) can and should be much more than that. If a design flow system supports the definition and visualization of data dependencies between tools, distinguishes between different design functions of a tool and guarantees the correctness of a design with respect to a configured design flow, it is more suited for use in a real-world design environment. In addition, a powerful design flow system should take the hierarchical decomposition and the different representations of a design into consideration and employ an intuitive mechanism for user-interaction. In this paper, we highlight these aspects of design flow management, based on the experiences gained by building design flow management in the NELSIS CAD Framework.

1 Introduction

In recent years, design flow management (sometimes called design methodology management) has attracted a lot of attention as one of the new technologies in the area of CAD frameworks. Although a variety of design flow systems with different capabilities have been presented (for an overview, see [1]), we feel that the benefit of using a design flow system is still underestimated. Design flow management can not only be used to invoke design tools in the right order, it can also be used to do *design tracking* (keeping track of the state of design and presenting it to designers in an attractive way), *constraint handling* (permit constraints on the design process to be defined and enforced), *guidance* (advise designers about further design steps) and *automation* (automatically execute parts of a design trajectory).

In this paper, we present our view on what functionality a powerful design flow system should have in order to be of use in a multi-tasking, distributed design environment, where multiple designers work with multiple design tools. We describe a number of aspects of design flow management to be taken

*This research was supported in part by the commission of the EU under ESPRIT contract 7364 (JESSI-Common-Frame).

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

into account when building a design flow system that has more functionality than just convenient tool invocation. For each aspect we describe its implications on the underlying flow model (i.e. the constructs to describe a design flow and the rules for design flow execution) and on the architecture of the design flow system (i.e. the design flow components and their integration into a CAD framework). We address the following aspects (there may be more, such as hierarchical flows):

- *data dependencies*:
  Temporal dependencies between design tools describe a design environment to a limited extent only. In addition to the order in which design tools should be invoked, it is also important to invoke them on data of the right type and on data that has the right state. The use of data dependencies in a design flow model increases the modeling power of the system, which permits a more accurate description of the design process.
- *fine-grain design flow management*:
  Instead of describing the design process in terms of tools, it is sometimes necessary to address functions of a tool. This yields a more detailed description of the design process. We describe how a design flow system can refine its description of the design process from a *coarse-grain* level, the level of tools, to a *fine-grain* level, the level of design functions of tools.
- *support for restrictive design flows*:
  Design flow systems that only use a design flow as a guideline for designers, are limited in the sense that they can not be used to configure and enforce design strategies. We claim that the enforcement of a predefined design flow can seriously improve the use of a design flow system without becoming too restrictive in experimental design stages.
- *complex design constraints*:
  For a seamless integration of design flow management into a design system, it is useful to offer design flow primitives that address other aspects of a design, like its hierarchical decomposition and its different representations. Such a design flow system can inform designers more precisely about the executability and successful completion of design activities. We describe a design flow model that supports the definition of design constraints that refer to the hierarchical multi-view structure of a design.
- *flow coloring*:
  An important part of the success of the design flow system is determined by its methods of user interaction. We present an approach that maximally exploits the configured design flow to present the state of the design process in an intuitive way to the end-users and that can be used in a concurrent design environment since it responds intelligently to changes in the design state.

To describe how these different aspects of design flow management contribute to a powerful design flow system, we take an incremental approach. We start from a 'minimal' design flow system and extend its functionality bit by bit to arrive at a design flow system that exploits the full potential of design flow management. Finally, we formalize the presented design flow model in a data schema.

2 A Minimal Design Flow System

The minimal design flow system from which we start is nothing more than a stand-alone framework tool that displays a graph of tool icons from which tools can be invoked. The graph expresses temporal dependencies between tools, which indicate the preferred order of execution. The flow model may use constructs like *and* and *or* to enable the definition of more complicated temporal relationships. An example of such a design flow is shown in figure 1. Tools are shown as rectangles and temporal dependencies are shown as arrows. This design flow shows a layout verification trajectory, with a

layout editor, a design rule checker, a layout to circuit extractor, a stimuli editor, a simulator and a simulation result viewer. The **and** keyword indicates that the simulator can only be executed after an extraction has been performed *and* a stimuli description has been selected.

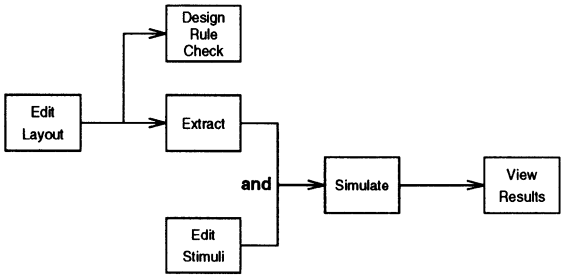

Figure 1: A design flow configured in a minimal design flow system.

3 Data Dependencies

A limitation of the minimal design flow system is that a design flow does not contain any explicit information on the data behavior of the design tools. It uses temporal dependencies, which express the executability of a tool in terms of the completion of tool runs. However, the executability of a tool may actually depend on a single piece of data produced within a tool run. Therefore, in order to describe the subtle dependencies among design tools with specific data access requirements, a design flow system should support the definition of *data dependencies*.

The addition of data dependencies to a design flow model does not mean that temporal dependencies should be discarded completely. They are inevitable to describe conditions that have no relation with data access at all. For instance, a requirement that an expensive system simulation may only be performed if some basic pre-checks have been performed, can only be described by a temporal dependency. A powerful design flow model should support data dependencies to guarantee the correctness of the design flow with respect to the data access requirements of design tools and temporal dependencies to allow the definition of conditions that do not directly originate from the data requirements of the design tools.

In order to support data dependencies tools can no longer be described as black boxes. Their behavior with respect to the data they access should be taken into consideration. The design flow model should be extended with a construct to denote data access or production. We use the term *design object* for a unit of design data for which the access and production is administered by the CAD framework and extend the flow model with a *port* construct (see also [2]) to describe access to design objects. A port is either an input port, describing read access to a design object, or an output port, describing the production of a design object. A data dependency can be described by connecting a number of output ports and a number of input ports to a *channel*. Temporal dependencies can be described in the same way as data dependencies, with the exception that no design object is read or produced via the ports of the channel. Figure 2 shows a design flow with data dependencies for the example design environment of figure 1. Ports are shown as small rectangles. An interesting observation is that the *and* construct in figure 1, which was needed to specify that 'simulate' can only be executed if 'extract' *and* 'edit stimuli' have been executed, is replaced by two distinct input ports for 'simulate', which determine its data access requirements. In fact, the *and* construct in figure 1 specifies the data access

requirements of 'simulate', which are made explicit in figure 2.

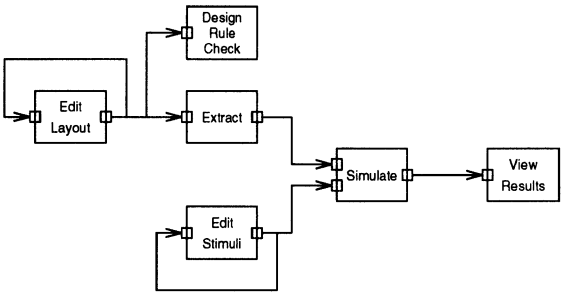

Figure 2: A design flow with data dependencies.

4 Fine-Grain Design Flow Management

For certain design applications, a description of the design environment in terms of tools and their data- and temporal dependencies, is still not detailed enough. For instance, in our example design environment, it may be that within one run of the layout editor multiple design objects can be edited. In this case, we would like 'edit layout' to refer to the editing of one individual design object in the design flow instead of the execution of the layout editor as a whole. Also, we may like to distinguish a flat layout to circuit extraction, which creates a flat circuit from a hierarchical layout, from a hierarchical layout to circuit extraction, which creates a hierarchical circuit from a hierarchical layout. For a hierarchical extraction, we would like to be able to describe its behavior on each level of the design hierarchy. This is visualized in figure 3.

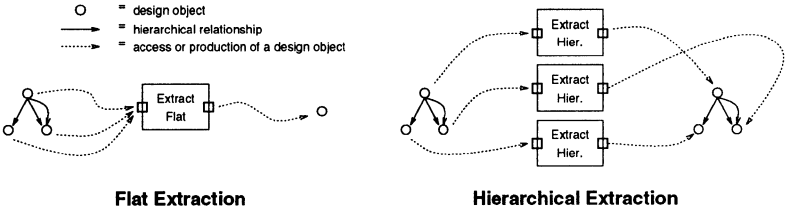

Figure 3: The operation of the extractor differs for a flat extraction and a hierarchical extraction.

In these examples, 'extract flat' and 'extract hier' are different *design functions* of the extractor. We say that a design flow system that allows tools to perform multiple, maybe different, design functions in one run and offers a mechanism to refer to these design functions from the design flow, performs *fine-grain design flow management.*

To support fine-grain design flow management, the design flow system should not only operate on the level of tool invocation and termination, it should be active during tool execution as well. In the flow model it must be possible to refer to the design functions of a tool. For this, we introduce the *activity* construct. (see also [2]). Several different activities may be defined for one design tool. A design tool may perform any of its activities any number of times during a single tool run. Figure 4 shows

a design flow with activities for the example design environment of figure 1. A rectangle denotes an activity of a tool. Note that the input condition for the simulate activity, that *either* a flat *or* a hierarchical extraction has been performed *and* that a stimuli description must have been created, is modeled naturally using data dependencies.

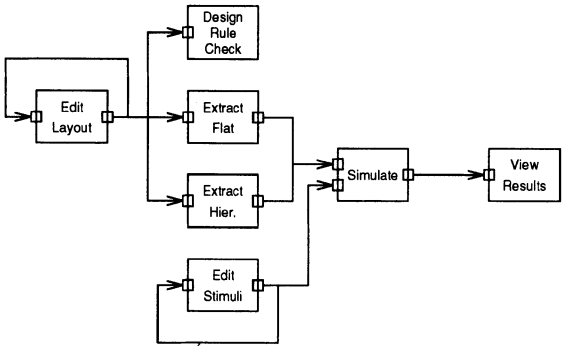

Figure 4: A fine-grain design flow with activities.

Besides a refinement of the design flow model, fine-grain design flow management has a strong impact on its place in the architecture of a CAD framework. To recognize individual activity runs within a tool run, the operation of a design tool needs to be monitored and compared with the configured activities for that tool. Figure 5 shows a general CAD framework architecture. The CAD framework (shaded components) consists of a set of framework services that guard the access to domain specific and domain neutral data, and a set of framework tools. Where a minimal design flow system could be implemented as a framework tool, a design flow system performing fine-grain design flow management must be integrated as one of the kernel framework services. In this way the design flow system can monitor all tools that access their design data via the CAD framework and perform design flow management on a fine-grain level. Support for interactive tools is then a natural fact. A detailed explanation of this material is given in [3] and [4].

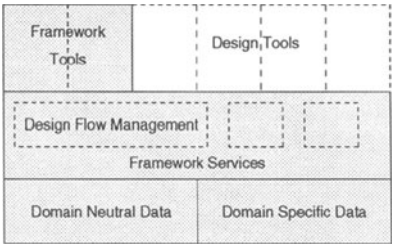

Figure 5: A framework architecture for fine-grain design flow management.

5 Support for Restrictive Design Flows

A common objection against design flow systems is that they would hamper designers in their experimental work. We stress that a powerful design flow system should allow the configuration of both

restrictive as well as *unrestrictive* design flows. In this way a design flow system can be used for any type of design application and the designers' freedom of design is guaranteed. For instance, in the layout verification example a restrictive design philosophy may be required to demand that each layout must have been checked before it may be used in a flat extraction. A design flow for this situation is shown in figure 6. Note that, since no data is exchanged between the design rule check activity and the extractor activity, the channel between the design rule checker and the extractor specifies a temporal dependency. Also note that the design flow still offers alternative design trajectories as designers may still choose between a flat or a hierarchical extraction.

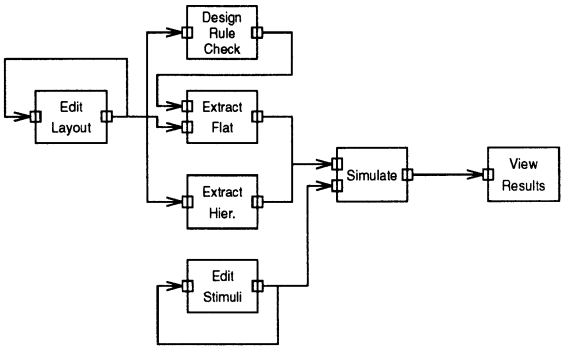

Figure 6: A design flow where each layout must have been checked before it may be used in a flat extraction.

If a design flow system supports restrictive design flows, this has implications on its implementation in the architecture of a CAD framework. In order to guarantee that tools started from outside the design flow user interface adhere to the configured design as well, the design flow system must be implemented in the kernel of the CAD framework. Therefore, the use of restrictive design flows is yet another reason to perform design flow management in the kernel of a CAD framework as described in the previous section (see also [3]).

6 Complex Design Constraints

In real-world design applications, it is quite common to organize large designs hierarchically, so that parts of the design can be described once and instantiated multiple times. This benefits the organization of the design data and facilitates concurrent design, since different designers may operate on different parts of the design hierarchy. Another well-known method for data organization is the use of different representations for a design, called views. For instance, in our example a design may have a layout view and a circuit view. The executability or successful completion of an activity may depend on the hierarchical multi-view organization of a design. For instance, in our layout simulation example we might want to require that a simulation may only be performed if the layout from which the input circuit has been derived and all its children layouts have been checked. We call such a design constraint that takes the hierarchical multi-view design organization into consideration a *complex design constraint.*

To support complex design constraints in a design flow system, its design flow model must offer constructs to refer to the hierarchical multi-view design organization. First of all, we need a way to refer to design objects that must satisfy a certain condition. For this, we introduce a special type of

port, called a *condition port.* Second, we need a way to refer to relationships between design objects. For this, we introduce an *object relationship constraint* (ORC). As an example, a design flow with the complex design constraint that the simulator may only be executed on circuits if the layout from which the input circuit is derived and all the children layouts have been checked, is shown in figure 7. Condition ports are shown as small circles and ORCs as dashed edges connecting ports. ORC *a* is used to refer to the layout from which the circuit has been derived, ORC *b* to refer to all its children of this layout and ORC *c* to refer to the check results of these layouts. The upper condition port is connected to the output port of the design rule check activity, indicating that the check results must have been produced by the design rule checker. For a more detailed examination of complex design constraints, see [5].

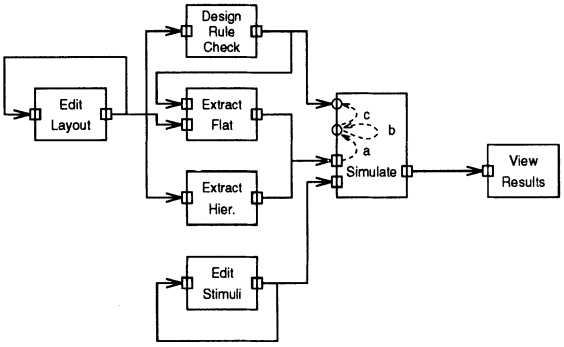

Figure 7: A design flow with a complex design constraint.

7 Flow Coloring

One of the fruits of a design flow system is that the state of the design process can be presented in relation to the configured design flow. Data and (activities of) tools can be visualized in one natural representation of the design environment. Crucial for the success of a design flow system is the method employed for interaction with the end-users. To be effective, it must display interesting parts of the design state and update the presented information when needed. A suitable mechanism for this is *flow coloring.* Flow coloring is the derivation of a *colored flow* that represents part of the design state. A colored flow is a design flow where channels and ports are 'colored' with a design object and activities are 'colored' with a state. This state denotes whether the activity is executable, has been executed or is not executable with respect to the design objects that color its ports. Flow coloring is the process of automatically deriving a colored flow starting from a *user-assigned* design object.

The colored flow to be derived represents the derivation context of the user-assigned design object. A number of *flow coloring operations,* like the assignment or removal of a design object to or from the colored flow, allow the designer to adjust a colored flow to his specific interests. An example colored flow is shown in figure 8. Colored channels are drawn thick and colored ports are drawn solid. Activities are thick if they have been executed, dashed if they are executable and dotted if they are not executable. In this example a layout has been generated with the layout editor, a circuit has been derived from the layout using a hierarchical extraction, and a stimuli description has been created. The simulate activity is not executable with respect to this data, since the layout from which the circuit has been derived has not been checked for design rule errors yet.

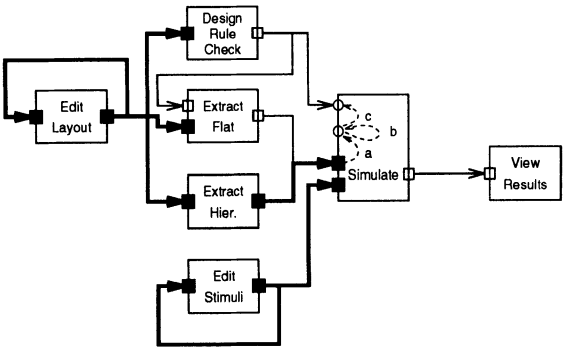

Figure 8: A colored flow, showing part of the design state.

A design flow user interface must be equipped with a facility to update the presented information upon changes in the design state. In a multi-tasking, multi-user design environment a designer may not want to be informed of design operations of its fellow-designers if they are not of his interest. In a user interface based on flow coloring, the information displayed in the colored flow is used to judge the designers interest in a change of the design state. If a change in the design state affects any of the design objects in the colored flow, the colored flow is recalculated and the designer is provided with a new colored flow. For a more detailed examination of flow coloring see [6].

8 Formalizing the Design Flow Model

Data modeling techniques permit a formal definition of the structural semantics of the information stored and maintained by a CAD framework [4]. In this section we use the OTO-D data modeling technique [7] to derive a *data schema* for the design flow model described in the previous sections. This data schema formally specifies the available design flow constructs and their relationships, and can be used to compare different design flow systems as has been done in [8]. The data schema, which is shown in figure 9, is a simplified[1] version of the NELSIS data schema. Boxes represent object types and lines connecting boxes represent attribute relationships.

The data schema specifies that an *activity* belongs to a *tool* and a *port* belongs to an *activity*. The *type* of a port describes whether it is an input or an output port and whether it is a condition port or not. Multiple ports can be *linked* to a *channel* to denote data and temporal dependencies between activities. An *ORC* is a directed binary relationship between a *Source-Port* and a *Destination-Port*. Multiple *activity runs* may be performed within a *tool run* of a *tool*. A *transaction* on a *design object* describes the access or production of a design object in an *activity run* via a *port* of the *activity* of that *activity run*. For each *transaction* holds that the *viewtype* of the *design object* involved in that *transaction* corresponds with the *viewtype* of the *port* via which that *design object* has been accessed or produced.

[1] For simplicity, we do not address the hierarchical organization of design flows in this data schema. The NELSIS data schema does support hierarchical design flows.

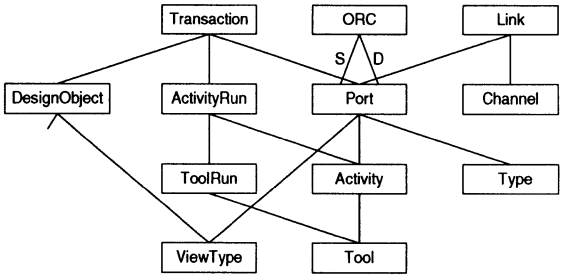

Figure 9: A data schema covering all aspects of design flow management addressed in this paper.

9 Related Work

Most design flow models described in literature do support data dependencies in some way. An exception is the Jessi-Common-Framework [9], where temporal dependencies with *and*, *or* and *xor* constructs are used to build design flows.

Examples of design flow systems that are capable to perform design flow management on a fine-grain level are VOV [10], which *traces* the data access and production of design tools and can therefore visualize changes in the design state while tools are still running and DECOR [11], which allows on-line monitoring of design actions.

Support for restrictive design flows can only be offered in those systems that have a notion of a predefined design flow, like the Jessi-Common-Framework [9], Hercules [12] and the system presented by van den Hamer et al. [13].

To our knowledge, no systems other than NELSIS address complex design constraints.

Flow coloring-alike paradigms have been used in other systems as well. For instance, in Monitor [14] a graph-like representation of the design flow was shown with traffic light colors, van den Hamer et al. [13] presented a user interface that shows data and tool runs in a *flowmap*, and in the Hercules system [12] a visualization of a *task graph* forms the basis of the user interface. Unique to our approach is that, starting from a user-assigned object, we derive a colored flow that represents those parts of the design state in which the designer is particularly interested and that we automatically adjust this colored flow upon changes in the design state.

10 Conclusion and Future Work

For various aspects of design flow management we described how a design flow system can be made suitable for use in a real-world design application. For this, a design flow system should use a design flow model that supports the definition of data dependencies as well as temporal dependencies, that is able to recognize different design functions of a tool performed during a single tool run, and that supports the definition of design constraints that depend on the hierarchical multi-view structure of a design. To perform design flow management on a fine-grain level and to support restrictive design flows, the design flow system should be integrated as one of the kernel framework services of the CAD framework. With respect to the interaction of the design flow system with the end-user, we propose a flow coloring paradigm, that uses the configured design flow to show parts of the design state in an attractive and comprehensible way to the end-user.

The presented design flow concepts have been implemented in the NELSIS CAD Framework to create a design flow system that has been used in a variety of design applications. Among the applications are a circuit design system, an analog system simulation environment, a design system for digital signal processor arrays, a synthesis system, a system for software testing, a simulation environment for medical applications and a generic system simulation environment. Future work will focus on how to deal with changing design environments.

References

[1] S. Kleinfeldt, M. Guiney, J. Miller, and M. Barnes. Design methodology management. *Proceedings of the IEEE*, 82(2):231–250, Feb 1994.

[2] K.O. ten Bosch, P. Bingley, and P. van der Wolf. Design flow management in the nelsis cad framework. In *Proc. 28th ACM/IEEE Design Automation Conference*, pages 711–716, San Francisco, June 1991.

[3] P. Bingley, K.O. ten Bosch, and P. van der Wolf. Incorporating design flow management in a framework based cad system. In *Proc. IEEE/ACM International Conference on CAD - 92*, pages 538–545, Santa Clara, Nov 1992.

[4] Pieter van der Wolf. *CAD Frameworks: Principles and Architecture*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1994. ISBN 0-7923-9501-8.

[5] P. van der Wolf, K.O. ten Bosch, and A. van der Hoeven. An enhanced flow model for constraint handling in hierarchical multi-view design environments. In *Proc. IEEE/ACM International Conference on CAD - 94*, pages ?–?, San Jose, Nov 1994.

[6] K.O. ten Bosch, P. van der Wolf, and P. Bingley. A flow-based user interface for efficient execution of the design cycle. In *Proc. IEEE/ACM International Conference on CAD - 93*, pages 356–363, Santa Clara, Nov 1993.

[7] J.H. ter Bekke. *Semantic Data Modeling*. Prentice Hall, Englewood Cliffs, N.J., 1992. ISBN 0-13-806050-9.

[8] P. van den Hamer, K.O. ten Bosch, P. Bingley, M.A. Treffers, and P. van der Wolf. A comparison of two approaches to design flow management by data schema analysis. Technical report, Philips Research Laboratories & Delft University of Technology, June 1991. JCF Project Deliverable, SP1.

[9] D.C. Liebisch and A. Jain. Jessi-common-framework design management – the means to configuration and execution of the design process. In *Proc. EURO-DAC 92*, pages 552–557, Hamburg, Germany, Sept 1992.

[10] A. Casotto, A.R. Newton, and A. Sangiovanni-Vincentelli. Design management based on design traces. In *Proc. 27th ACM/IEEE Design Automation Conference*, pages 136–141, 1990.

[11] E. Kupitz and J. Tacken. Decor - tightly integrated design control and observation. In *Proc. IEEE/ACM International Conference on CAD - 92*, pages 532–537, Santa Clara, Nov 1992.

[12] P.R. Sutton, J.B. Brockman, and S.W. Director. Design management using dynamically defined flows. In *Proc. 30th ACM/IEEE Design Automation Conference*, pages 648–653, 1993.

[13] P. van den Hamer and M.A. Treffers. A data flow based architecture for cad frameworks. In *Proc. ICCAD - 90*, pages 482–485, 1990.

[14] A. Di Janni. A monitor for complex cad systems. In *Proc. 23rd ACM/IEEE Design Automation Conference*, pages 145–151, 1986.