25

Framework Services for Design Data and Design Flow Management

Rene van Leuken and Alfred van der Hoeven
Delft University of Technology / DIMES-DDTC
Department of Electrical Engineering
Mekelweg 4, 2628 CD Delft, The Netherlands
rene@cas.et.tudelft.nl, alfred@cas.et.tudelft.nl
http://www.ddtc.dimes.tudelft.nl

Abstract

In this paper we present an overview of principles and mechanisms which support management in their task to control and track the engineers work and which guide engineers through the creative design stage. In today's engineering environments engineers use design tools to create, modify and verify their designs. During this work the primary focus of an engineer is to successfully perform design tasks. Design descriptions and the tools that operate on them become more complex every day. Today design descriptions are usually constructed hierarchically, consist of multiple levels of abstraction (views) and have a status. The central theme in our approach is that the framework maintains information *about* the design descriptions and the design activities, so called *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 information model we use is the OTO-D data model. It is particulary suited to visualize object type hierarchies.

The principles and mechanisms are implemented in a set of framework services which are part of the Nelsis Framework. The Nelsis CAD Framework User Services allow engineers to interact with the framework, for instance to invoke tools or to browse information stored by the framework. The services include a hierarchy browser, a version browser, a design flow browser, and more.

Keywords

CAD frameworks, design flow management, integration platform, data modeling, user interface.

1. INTRODUCTION

Design descriptions and the tools that operate on them become more complex every day. Today design descriptions are usually constructed hierarchically, consist of multiple levels of abstraction (views) and have a status. Quite often an engineer wants to retain multiple versions of a design description. To manage these design descriptions and the tools that operate on them, a CAD framework is needed.

J. Goossenaerts et al. (eds.), *Information Infrastructure Systems for Manufacturing*

According to the CAD Framework Initiative "a CAD framework is a software infrastructure that provides a common operating environment for CAD tools. A framework should enable users to launch and manage tools; create, organize and manage data; graphically view the entire engineering process; and perform design management tasks such as configuration management and version management."

A framework that provides these facilities, is the Nelsis CAD Framework. The Nelsis CAD Framework is a flexible, light-weight framework that enables the engineering community to exploit high performance environments. The research and development of the Nelsis CAD Framework concentrates on the following key requirements:

- Openness
- Design data and design flow management services
- Configurability
- Performance
- Data Distribution
- Domain Independent
- User Friendliness

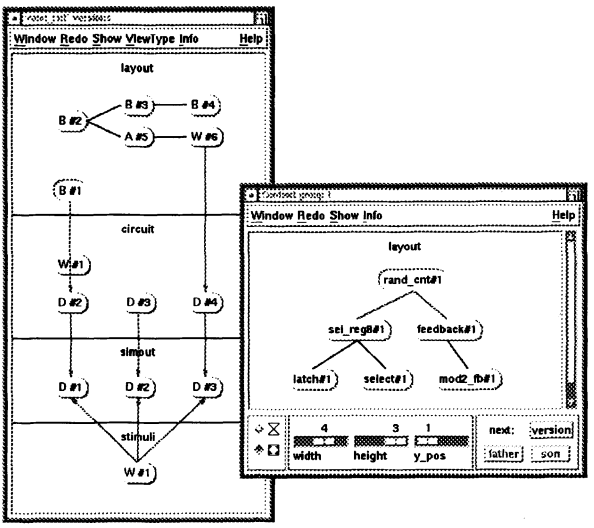

Figure 1. The graphical version browser displays version derivation graphs. Simultaneously a hierarchy browser displays the hierarchical decomposition.

The Nelsis CAD Framework Design Management Services perform tasks such as hierarchy and version management [Wolf88a] , design flow management [Bingle92a] , [Bosch93a] , [Wolf94a] , and access control[Hoeven94a]. The Nelsis CAD Framework User Services provide a graphical interface to the engineer. These services allow

engineers to interact with the framework, for instance to invoke tools or to browse information stored by the framework. The services include a hierarchy browser, a version browser, (Figure 1) a design flow browser (Figure 9), and more.

2. META DATA VERSUS RAW DESIGN DATA

The central theme in our approach is that the framework maintains information *about* the design descriptions and the design activities, so called *meta data*, rather than operating at the level of the detailed design descriptions [Leuken85a]. 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.

A *design object* is the smallest object management by the framework. It contains one or more design descriptions: The design objects may, for instance, contain detailed mask descriptions, schematics, or behavioral descriptions.

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). This raw design data is operated upon by the design tools. This approach is in line with work presented by [Katz86a] and [Batory85a].

The meta data is collected by the framework while the tools communicate with it to obtain access to the actual design descriptions. A well structured meta data pool can be used to enhance design system functionality in various respects:

- The framework itself exploits the meta data to perform various *services*, such as versioning, concurrency / access control, or design management. To provide these services the meta data is used as the scratch-pad of the framework itself.
- 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 engineer to interact with the framework and its meta data in order to keep track of the design process. The potential power of our approach is illustrated by the *framework browsers* which make the high level information about the structure and status of the design available to the engineer.

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 engineering environment, for data as well as functions. 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 engineering 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*.

3. META DATA ORGANIZATION

One of the major organizing principles in the Nelsis framework is the notion of *project*. A project provides a local context for the design activities of one or more engineers. It contains a collection of design descriptions that can be operated upon from within the project context. References across project boundaries are handled via an import-export mechanism, permitting projects to be used as library projects.

The meta data of a particular project describes which design descriptions are present in the project, how they are related and which operations have been performed on them. To describe which meta data has to be maintained by the framework and what its structure and properties are we have applied *data modeling* techniques. In contrast to work presented by [Katz86a] and [Batory85a] we have formalized the semantics of the meta data with well-defined abstraction primitives to yield an accurate definition of the information that is to be maintained. We selected the OTO-D data model [Bekke92a] to pursue our modeling activities. OTO-D stands for Object Type Oriented Data model. It is a powerful semantic data model, in which object types and their relationships can be defined, including various integrity constraints. The result of such a modeling activity is a *data schema*, which describes the (structural) organization of the application environment.

We have applied the OTO-D modeling techniques to formally describe such aspects as design object, hierarchy, equivalence, versioning, tool transactions, etc. The resulting data schema is depicted in Figure 2. We briefly explain the data schema; for more information on OTO-D and the data schema we refer to [Wolf88a].

With the OTO-D data model an *object type* is defined in terms of its *attributes*. An attribute relationship is graphically represented by a line that goes from the bottom of the composed type to the top of the attribute type, which may be either a base type or, again, a composed object type. Object types are represented as boxes, lines connecting corners of boxes indicate specialization and lines connecting boxes from the middle indicate attribute relationships.

The central object type in the data schema is *Design Object*. To a design object corresponds a design description (e.g. a schematic or a netlist) that may be accessed by the tools. A design object belongs to a module as one of its versions. Hence, *DesignObject* has the attributes *Module*, *Vnumber*, *Vstatus*, etc, to administer this module together with version-specific information such as the version number and version status. The transactions that tools perform on design objects are administered in the meta data using the object type *Transaction*. Design objects may be related by either *hierarchical relationships* or *equivalence relationships*. The object types *ImportedDO* and *Export* model the library mechanism, where design objects from one project can be imported (by reference) into another project.

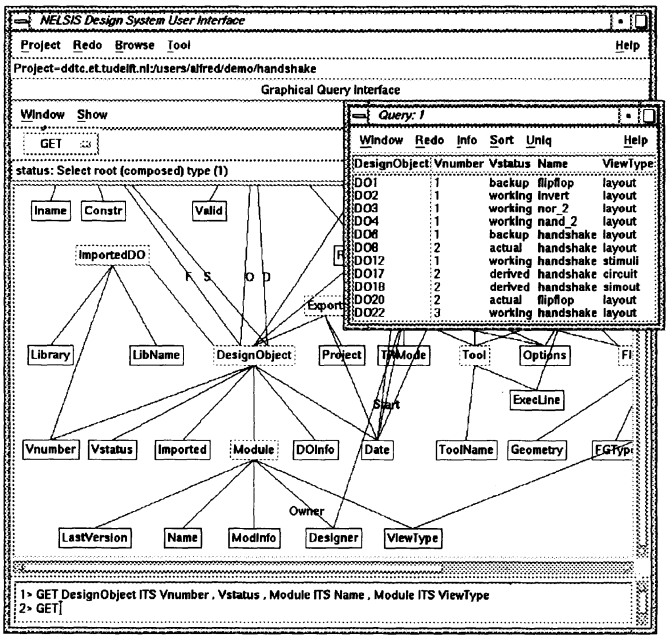

Figure 2. Data schema as constructed to organize the meta data

4. DESIGN FLOW MANAGEMENT

4.1 Requirements

We provide design flow management with a maximum of functionality and a minimum impact on existing tools (executable programs). We describe the concepts of flow management in relation to a real-life framework architecture, in relation to tool interfaces and in relation to kernel framework services.

Powerful flow management in a CAD framework includes:

- Defining concepts for flow description (flow statics).
- Defining concepts for flow execution (flow dynamics).
- Proper implementation in the software infrastructure that the framework provides, such that all potential functionality of the concepts becomes available.

For incorporating flow management into a CAD system we have the following requirements:

- Incorporating flow management should *not* result in restrictions on the tools that can be used in the CAD system; at least the current tool set, and other similar or well known tools, should be able to function in the CAD system (including interactive tools).

- Tools should *not* have to perform actions specific for flow management, e.g. call special flow management functions; tool sources should not have to be modified.
- It must be possible to start tools outside the flow management user interface, e.g. from a UNIX™ command shell. In this case flow management must still be active.

4.2 Terminology and concepts

In today's engineering environments engineers use design tools to create, modify and verify their designs. During this work the primary focus of an engineer is to successfully perform design tasks. Design flow management is concerned with the execution of tools in the correct order, according to the dependencies configured in a flow description.

In our concept for design flow description we treat tools as "functional units", which can be connected by "channels" to describe data transfer. By describing the communication between tools in terms of abstract "datatypes", file details are hidden from the user. So the user may think in terms of "schematic data", "test data", etc, while data of these datatypes is actually made up of several related files. Functional units can input and output data via "ports". A port is either an input port or an output port and corresponds to a specific datatype. Channels connect ports of the same datatype to let data flow from one functional unit to another. Both branches and merges of channels are supported.

Input ports can be of type "required" (data has to be present in order to run the functional unit), "optional" (data may be used by the functional unit) or "traversal" (used to traverse a hierarchical design). Output ports can be of type "extension" (the produced data is stored in an existing design object) or "modification" (a new design object is created for storing the produced data). Distinction between extension and modification is necessary to offer a flexible concept for logical organization of the design data to the tool integrators.

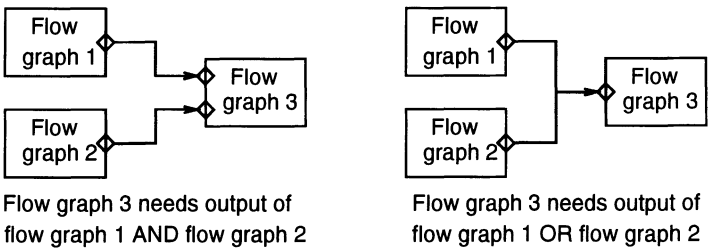

Figure 3. Basic operations with flow graphs

An "activity" is a functional unit which can run only when data is present for all its input ports of type 'required' and which produces data for all its output ports, every time it executes successfully. Multiple activities must be defined for a tool if the input / output characteristics are different for these activities. Generally, each activity will correspond to one of the design functions of a tool. A "flow graph" is either an activity, or consists of several other flow graphs. This enables hierarchical specification of flow graphs and provides the expressive power needed to model design tasks. Examples of flow graphs

are shown in Figure 3 and 4. Figure 3 shows the basic "and" "or" operation. Figure 4 shows that activity "act1" produces data of two datatypes: one is used as input by "act2" and "act3", the other is used as input by "act3" only. The two input ports of "act3" indicate that *both* datatypes are needed as input. "act4" needs input data produced by "act2" *or* "act3".

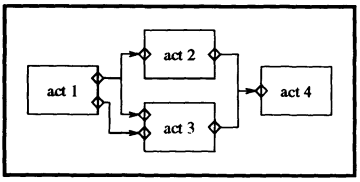

Figure 4. Flow graph with activities (rectangles), ports (diamonds) and channels (arrows)

4.3 The information model

The information model shown in Figure 5 is the model that is currently being used for design flow management. It is an OTO-D data schema and visualizes an object type hierarchy.

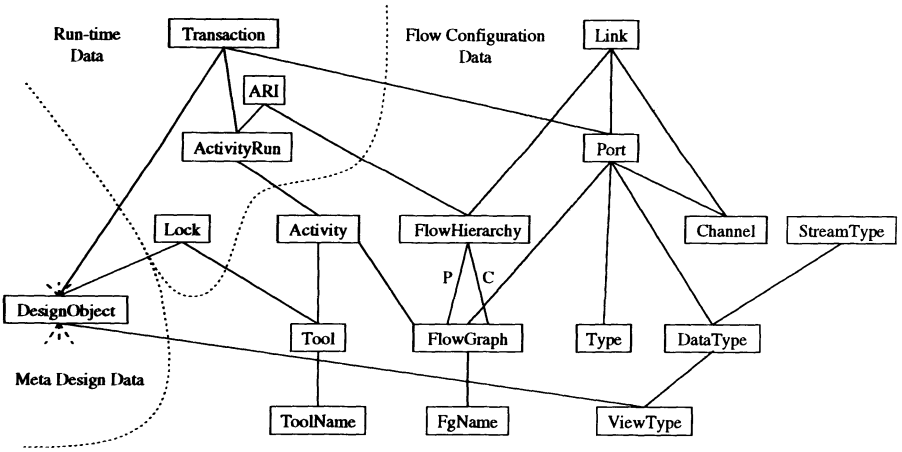

Figure 5. The information model used by the framework

Thus Activity *is a* (specialization of) FlowGraph, and Activity *has a* Tool as attribute, indicating the tool it is an activity of. The information model consists of parts for meta design data, flow configuration data and run-time data.

- *Meta design data* contains information about the design, such as versioning information, hierarchical relationships between different parts of the design and equivalence relationships. A detailed description of an information model for meta design data is given in [Wolf88a].

- *Flow configuration data* specifies the available tools, their activities and the relations between activities. The flow configuration is typically brought in by a design methodology manager before the actual design work is started. The top level flow graph is called a "flowmap".
- *Run-time data* is generated during the design process to correctly administer the state of design. It can be exploited to:
  - enforce the consistency of design data,
  - inform the user about the state of the design, and
  - decide what tools can or should be run (tool scheduling).

The meta design data and the run-time information are updated by the framework during each tool run according to the operations performed on the data by the tool. The flow configuration information changes only when tools or flow graphs are added, removed or modified.

5. THE CAD SYSTEM ARCHITECTURE

The CAD system architecture presented in Figure 6 has been widely accepted by standardization bodies and the industry [Liebis92a] and [CFI92a]. Therefore we believe that the approach described is applicable to other CAD systems. Figure 6 shows that tools, which may be encapsulated tools, tightly integrated tools or framework tools, are integrated on top of a CAD framework. The framework provides various services, such as data management and flow management. The framework services consult and maintain the "domain neutral data" (meta data / framework data). Design tools obtain access to the "domain specific data" (design data / tool data) via the framework.

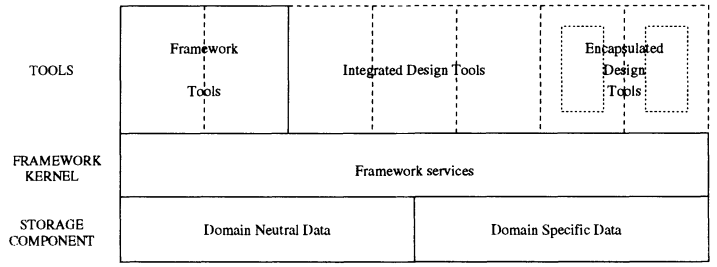

Figure 6. CAD system architecture

Figure 7 shows the architecture of the Nelsis CAD system. As can be seen this architecture conforms to the general CAD system architecture presented above.

Additional details convey the internal structure in the Nelsis CAD system. Flow management is a service in the framework kernel, and not part of a desktop or other framework tool. This ensures that tools cannot simply by-pass flow management. Domain neutral data is stored in a special purpose lightweight DBMS that can be configured with an OTO-D data schema, such as the information model in Figure 5. The DBMS provides a SQL-like query interface for convenient data retrieval and manipulation. The domain specific data is currently stored in the UNIX™ file system

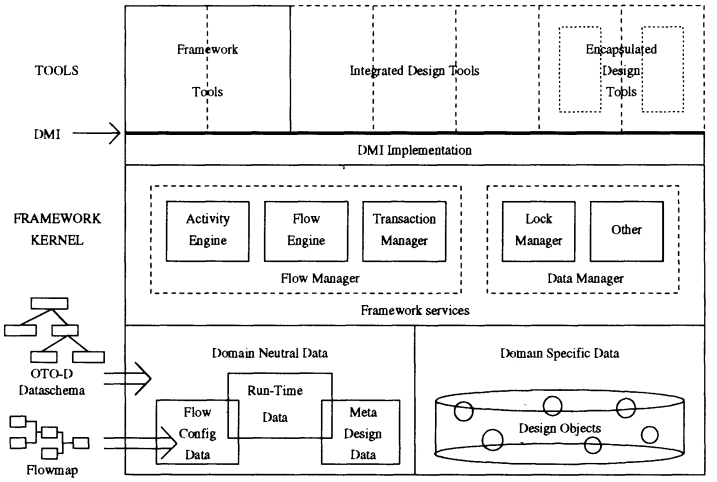

Figure 7. Nelsis CAD system architecture

thereby providing efficient and compatible storage for many of the file based design tools. Domain specific data is organized as design objects. A design object *is* a version of (a part of) the design and contains data of a certain view type. E.g., a design object may contain schematic data of a flip-flop. Per design object data is organized as one or more data "streams" that may, for example, correspond to design files.

The framework services have the following functionality:

- **Flow Management**:
  - *Activity Engine*: Checks the data access operations of a tool against the tool's configured activities. Identifies the activity the tool is performing.
  - *Flow Engine*:
    1. Determines the internal state of design objects (availability / validity of data in design objects).
    2. Checks whether activities are allowed concerning: (1) input data and (2) the flow configuration. That is, it determines whether the producer of the input data is a valid producer for the activity being performed.
  - *Transaction Manager*: Administers activity runs and transactions describing the use of design objects in an activity run.
- **Data Management**:
  - *Lock Manager*: Locks and unlocks design objects for coordinated multi-user access. It prohibits conflicting operations, such as multiple concurrent write operations, on the level of design objects.
  - Other: E.g. configuration management, hierarchical consistency management and version management.

Tools interact with the framework via the Data Management Interface (DMI, a procedural interface) to obtain access to design data. Tightly integrated tools perform calls to the DMI functions in the source code; for encapsulated tools a 'wrapper' program performs the necessary DMI calls. By offering a proper set of "anchor points", the standard DMI greatly facilitates software exchangeability and permits framework and tools to evolve separately to a large extent. The DMI does not incorporate any specifics for purposes of flow management. Being a pure and simple interface geared towards data access only, the framework services are hidden from tools. The calling pattern in Figure 8 illustrates how the DMI functions cooperate.

```
DM_PROJECT projectkey;
DM_DESIGNOBJECT desobjkey;
DM_STREAM streamkey;

dmInit (toolname);
  projectkey := dmOpenProject (projId, mode);
    desobjkey := dmCheckOut (projectkey, desobjId, mode);
      streamkey := dmOpenStream (desobjkey, streamId, mode);
        dmPutDesignData (streamkey, format, arguments);
        dmGetDesignData (streamkey, format, arguments);
      dmCloseStream (streamkey, mode);
    dmCheckIn (desobjkey, mode);
  dmCloseProject (projectkey, mode);
dmQuit ();
```

Figure 8. DMI calling pattern

6. RESULTS

The design flow management of the Nelsis CAD framework has been completely transparent to the tools since design flow management has been implemented "underneath" the standardized DMI. The DMI has currently been stable for over 8 years. During this time the framework implementation evolved from a simple single-user, single-host data management system, that was little more than an abstract file system, to the powerful multi-user fully distributed framework that it is today, relieving engineers from many burdens and directly helping them to concentrate on their main task: *design.* This clearly proves the DMI paradigm.

Since the framework keeps track of the data accesses performed by tools, activities can be recognized while the tools are running. Thus the modified internal state of design objects can be displayed before tools finish execution. This is of special interest for long running interactive tools such as, editors or interactive simulation environments, with which engineers may perform multiple design actions during a single tool run.

Based on the implemented flow management components we have implemented a graphical flow browser that visualizes the state of the design. The flow browser obtains the state of design objects from the flow engine. It displays the existence of data on input and output ports and highlights the flow graph instances, using colors or line styles, to reflect the flow status. Using the mouse, engineers can start activities and zoom into hierarchical flow graphs. The flow browser can also start activities automatically and give advice on tool scheduling based on the flow configuration data. The flow browser is

part of the Design System User Interface [Leuken95a], the central graphical user interface of the Nelsis CAD Framework, which integrates information retrieval, design object selection and tool activation in a coherent user interface. Figure 9 shows a window dump of the flow browser displaying status information on version number 3 of a design called 'rand_cnt'.

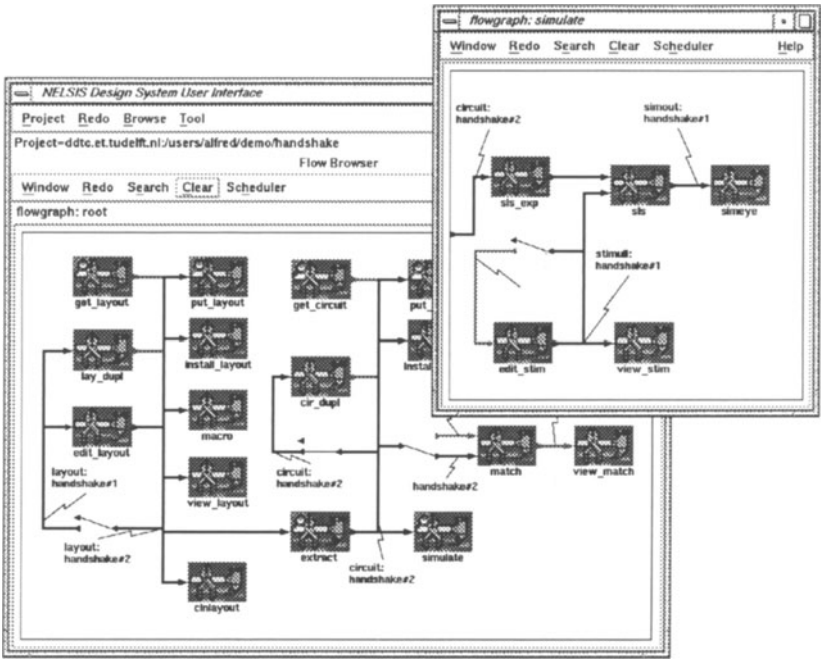

Figure 9. Flow browser window

Thick lines and filled ports indicate that data is present, a dotted box indicates that a flow graph is not executable, a dashed box indicates that a flow graph is executable and a solid box indicates that a flow graph has executed. Figure 9 shows that the box simulate is a compound. In simulate the program 'sls' has been executed, and the program 'simeye' can be executed. In the flowgraph root, the program 'match' cannot be executed because not all input data is available.

7. CONCLUSIONS AND FUTURE WORK

The Nelsis CAD Framework has been implemented in the C-language on the UNIX™ operating system. The graphical framework tools, such as the flow browser in the DSUI, make use of OSF/Motif™ and the X Window System™.

Evaluations of the implementation have shown that the ideas do indeed provide a fully functional design system that guides engineers through the different design steps, shows them the state of the different parts of their design and enables convenient automatic and

manual activation of design tools.

The user community of the Nelsis CAD Framework includes IMEC (Belgium), University of Southern California (USA), CRS4 (Italy), GMD (Germany), Philips (NL), and others. The application areas are among other: automated software testing, semiconductor design (synthesis), climate studies and image processing. In general the Nelsis CAD Framework is used for system design and system simulation environments. Whereas these users have reported positively about the effectiveness of the technology provided, they have also expressed the need for the 'next level of automation' to support coordination and communication among engineers in the larger engineering projects. Research and development activities in this area have started.

References

Wolf88a. P. van der Wolf and T.G.R. van Leuken, "Object Type Oriented Data Modeling for VLSI Data Management", pp. 351-356 in *Proc. 25th ACM/IEEE Design Automation Conference*, , Anaheim (June 1988).

Bingle92a. P. Bingley, K.O. ten Bosch, and P. van der Wolf, "Incorporating Design Flow Management in a Framework Based CAD System", pp. 538-545 in *Proc. IEEE/ACM International Conference on CAD - 92*, , Santa Clara (Nov 1992).

Bosch93a. K.O. ten Bosch, P. van der Wolf, and P. Bingley, "A Flow-Based User Interface for Efficient Execution of the Design Cycle", pp. 356-363 in *Proc. IEEE/ACM International Conference on CAD - 93*, , Santa Clara (Nov 1993).

Wolf94a. P. van der Wolf, K.O. ten Bosch, and A.J van der Hoeven, "An Enhanced Flow Model for Constraint Handling in Hierarchical Multi-View Design Environments", in *Proc. IEEE ICCAD 1994*, (1994).

Hoeven94a. A.J. van der Hoeven, T.G.R. van Leuken, K.O. ten Bosch, and P. van der Wolf, "A flexible Access Control Mechanism for CAD Frameworks", in *Proc. Euro-Dac 1994*, (1994).

Leuken85a. T.G.R. van Leuken and P. van der Wolf, "The ICD Design Management System", pp. 18-20 in *Proc. IEEE ICCAD - 85*, (1985).

Katz86a. R.H. Katz, M. Anwarrudin, and E. Chang, "A Version Server for Computer-Aided Design Data", pp. 27-33 in *Proc. 23rd ACM/IEEE Design Automation Conference*, (1986).

Batory85a. D.S. Batory and Won Kim, "Modeling Concepts for VLSI CAD Objects", *ACM Trans. on Database Systems* **10**(3) pp. 322-346 (Sept 1985).

Bekke92a. J.H. ter Bekke, *Semantic Data Modeling*, Prentice Hall, Englewood Cliffs, N.J. (1992). ISBN 0-13-806050-9

Liebis92a. D.C. Liebisch and A. Jain, "Jessi-Common-Framework Design Management - The Means to Configuration and Execution of the Design Process", pp. 552-557 in *Proc. EURO-DAC 92*, , Hamburg, Germany (Sept 1992).

CFI92a. CFI, *Framework Architecture Reference*, CAD Framework Initiative Oct 1992.

Leuken95a. T.G.R. van Leuken and et al., *The Nelsis CAD Framework Documentation*, Dimes Design and Test Centre, Delft University of Technology (1995).