On the Architecture of a CAD Framework: The NELSIS Approach

P. van der Wolf, P. Bingley and P. Dewilde
Delft University of Technology
Department of Electrical Engineering
Mekelweg 4, 2628 CD Delft
The Netherlands

ABSTRACT

In this paper we present the principles of the architecture of a CAD framework that puts a substantial added-value under the fingertips of the designer by organizing the design information and keeping track of the design evolution, and in addition permits integration of tools of different origin and achieves run-time efficiency.

We provide a rationale for partitioning the design information into raw data and meta data, then we model the meta data and define the architecture of the framework. Key features of this framework are its high level services for tool integration and extensive browsing facilities for the designer. The principles presented have been implemented in the Nelsis IC Design System in which they have proven their usefulness and efficiency.

1. Introduction

Traditionally the term CAD framework has been used to denote the set of common software facilities that are required to build an integrated CAD environment. The crucial role of a CAD framework in the area of tool integration has often been emphasized [1,2,3]. However, more is to be earned than just having a common repository for design data. By organizing the design information in a way that relates to the designer's perception and by keeping track of the evolution of the design as performed via tool operations, the framework becomes a facility that helps the designer to control his design activities. The net result is a gain in design productivity, because the designer can concentrate on his main task, electronics design.

The problem we face in this paper is the definition of the architecture of a CAD framework that provides this "design orientedness" while satisfying other key requirements for CAD frameworks, such as:

- Openness of the framework. It must be an open framework for tool integration that does not restrict the functionality of the design system or forces a design methodology on its users. Foreign tools must be allowed to operate within the system as consistent parts of the environment. In particular it has been our challenge to provide high level design services, such as support for hierarchy and equivalence, versioning, concurrency control, state management, etc., while retaining openness.
- Offer a high degree of configurability of the environment. In particular when aiming at high level design services, there are many aspects that must be configurable from outside the system rather than being hardwired deep inside.
- Simplicity of concepts and their implementation. The objective is to have a flexible lightweight CAD framework that can easily be understood, implemented and used.
- Efficiency. Introduction of the framework facilities should cause no significant run-time performance degradation for the tools interacting with the framework.

The red wire in our solution to this problem is that we focus on intelligent management of so-called meta data on top of the actual design descriptions. In the next section we distinguish meta data from raw design data and show that proper meta data management can enhance design system functionality in various respects. In section 3 we apply semantic data modeling techniques to describe the structure and properties of the meta data. Section 4 illustrates how high level framework services can be built to actually manage the meta data. Framework browsers, which exploit the meta data to inform the designer about the structure and status of his design, are introduced in section 5. In section 6 we present an overview of the architecture of the Nelsis IC Design System in which the concepts presented have been realized, and give some details on its implementation.

2. Meta Data versus Raw Design Data

The central theme in our approach is that the framework administers information about the design and the design activities rather than operating at the level of the detailed design descriptions. This high level information about the design and the design activities is what we call meta data. This involves primarily information on the presence of cells, or design objects, their relationships, their version history, and the operations that have been performed on them.

The framework does not make any assumptions about the actual detailed design descriptions contained in the design objects (e.g. their granularity, or the data formats). The design objects may, for instance, contain detailed mask descriptions, netlists, or behavioral descriptions. This raw design data is operated upon by the design tools. This approach is in line with work presented by Katz [4] and Batory [5].

The meta data gets collected (by the framework) while the tools communicate with the framework to obtain access to the actual design descriptions. Compare this procedure to a public library, where the library personnel (framework) maintains card-trays (meta data) with information about the books (design objects) that are borrowed by the customers (tools).

A well structured meta data pool can be used to enhance system functionality in various respects:
• The framework itself exploits the meta data to perform various services (see section 4). Since the design object is the unit of access for operation by the design tools, such issues as versioning, concurrency control and state management may be handled by the framework at the level of the design object. To provide these services the meta data is used as the scratch-pad of the framework itself.

• Classical tools typically operate upon raw design data as present in the design objects, paying no attention to additional information that may be available. However, having the meta data at our disposal, smarter tools may be constructed. An example is a hierarchical verification program that exploits verification statuses from the meta data to selectively process only the design objects in the design hierarchy that have been changed since the last verification run.

• Special framework tools may be constructed which permit the designer to interact with the framework and its meta data. The designer manipulates design objects via his design tools to edit, simulate or verify his design descriptions. Access to an administration about these design objects is crucial to keep track of the design process. Particularly useful may be the framework browsers which make the high level information about the structure and status of the design available to the designer: designer orientedness (see section 5).

The distinction we make between meta data and raw design data helps us to distinguish between (global) system aspects and (local) tool aspects of the design environment, both for data to be handled and functions to be performed. The global system aspects are implemented as part of the framework, to be offered as system-wide facilities for data organization and design process control. Thus, the framework is based on invariants that can be recognized in the design environment, rather than the features of a particular tool set or design representation.

This is an important contribution to the openness of the environment.

The distinction between meta data and raw design data is also key in achieving run-time efficiency. As will be described in section 6 an efficient storage module can be implemented tuned to the characteristics of the meta data and meta data access.

3. Meta Data Modeling

To describe which meta data has to be maintained by the framework and what its structure and properties are we apply data modeling techniques for the purpose of information analysis. In contrast to work presented by Katz [4] and Batory [5] we formalize the semantics of the meta data with well-defined abstraction primitives to yield an accurate definition of the information that is to be maintained. The data modeling exercise also provides a basis for an implementation of the framework, since the data model may be implemented yielding a configurable storage module for the meta data.

We selected the OTO-D data model [6] to pursue our modeling activities. OTO-D stands for Object Type Oriented Data model. It is a powerful semantic data model, with which object types and their relationships can be modeled, including various integrity constraints. The result of such a modeling activity is a data schema, which describes the (structural) organization of the application environment. The OTO-D data model was chosen specifically because it offers a high level of semantic expressiveness, well-defined modeling constructs, an attractive high level query language, and has a clear way of visualizing the data schemas one defines. OTO-D data schemas have an unambiguous interpretation and reflect important integrity constraints among the object types defined.

We have applied the OTO-D modeling techniques to the Nelsis IC Design Environment to formally describe such aspects as design object (cell), hierarchy, equivalence, versioning, tool transactions, etc. The data schema that resulted from this modeling activity represents the organizing principles that are used by the Nelsis framework to organize the design data, including the various relationships and integrity constraints. This data schema is depicted in figure 1. We will shortly explain the data schema; for an extensive description we refer to [7] and [8].

![Figure 1. Data schema as constructed to organize the meta design data](image)

With the OTO-D data model an object type is defined in terms of its attributes. When drawing a diagram for this an attribute relationship is represented by a line that goes from the bottom of the composed type to the top of the attribute type. An attribute type may be a base type or a composed object type which is then again defined in terms of its attributes. The referential integrity constraints along the attribute relationships always have to be satisfied: for an instance of a composed type to exist all referenced instances of the attribute types must exist. The basic entity in the Nelsis Data Management System (DMS) is the Design Object. This is the central object type in the data schema. To a design object corresponds a design description (e.g. a layout or a circuit diagram) that may be accessed by the tools. That is, the design object is the unit of access to the design data for the design tools [7, 8, 9].

A module generally consists of several design objects as its versions. In the data schema a Module is described as something which has a Name, a ViewType, etc. These are the attributes of Module. A design object belongs to a module as one of its versions. Hence, DesignObject has the attributes Module, Vnumber, Vstatus, etc, to administer the module the design object belongs to, together with version-specific information such as the version number and version status.
Tools perform transactions on design objects, and these transactions are administered by the DMS in the meta design data. For this purpose Transaction is modeled as the DesignObject on which the transaction operated, the Tool that operated it, and some additional information such as Date and Designer.

Design objects may be related by either hierarchical relationships or equivalence relationships. An Hierarchical Relationship relates a Father-DesignObject to a Son-DesignObject, together with an InstanceName and some construction information. Similarly, an Equivalence Relationship relates an Original-DesignObject to a Derived-DesignObject, together with the Tool that derived the relationship. The two types of relationships have been modeled in such a way that the OTO-D referential integrity constraints permit relationships between existing objects only.

The object types ImportedDO and Export model the library mechanism, where design objects from one project can be imported into another project. Such an import is administered in the importing project as an ImportedDesignObject, while in the exporting project it is administered as an Export.

Besides being a useful tool for information analysis, the OTO-D data model also provides a basis for an implementation of the framework. A data model is a set of concepts for data definition (schema definition) and data manipulation (query language). Consequently, an implementation of a data model may provide a generic facility for maintaining and retrieving the meta data of the framework. Since OTO-D permits many properties of the meta data to be modeled and offers a high level query language, implementation yields a high level generic storage module for our meta data.

4. Framework Services and Tool Integration

Once the generic meta data facility is in place, the DMS layer can be built on top of it to provide high level framework services. It are these services that actually provide the IC design specific support for management of the design data and design activities. Examples of framework services are:

- Support for design transactions.
  Design transactions on design objects can be supported by adopting the transaction model described by Lorie [10]. A design transaction is initiated by a CheckOut request and terminated by a CheckIn request. The meta data module can be used to administer the in-progress transactions and maintain a transaction history of completed transactions for each design object. Additional facilities include concurrency control and recovery at the level of these design transactions. For example, a CheckOut may be rejected if an edit transaction is already in progress on the requested object.

- Version mechanism.
  A version mechanism permits multiple versions to co-exist for a single module. It may distinguish the individual versions by means of version numbers or statuses. The mechanism may, for instance, support the notion of current version and decide to either overwrite an existing version or create a new version upon an edit transaction. Administering the individual versions can be done with the meta data module, along with such attributes as version-number or version-status, as was shown in figure 1.

- Support for hierarchy and equivalence.
  If the hierarchical and equivalence relationships are administered in the meta data module, the framework can invalidate relationships when necessary and propagate updates across these relationships to re-validate.

- Design flow management.
  To automate the design process even further, a so called design flow manager can be implemented. Such a design manager typically is configured with knowledge about the actual tools available and their dependencies, and helps the designer to adhere to a particular design flow and thereby correctly traversing his verification trajectory. A design manager may also advise the designer about which tool to run on which object. Ultimately tools may even be invoked by the design manager to automatically perform certain verification steps or derive alternative design descriptions.

The meta data facility provides the "fact base" for this design manager. It describes the actual state of the design at the level of design objects, relationships between them, and transactions performed on them (transaction history).

These framework services rely on the presence of particular object types and attributes in the data schema, in order to administer or retrieve the necessary information. Since the meta data module can be configured with arbitrary OTO-D data schemas, it can fully adhere to the storage facilities required by the framework services. The framework services actually maintain the meta data administration, just as the library personal maintains card-trays about the books. As a result, both designers and tools end up with design information organized according to the data schema.

A particular challenge in providing these services in a CAD framework is to retain openness. The services have to be sufficiently general so that the framework does not restrict the functionality of the design system or the design methodology on its users. Also, the tools should not become too dependent upon the actual services provided by a framework implementation. The solution is to have a standardized interface between the tools and the framework which hides many of the specific aspects of the framework services. For instance, concurrency control or design management functions can be performed in a fully transparent way by the framework underneath such an interface.

The standardized interface used in the Nelsis IC Design System is called the DMI: Data Management Interface [11]. It is a procedural interface that can be used to obtain access to the design objects. The DMI is structured in such a way that it decouples the development of the tools and the framework. Prominent functions are CheckOut and CheckIn, which can be used by the tools to initiate and terminate design transactions. When called by a tool, which requests access to design data, the DMI functions activate the various framework services to perform the necessary operations before returning control to the tool. See [11] for a more elaborate discussion on the DMI-principles.

The most efficient interaction between a tool and the framework occurs if the DMI functions are called directly from within the tool object module. However, this situation may not always be possible, for instance, if source code is not available. In these cases the DMI allows the actual tool to be encapsulated in a so called wrap-tool, which performs the DMI-calls before and
after the actual tool run, to prepare the data for the tool and return it afterwards. Of course this is a less efficient technique for tool integration. It must be noted that encapsulation may be very problematic if the required design data resources cannot be predicted beforehand, as may be the case with complex interactive tools.

5. Framework Browsers

Now that the meta data gets collected in the meta data facility by the framework services we may enter the domain of a whole new class of tools: the framework browsers. These framework browsers bring the potential power of the framework right under the fingertips of the designer. Having all the meta data at their disposal, they may answer questions about global characteristics of the design, such as: Which versions of this cell exist? If I change this cell, which other cells will be affected? Has this layout been extracted? etc. By informing the designer about the structure and status of his design, decisions on which tool to apply next on which part of the design are facilitated.

The framework browser in the Nelsis IC Design System is a tool called the Design System User Interface (DSUI). The DSUI is an interactive graphical meta data browser built on top of the framework kernel. Rather than building only IC design specific browsers, based on IC design specific aspects modeled in our data schema, we decided to offer both a generic OTO-D query interface and simple-to-use IC design specific browsers as complementary facilities:

1. The generic OTO-D query interface can be used to retrieve information with the OTO-D query language for arbitrary data schemas. It exploits the attractive diagrammatic notation of the OTO-D data model to graphically represent the current data schema.

The OTO-D query language closely matches this notation, thus permitting extensive graphical support for query composition. By making the data schema an integral part of the browser the designer is relieved from the burden of having to remember the data schema and the syntax of the query language; queries can be formulated by simply selecting the object types and attributes of interest (see figure 2). The browser highlights selected and selectable object types and attribute relationships, and also displays the textual representation of the query.

2. Simple to use IC design specific browsers retrieve information from specific object types in the data schema and display this information in an attractive graphical manner. A designer can quickly obtain an overview of his design by looking at a few pictures showing information on his design objects and their relationships. Examples are a hierarchy browser and a version browser (shown in figure 3), which have been built in a modular way on top of the OTO-D meta data facility. The DSUI may retrieve tool transactions from the framework kernel, to display design histories for the design objects.

6. Architecture of the Nelsis IC Design System

The concepts described so far have been implemented in the Nelsis IC Design System. We present an overview of its architecture to illustrate how a designer-oriented framework, based on the meta data management techniques described above, can actually be realized. Additional implementation details show that efficiency can be obtained. The figure below presents the architecture of the Nelsis IC Design System.

The kernel of the Nelsis IC Design System Framework is a configurable meta data storage module, based on the semantic data model OTO-D. It employs an efficient table handler module based on shared memory techniques. A query interpreter has been implemented to enable high-level access to the meta data. An OTO-D data schema can be fed to the framework kernel which can then respond to requests formulated with the OTO-D query language to store and retrieve information that is organized according to this data schema.

The DMS-layer, containing the actual framework services, has been built on top of the query module. At the top side the DMS
interfaces to the, possibly encapsulated, design tools via the DMI, which hides many DMS-specific aspects from the tools. The DMS, with its services, controls access to the design objects and administers meta data by performing OTO-D queries. Once access to a particular design object has been granted, a tool may efficiently access the actual design data.

A special class of tools are the (meta data) browse tools, like the DSUI presented in section 5. These tools focus on access to the meta data to present high level information about the structure and status of the design to the designer. This information is retrieved by performing OTO-D queries on the meta data module. These tools may also incorporate utilities for meta data modification, for example to affect the version-status of a design object.

Although a framework could be built with a single (possibly object oriented) storage facility for both design data and meta data, we have chosen differently. Our solution is to have a basic storage facility for the raw design data, and a high level configurable module for the meta data which satisfies the characteristics of meta data access. These characteristics differ significantly from tool access to the raw design data, not only in the granularity and overall size of the data, but also in frequency of access, search procedures, concurrency requirements, data sharing, efficiency claims, etc. The implemented module handles the meta data in core, providing services for concurrency control, data sharing, atomic transactions, recovery and physical distribution. The result is a lightweight system that was relatively easy to implement and that efficiently handles meta data access.

At the side of the raw design data the openness of the system is not hampered by enforcing a particular storage regime. Foreign tools may rely on their private design data storage policy, while operating (via encapsulation techniques) as a component of the system. Tool-kit like facilities may be offered for optional use by tool builders. For instance, a tool-kit for storage and retrieval of in core data structures (persistent object facility), or a tool-kit for direct tool-tool communication may be provided. The important point is that tool builders and tool integrators must be free to store data as they like with the facilities they like. It is not our choice to have tools integrated, it is the choice of the tool builders and tool integrators to do so. It is our task to make it possible.

7. Conclusion

The presented CAD framework focuses on handling meta data on top of the actual design descriptions. While the design tools interact with the framework to store and retrieve design descriptions, the framework provides various services and administers information about the design and the design activities. As a result the framework turns out to be more than just an integration frame; it actually provides substantial additional functionality in terms of framework services and browsing capabilities. Moreover it provides a sound basis for the incorporation of design management facilities to automate the design process even further.

References


