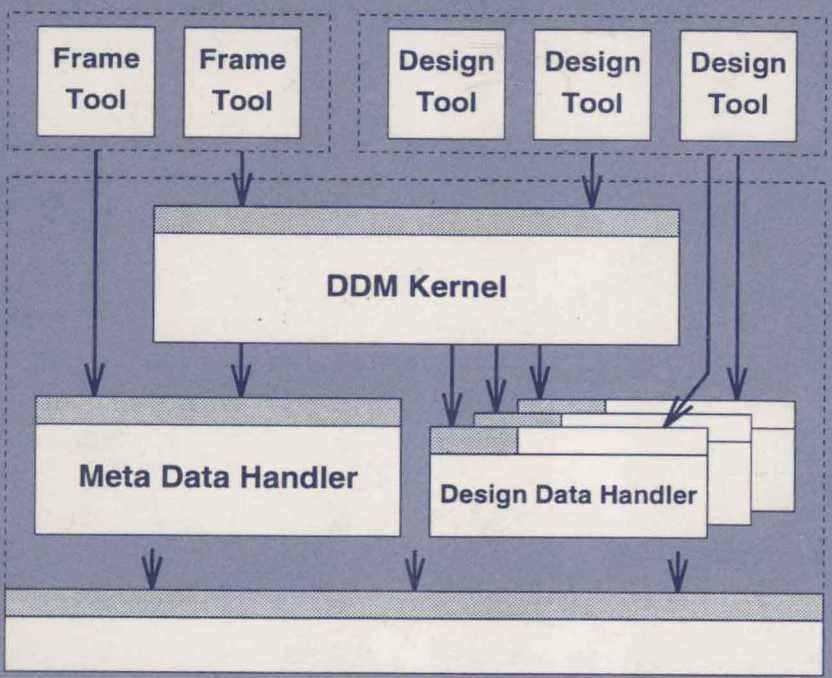
CAD FRAMEWORKS

PRINCIPLES AND ARCHITECTURE



Pieter van der Wolf

Kluwer Academic Publishers

CAD FRAMEWORKS

PRINCIPLES AND ARCHITECTURE

by

Pieter van der Wolf

Department of Electrical Engineering,
Delft University of Technology,
Delft, The Netherlands

KLUWER ACADEMIC PUBLISHERS
BOSTON / DORDRECHT / LONDON

CAD FRAMEWORKS

PRINCIPLES AND ARCHITECTURE

THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

VLSI, COMPUTER ARCHITECTURE AND DIGITAL SIGNAL PROCESSING

Consulting Editor

Jonathan Allen

Other books in the series:

PIPELINED ADAPTIVE DIGITAL FILTERS, Naresh R. Shanbhag, Keshab K. Parhi
ISBN: 0-7923-9463-1

TIMED BOOLEAN FUNCTIONS: A Unified Formalism for Exact Timing Analysis, William K.C. Lam, Robert K. Brayton
ISBN: 0-7923-9454-2

AN ANALOG VLSI SYSTEM FOR STEREOSCIPIC VISION, Misha Mahowald
ISBN: 0-7923-944-5

ANALOG DEVICE-LEVEL LAYOUT AUTOMATION, John M. Cohn, David J. Garrod, Rob A. Rutenbar, L. Richard Carley
ISBN: 0-7923-9431-3

VLSI DESIGN METHODOLOGIES FOR DIGITAL SIGNAL PROCESSING ARCHITECTURES, Magdy A. Bayoumi
ISBN: 0-7923-9428-3

CIRCUIT SYNTHESIS WITH VHDL, Roland Airiau, Jean-Michel Berge, Vincent Olive
ISBN: 0-7923-9429-1

ASYMPTOTIC WAVEFORM EVALUATION, Eli Chiprout, Michel S. Nakhla
ISBN: 0-7923-9413-5

WAVE PIPELINING: THEORY AND CMOS IMPLEMENTATION, C. Thomas Gray, Wentai Liu, Ralph K. Cavin, III
ISBN: 0-7923-9398-8

CONNECTIONIST SPEECH RECOGNITION: A Hybrid Appoach, H. Bourlard, N. Morgan
ISBN: 0-7923-9396-1

BiCMOS TECHNOLOGY AND APPLICATIONS, SECOND EDITION, A.R. Alvarez
ISBN: 0-7923-9384-8

TECHNOLOGY CAD-COMPUTER SIMULATION OF IC PROCESSES AND DEVICES, R. Dutton, Z. Yu
ISBN: 0-7923-9379

VHDL '92, THE NEW FEATURES OF THE VHDL HARDWARE DESCRIPTION LANGUAGE, J. Bergé, A. Fonkoua, S. Maginot, J. Rouillard
ISBN: 0-7923-9356-2

APPLICATION DRIVEN SYNTHESIS, F. Catthoor, L. Svenson
ISBN:0-7923-9355-4

ALGORITHMS FOR SYNTHESIS AND TESTING OF ASYNCHRONOUS CIRCUITS, L. Lavagno, A. Sangiovanni-Vincentelli
ISBN: 0-7923-9364-3

HOT-CARRIER RELIABILITY OF MOS VLSI CIRCUITS, Y. Leblebici, S. Kang
ISBN: 0-7923-9352-X

MOTION ANALYSIS AND IMAGE SEQUENCE PROCESSING, M. I. Sezan, R. Lagendijk
ISBN: 0-7923-9329-5

HIGH-LEVEL SYNTHESIS FOR REAL-TIME DIGITAL SIGNAL PROCESSING: The Cathedral-II Silicon Compiler, J. Vanhoof, K. van Rompaey, I. Bolsens, G. Gossens, H. DeMan
ISBN: 0-7923-9313-9

CONTENTS

1

INTRODUCTION

1.1 THE NEED FOR CAD FRAMEWORKS

More and more these days, the bottleneck in the development of advanced electronic products is *design*. More and more designs of increasing complexity have to be done faster and faster to bring more advanced end-products to the market in time [Man92, dG93]. The key to gains in design productivity is Electronic Design Automation (EDA). The optimistic EDA picture is that a large number of *tools* have become widely available. For specific design tasks these tools help the designer to master the complexity and perform these tasks efficiently. Together with the interactiveness provided by modern graphical workstations this has yielded significant productivity improvements for parts of the design process.

Due to this focus on *tool automation*, design systems have become large tool-boxes offering the designer a great variety of loosely coupled tools to perform the many design tasks. The realistic EDA picture, however, is that these tools support only individual design tasks, leaving the designer with the problem of handling the multitude of tools and of successfully moving his design data from one tool to the other. Moreover, the number of tools to be operated is growing continuously, as are the amounts of design data to be handled. What is lacking is *integration* and overall support for managing the *design process*.

People have recognized these problems and have realized that attention has to be paid to the overall efficiency of the design process in order to continue achieving gains in productivity. As a consequence, one of the buzzwords in the EDA arena is *CAD framework*. CAD stands for Computer-Aided Design.

We adopt the following definition of CAD framework, as originally given by the CAD Framework Initiative (CFI), the international consortium developing framework standards [CFI90b]:

Definition 1.1 (CFI) *A* CAD framework *is a software infrastructure that provides a common operating environment for CAD tools.*

CAD frameworks play a role in *building* as well as in *operating* integrated design environments. First, a CAD framework has to provide facilities for conveniently integrating multiple CAD tools into a coherent design environment. It is a basis for *tool integration*. In this respect there is a direct analogy with the role an operating system plays in the development of general-purpose software applications.

Second, a CAD framework can support the end-user in conveniently operating the design environment. Being the infrastructure that binds the tools together, the framework is the proper place to incorporate facilities for organizing the design information and managing the design process. In an integrated design environment, these facilities can support the end-user in keeping track of the state of his design and in applying tools effectively. This will help him to master the complexity of the design and the design process. More complex circuits, satisfying more stringent performance and quality standards, can then be done faster under the increasing time to market pressures. Thus, a CAD framework is to become the *electronic assistant of the designer* for organizing the design information and managing the design process.

From the above description we identify two categories of framework users: *developers* (e.g. CAD tool developers, CAD tool integrators) and *end-users* (e.g. design engineers, administrators, project managers). To get a more concrete idea of what the introduction of CAD frameworks may yield in practice for the end-user, consider the following support that a CAD framework is expected to give:

- Help the design engineer to maintain an overview of his design descriptions: which components are available, what is their status and history, how are the components related.
- Tell the design engineer which tools are available for which design tasks and give information on their usage. Moreover, a framework may take

care that tools are applied in the proper sequence such that a pre-defined design methodology is adhered to.

- Make design projects manageable by performing book-keeping on the status of achievement for consultation by the project manager.
- Allow teams of design engineers to cooperate effectively on a design project. A related buzzword here is *concurrent engineering*, which is "the art of decomposing a complex serial task into smaller, relatively independent tasks that can be executed in parallel" [CFI90a].

Effective framework technology providing such advanced facilities as exemplified above can thus help to:

- *Cut down design time.* Recent studies show that for every six months a project is late, the potential profit is reduced by a third [DEC92].
- *Make the design process less error prone.* According to recent studies, design errors account for an average of 20 percent of product costs [DEC92].
- *Master the increasing complexity* of the design and the design process.
- *Increase performance and quality* of electronic products.

In addition, a CAD framework provides *environmental stability* as it offers a standard operating environment to an ever evolving set of CAD tools, in which many services have yet been captured. It promotes *modularization* of CAD systems, as these are to be constructed as cooperating tool components on top of a CAD framework, rather than being implemented as monolithic super-tools. In fact, CAD tools may become simpler as the generic services of CAD frameworks become more powerful. Further, we think that CAD frameworks do not simply add to what a designer has to learn; Increased uniformity and user-friendliness may actually make CAD systems easier to operate.

1.2 THE SEARCH FOR CAD FRAMEWORKS

There appears to be world-wide consensus on the point that there is an urgent need for CAD framework technology in order to build effective integrated design environments that help improve design productivity. Also, the EDA community appears to agree on the major functional requirements for a CAD framework. However, there is no common idea how to go about developing and implementing one. As is said in [Val90], "The tremendous amount of importance placed on frameworks, as they directly relate to improving productivity, combined with the lack of agreement on their exact nature has shrouded the subject in mystery and uncertainty". The current situation is that effective framework based design systems satisfying the major framework requirements have not yet reached the designer's workbench [vdH91, Mal92]. In fact, major EDA vendors are investing heavily but have problems in providing the required functionality while making the system efficient [BHNS92].

The CAD Framework Initiative (CFI), the international consortium developing framework standards, is trying to alleviate the mystery surrounding frameworks. The mission of CFI is: "To develop industry acceptable guidelines for design automation frameworks which will enable the coexistence and cooperation of a variety of tools" [CFI90b]. Having standard interfaces to CAD framework components will substantially reduce the cost of combining multi-sourced CAD tools into an effective design environment. But, progress is slow and consensus on specifications, detailed requirements and implementations appears difficult to reach [vdH91].

The major European framework effort is the Jessi-Common-Frame (JCF) project, which is aimed at building a CAD framework. Also in this project the different development partners, though having extensive framework experience, appeared to have quite different views on how to build a framework. Extensive exertions were required in order to identify critical differences in the respective approaches and to obtain a common understanding on a global framework architecture.

Now, what makes it so difficult to develop, implement, and even discuss CAD frameworks? Obviously CAD frameworks are complex systems; this is the nature of the subject. First of all frameworks have to satisfy many functional and operational requirements posed by different categories of users. A great variety of services has to be provided to tool integrators, design engineers, administrators, etc. A CAD framework is not a piece of software

performing a specific design task in a specific way with measurable results. In many respects it must be generic, customizable, and open, in order not to have built-in restrictions for a particular usage of the system. For example, it should not make restrictive assumptions on the kinds of tools to be supported. The generic nature required for many framework functions adds to the complexity. Further, the framework is a multi-user system, which must provide services to many concurrent users in a distributed hardware environment, responding well under all kinds of circumstances. This significantly adds to the complexity of the framework problem.

In addition to the complex nature of CAD frameworks, discussions on framework architectures are hampered by the lack of agreement on a common formal "language" or "model" to represent and discuss framework architectures. Each active member of the framework community appears to have his/her own way of (informally) representing certain aspects of a framework architecture. As a result, ideas on framework architectures do not get communicated well. It is hard for framework developers to disengage from their own background and learn the essence of the work of other people. A confusion of tongues in framework discussions is often the result.

In the following chapters we respond to the situation sketched above by systematically deriving the architecture of a CAD framework. This architecture will be described by a global framework model and three more specific framework views. We will define key principles to direct design choices and will use well-defined primitives to describe the framework views.

Two key features of the CAD framework architecture will be *openness* and *efficiency*. Openness is the ability of the CAD framework to easily incorporate new tools, new types of data, and new design methodologies. It must be flexible, configurable, and largely independent of a specific application domain. Efficiency implies that overall efficiency of the design process is optimized. This relates to run-time performance of individual framework services as well as to framework functions that help the end-user to work more effectively.

Before we start the architecture definition process, we investigate in the next chapter the state of the art in the area of CAD frameworks, and their architectures in particular. We also present the principal requirements a CAD framework has to satisfy.

2

STATE OF THE ART AND REQUIREMENTS

2.1 THE EVOLUTION OF CAD FRAMEWORKS

2.1.1 File-and-Translator Based Systems

From the literature it appears that over the last ten years people have gradually discovered the topic of CAD frameworks, and have gradually allocated more functional requirements to this topic. The state of the art in CAD frameworks is the result of increasing awareness and evolutionary developments rather than specific scientific breakthroughs. We therefore start our assessment of the state of the art from an historical perspective.

The history of CAD frameworks starts in the early eighties when people realized that with the growing number of CAD tools data transfer between tools became an ever growing pain. At that time tools typically used proprietary and tool-dependent ascii or binary formats to represent the design data. Communication between design tools was possible only if a translator for the respective formats was available [Kat83, Kal85, Kat85b, GE87, HNSB90]. This situation is illustrated in Figure 2.1. Bringing the growing number of tools together into an integrated design environment involved writing a large number of translators. The emergence of de-facto standard formats, allowing the number of translators to be reduced, eased this pain a bit.

These tool integration efforts had made people realize that effective EDA solutions not only had to provide the individual tools but also the integration facilities, or 'glue', to support the communication between tools. The integration facilities in the file-and-translator based systems consisted of translators

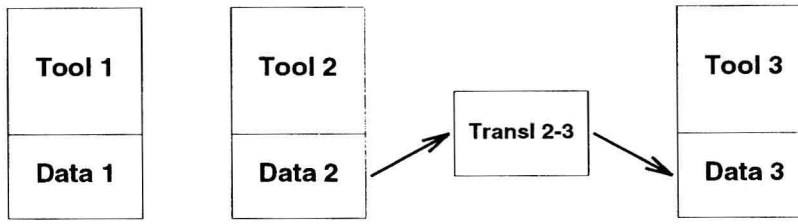


Figure 2.1 Tools having proprietary data representations. Translation across data representations is required for sharing data among tools.

and a whole collection of ad-hoc utilities to make life easier for the end-user. These simple integration facilities can be termed the first primitive CAD frameworks, and a new EDA topic had been born.

2.1.2 First Role: Design Database

Historically, the first role allotted to CAD frameworks is that of *common data repository* [NPSVtS81, Kat82, Goe85, Tur85, CFHL86]. Data which is common to a number of CAD tools is stored only once in the repository, from where it can be used as input for all tools. For tools that have not been written to operate directly on the common data repository, reformatting may be performed on input and output. The common repository has come to be called 'integrated design database', or just '*design database*'. It is the key to *tool integration*, and in particular to *tool interoperability*: the ability of CAD tools to communicate and share data. See Figure 2.2.

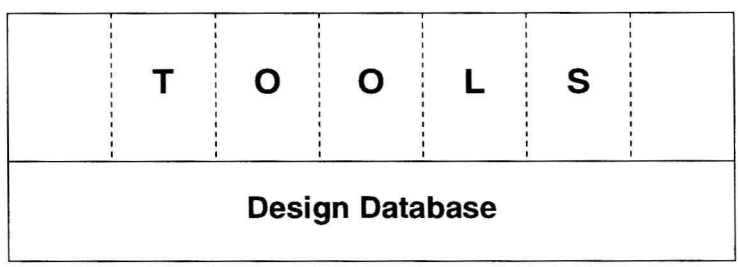


Figure 2.2 Tools integrated on top of a design database.

Advantages of this approach are increased consistency and increased efficiency. For the end-user there is increased convenience as his design data is stored in a structured way in a single place. Additional database utilities support the end-user in managing his design data.

File Based Systems

In many CAD systems the design database is a file based system, implemented as a small layer on top of the host operating system [NPSVtS81, Kat83, Kat85b, May87]. This layer augments the operating system with specific support for managing the design files. Design files are stored in a structured file organization, for example, employing a hierarchical file system and pre-defined naming conventions. Access methods are provided for storage and retrieval of design data. These access methods may offer atomic update capabilities.

Examples of the file based approach are the Nelsis CAD Framework, Release 2 and Release 3 systems [Dew86]. Several more advanced CAD frameworks still employ file based systems for persistent design data storage. Examples are Design Framework II from Cadence Design Systems, Falcon from Mentor Graphics, PowerFrame from Digital Equipment Corporation, and the Nelsis CAD Framework, Release 4.

Use of Conventional DBMSs

Another approach has been to employ conventional (record-oriented) database management systems (DBMSs) to fulfil the role of design database. These conventional DBMSs are typically used as storage and transaction processing components in business applications. The interest in DBMSs was driven mainly by the apparent 'keyword-level' match between DBMS functions and facilities and the emerging requirements for design databases. A DBMS provides mechanisms for the reliable storage of data, including recovery facilities, it protects data from unauthorized access, and it provides concurrency control and integrity maintenance. The notion of *transaction* supported by these systems also seems useful: a sequence of database operations that are either executed completely or not at all [Dat86].

Quite a number of attempts to employ conventional database techniques have been published in the literature [Hay81, RBJ81, Zin81, Kat82, WBS82, Hay83, HNCL84, Har84]. However, none of them reported a straightforward

applicability of off-the-shelf DBMSs. Most of the conventional DBMSs have been targeted for business applications and do not specifically address the problems encountered in a design environment. Critical differences occur in characteristics of data to be handled (e.g. granularity and inter-relatedness) and access characteristics (e.g. frequency, granularity and duration of transactions) [Sid80, LP83, Buc84, HY85, SA86]. There is a mismatch between the facilities provided by the DBMSs and the requirements posed by engineering applications. Due to this mismatch, efficiency is hard to obtain, if at all, given the internal mechanisms (for concurrency control and recovery, for example), which have been geared towards the characteristics of business applications. We are currently not aware of any successful application of a classical business DBMS for the purpose of a CAD framework design database, and consider this to be a dead end.

Successes have been reported in the application of conventional DBMSs for meta data handling only, that is, to hold administrative data including references to the actual design data stored in files. An example of the use of a conventional DBMS for meta data handling is [BD91].

Object-Oriented DBMSs

The database community has recognized the fundamental nature of the requirements posed by engineering applications and other new database applications such as multimedia databases and knowledge bases. Work was started on next-generation DBMSs targeted at these applications: *object-oriented DBMSs*. In contrast to conventional DBMSs, object-oriented DBMSs offer far more flexibility for handling highly interrelated data of different granularities on which different types of access are performed. For engineering applications, the DBMS must be capable of handling many different data types and large numbers of instances of each type. Moreover, flexibility must be provided for modifying or extending the conceptual schemas to match the different views of data operated on by different application programs [SA86, BP86].

Databases typically found in Artificial Intelligence (AI) systems tend to provide flexibility for many different data types, which may also be defined dynamically. An example of such a system is the Pearl AI Package (Package for Efficient Access to Representations in Lisp) [DFW82]. These systems, however, have not been geared towards the large numbers of instances of each type, as found in engineering applications [SA86]. Moreover, they typically are single-user systems. They may be used for purposes of prototyping,

but are not suited as production systems.

The data models provided by powerful object-oriented DBMSs permit arbitrary types of design information to be represented and accessed conveniently. Typically the design information is modeled as a web of highly *interrelated objects* [HMSN86, FFHe91, Obj91, HJR92]. Granularity of the objects may vary significantly; from a simple integer attribute value to a complete design file. Typically, a *traversal type of access* is provided to the application programs built on top of the DBMS. This permits them to navigate through the object web, to retrieve objects or to add new ones to the web. A key in achieving efficiency for EDA applications is maintaining locality of larger aggregates of data that are typically accessed as a (compound) entity. For this purpose object-oriented DBMSs targeted at these applications provide such features as *clustering* or *complex objects* [LP83, Buc84, HS87, FFHe91, Obj91, HPC93].

A number of object-oriented DBMSs have been realized and have become the base components of some of today's frameworks. For example, the Objectivity/DB [Obj91] has been used for meta data management in ValidFrame [Val90] and the Cadlab OMS (Object Management System) [FFHe91] is used in the Jessi-Common-Framework [JCF91b]. Cadence Design Systems has announced an agreement with Object Design, Inc. to use the ObjectStore DBMS in its next-generation EDA software. These developments signal a move by the EDA software industry away from proprietary design databases to the use of standard commercially available DBMSs.

2.1.3 Second Role: Design Data Manager

The second major role allotted to CAD frameworks is that of *design data manager*. A design database, in the sense of common data repository, provides a facility for storing the design data, but provides no support for *managing* the data. Randy Katz from U.C. Berkeley was one of the first to realize that 'brains' could be added to the design database 'muscle', to actually help the designer in organizing his design information [Kat83]. A design data management system uses knowledge of the structure and status of design information to provide management support and enforce constraints on the design process.

Clearly, system integration can be, and should be, much more than the definition of common formats for the purpose of tool communication. With