Issues and Answers in CAD Tool Interoperability

Mike Murray
Acuson
Mountain View, Ca. 94039
mikem@acuson.com

Uwe B. Meding
Concurrent CAE Solutions
Santa Clara, Ca. 95050
ccaes@netcom.com

Bill Berg
Mentor Graphics
Wilsonville, Or. 97070
bberg@wv.mentorg.com

Yatin Trivedi
Seva Technologies
Fremont, Ca. 94539
trivedi@seva.com

Bill McCaffrey
High Level Design Systems
Santa Clara, Ca. 95054
billmc@hlds.com

Ted Vucurevich
Cadence Design Systems
San Jose, Ca. 95134
tedv@cadence.com

Abstract- CAD tool interoperability issues are a recurring impediment to constructing a design methodology, especially if the methodology incorporates point tools from several vendors. Failures in data transfer between tools often arise unexpectedly, and can delay design work until a workaround is developed. This paper examines interoperability issues for four classes of tools: schematic tools, simulation and synthesis tools, IC physical design tools, and workflow management tools. This paper also describes current research in modeling interoperability problems. Using this paper, the reader can develop a checklist of potential interoperability issues in his CAD environment, and address these issues before they cause a design schedule slip.

1. Introduction

Why is CAD tool interoperability such a ubiquitous problem? Any experienced CAD manager, particularly one using tools from multiple CAD vendors, can recite the list of the difficulties he's encountered integrating tools into a design flow. Yet compared to the technical innovation required for algorithmic developments in simulation, synthesis, or place and route, interoperability problems appear easy to solve. After all, it's just an issue of moving structural, behavioral, physical, and process design data around!

The answer is "Interoperability should have been easier, but it wasn't". To understand why, we have to look at how CAD vendors and users got to this point. The CAD industry was in its infancy when most of the major tools were developed to service the design community. CAD tool design specifications were technologist driven, as opposed to user driven. Consequently, tools were locally optimized for features and performance, and interoperability was at best an afterthought. During the mid to late 80's CAD companies began to recognize that interoperability was becoming a user requirement. CAD companies believed that they could achieve a competitive and technological advantage by making their tools interoperate better than their competitors. They approached the problem of tool interoperability by taking their existing solutions and attempting to glue them together. However, glue alone could not compensate for the tools' not having been designed for interoperability. Furthermore, in spite of vendor initiatives such as CFI, the glue was unique to each vendor. Major vendors typically focused on integrating their own offerings, rather than on designing tools which could cooperate in the typical customer's multi-vendor CAD environment.

CAD tool interoperability is an important problem because without interoperability, CAD users are not free to use best-in-class point tools at each step in the design process. In addition, CAD customers are finding the need to exchange design information both internally and with external organizations. With different CAD tool suites being used by different groups, companies who wish to use design information from other groups have found the limiting factor to be the format of the data itself. In these environments, design transportability and tool interoperability are critical components to the long term success of the data transfer and ultimately, to the design project.

The next four sections of this tutorial paper describe current interoperability problems in four classes of tools: schematic capture tools, simulators and synthesizers, IC design tools, and workflow management tools. Each section gives specific examples of problems which arise when data is passed among tools. There is no new theory here. Instead, the objective of these sections is to disseminate real world experience in applying CAD tools to commercial design problems. The final section of this paper describes research work which may help reduce interoperability problems over the long term.

2. Schematic Capture Tools

This section presents problems which may be encountered passing data between schematic capture tools. This section is based on consulting work done by Concurrent CAE Solutions for Exar Corporation. Exar's objective was to move existing design information from the Viewlogic Viewdraw schematic tool to the Cadence Composer schematic tool.

Exar was facing a typical interoperability task. Exar's goal was to maintain their schematic front end in Viewlogic, and at the same time use several of the Cadence back end capabilities, like cross-probing. Exar had an existing investment in both Viewlogic Viewdraw schematics, and in Cadence Composer libraries. Switch level and analog modeling were key components of Exar's design methodology, and Exar had invested time and resources in qualifying the Composer libraries. These libraries included component description formats, technology rules, and schematic symbols.

Exar's requirements included taking the existing schematics from the Viewlogic system, and replacing the Viewlogic primitive library components with existing library components from the Cadence system. As shown in Figure 1, this component replacement required ripping up specific existing components, along with the segments of the nets connected to the pins of those components. The ripped up net segments were then rerouted to the pins of the replacement components symbols. The number of ripped up net segments was minimized, and the resulting Cadence schematic with the replaced components appeared graphically very similar to the original Viewlogic schematic.

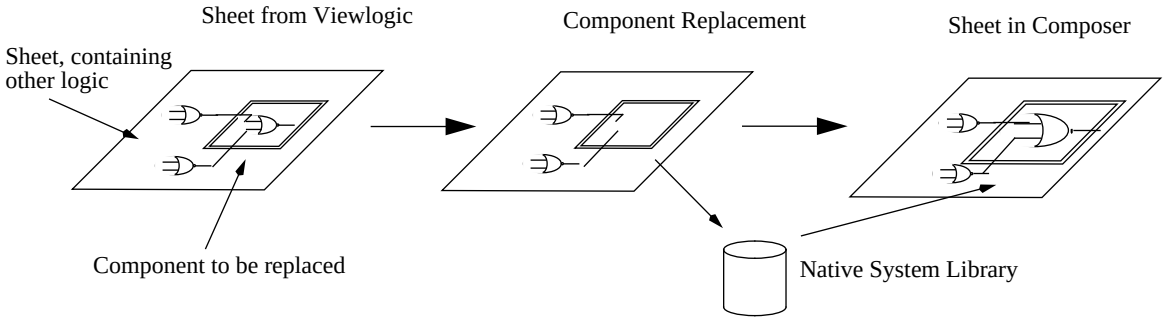
In transferring schematic data between Viewlogic and Cadence tools, the following issues had to be addressed:

Scaling: The schematic symbols used on the Viewlogic schematics were drawn on a 1/10 inch grid with a 2/10 inch pin spacing. The target Composer symbol libraries were drawn on a 1/16

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

Figure 1: Component Replacement



inch grid with a 2/16 inch pin spacing. The symbols and schematics were scaled down in size to adjust to the Composer grid spacing.

Symbol replacement mapping: Maps were created for replacing components between the Viewlogic and Cadence systems. Library, name, and view mappings, along with origin offsets and rotation codes, were defined for each Viewlogic component to be replaced by a Cadence component. For situations where pin naming conventions differed, a pin name map was also created.

Standard property mapping: Rules were defined for mapping standard properties and labels between the two systems. The mapping included the addition, deletion, renaming or changing of property names, values, and text labels.

Non-standard property mapping: Special property mapping requirements for analog properties required the reformatting of single properties into multiple properties. These requirements were handled by the addition of Access Language (a/L) callbacks for a selected set of objects. Concurrent CAE Solution's a/L is a Lisp dialect and is set up so that a user can interact with the entire design hierarchy during the migration process. By using the a/L interpreted language to handle the unique formatting requirements, Exar achieved a high degree of automation with no manual post translation cleanup.

Bus syntax translation: Viewlogic and Cadence differ in their definition of legal bus syntax. For example, Viewlogic allows condensed busing syntax, i.e. "A0" is equivalent to bit 0 of bus A<0:15>. However, Cadence requires that bus syntax be explicit, and A0 is not equivalent to A<0>. Translation rules were created to map between syntaxes. Viewlogic and Cadence also differ with respect to bus postfix indicators. Viewlogic permits the use of postfix indicators such as the minus sign in the "myBus<0:15>-". This syntax is not understood by Cadence. For these nets, the postfix indicators were adjusted to keep the net names unique (myBus-<0:15>).

Hierarchy and off page connectors: Viewlogic and Cadence Composer maintain different rules defining the interaction between different levels of a design hierarchy. Viewlogic does not require the explicit use of either hierarchy or off-page connectors, however, Cadence Composer requires both hierarchical and off page connectors. Translation rules were set up governing the mapping and addition of the hierarchy and off page connectors. These rules defined translations of the type of connector (input, output, or bidirectional); the library, cell, and view names of the connector; and any offsets or rotations codes. Off page connectors represented a more difficult task, because Viewlogic connects same signal names across multiple pages implicitly. However, Cadence Composer requires these connections to be explicit by using off-page connectors. The connectivity challenge was addressed by maintaining an understanding of the connections during the migration process. The geometrical challenge was addressed by adding off-page connectors to the end of wires if a floating wire was determined, or to the side of the schematic sheets for these internal connections.

Globals: Rules were defined for the labels, names, and/or instances of objects, and how they were mapped to the corresponding instances on the target system. Similar to the replacement of components, offsets and rotation codes were required to map the replaced components to the correct location on the translated schematic. When the schematic was received by the target system, it used global instances and connectors from the native component libraries.

Cosmetic issues: A host of cosmetic issues existed, including the height and width of fonts. Font characters in Viewlogic are typically smaller than in Cadence, and the origin of each character is offset from the baseline. For example, if the character "E" is placed on a line in Viewlogic, it may appear as an "F" when translated directly to Cadence Composer. Rules for character scaling and offsets were defined in order to correctly align text.

Verification: Careful design of a data translation strategy is insufficient to guarantee correctness of the translated data; design data translations must be independently verified. For this translation, Exar performed the verification using their own software package.

3. Simulation and Synthesis Tools

The section presents problems which may be encountered in passing data among simulation and synthesis tools. It is based on the consulting experience of SEVA Technologies.

3.1 Simulator issues

Language standards: SEVA Technologies' product evaluations in 1994 and 1995 showed that most commercial products lack compliance to language standards, such as IEEE 1076 for VHDL and OVI LRM2.0 for Verilog HDL. (Verilog was subsequently adopted as the IEEE 1364 standard.) Even when standards exist, they may be incomplete or controversial. For example, simulation results depend on the scheduling algorithm the simulator uses to order and process events. Different Verilog simulators can legitimately disagree on the outcome of the same simulation, because the simulation cycle and processing order for simultaneous events are not completely defined by the language. Typically, if different simulators give different results when simulating the same model, there is a race condition in the model being simulated, and the potential for a bug in the real hardware. However, determining whether a discrepancy between the simulations is due to a model race condition or to a simulator bug can be troublesome.

```
assign a = b & c;
always begin
...
b = d;
 if (a != d) // which value of a?
```

Backward compatibility: Simulator timing models can change as new versions are released, causing simulation timing results to drift unless backwards compatibility is specifically addressed. For example, Verilog-XL, a popular Verilog simulator, supports the "+pre_16a_path" command line option. This option forces simulators with version 1.6a or later to use the same timing check behavior as was used prior to the 1.6a version. Users frequently use this option to ensure that certain modeling styles remain compatible across the versions of this simulator.

Co-simulation: Making two simulation tools work together, specially a Verilog HDL - VHDL co-simulation, is typically problematic. Although co-simulation attempts have been made by all major CAD vendors, most have fallen short of their targets. Inconsistencies in the signal value set (e.g. 0, 1, x, and z) and in the simulation cycle definition are common sources of problems.

Environment: In addition to language, other elements of the simulation environment have not been standardized. If the design environment uses multiple simulators, it is difficult to write a single script for running the simulation, as the command line options and user interaction mechanisms vary considerably between interpreted and compiled code simulators. Usually, this is not a major problem as one group of users (e.g. designers) use one simulator and the other group (e.g. verification engineers) use another simulator.

3.2 Synthesis issues

Language standards: Standards for language and design environment are also an issue for synthesizers. For each HDL and synthesis tool, there exists a subset of the HDL that the synthesis tool can accept. However, for a given HDL, there is no standardization of the synthesizable subset across synthesis vendors. The lack of standardization is understandable, because synthesis vendors can differentiate themselves through the HDL subsets that they accept. Consequently, if a model will be transported between synthesis tools, it should be written using only those HDL constructs contained in the intersection of the vendors' subsets.

Environment: In addition to differences in the HDL subsets that they accept, synthesis tools also differ in the specification or contents of design constraint files, technology libraries, report generation, and runtime control mechanisms (e.g. command line options and user interfaces). These differences make it nearly impossible to migrate a design synthesis description from one synthesizer to another without significant effort and schedule impact.

Modeling style: Synthesis tools sometimes interpret more than what is described in the model. For example,

```
always @(a or b)
    out = a & b & c;
```

You would expect the signal out to be modified when a or b changes. However, the synthesis software interprets your model as if out was sensitive to signals a, b and c.

```
always @(a or b or c)
    out = a & b & c;
```

The advantage of generating combinational logic may not be acceptable to your latch-based architecture!

3.3 Naming issues

Several interoperability problems which affect both simulators and synthesizers are related to the identifiers used in HDL models.

Name length: Even though an HDL definition may allow arbitrarily long names, several PC based simulators consider only the first eight characters as significant. Two names that differ after the first eight characters are treated as the same name, causing undesirable aliasing. For example, cntr_reset1 and cntr_reset2 are treated as the same as cntr_res.

Escaped identifiers: Several simulators and HDL based tools differ in their definition and interpretation of legal characters for identifiers, and how illegal characters can be escaped. Specifically, in Verilog HDL, the use of escaped identifiers (names that begin with \ and terminate with a white space) has been confused by many simulators. Some analysis tools always assume that the use of [] implies a bit on a bus, or a * implies an active low signal. Such specific interpretations are not valid across all tools,

Keywords: VHDL and Verilog differ in their definition of keywords and legal identifier names, and this difference can complicate the translation of models between languages. For example, "in" and "out" are valid Verilog HDL identifiers (e.g. signal names) that are reserved keywords in VHDL. Even if a translation tool can rename Verilog identifiers so that VHDL syntax errors are avoided, the identifier names will no longer match between models, and simulation analysis scripts may need to be modified.

Hierarchy removal: Certain HDL based tools work only on a flat design description as opposed to a hierarchical model. When such a tool imports a hierarchical design, it must flatten the design. New names get derived in some systematic way, such as joining the names in a hierarchical path using an underscore. However, the design process is often iterative, and if a a problem is found in the flat representation, the user must map back to the name used in hierarchical representation.

Before beginning a project, a user should study the naming conventions used by the tools he will use, and adopt a naming convention which will minimize problems such as those listed above.

3.4 Hardware and software platform dependencies

Quite frequently, interoperability problems are really manifestations of transportability problems, i.e. being unable to move existing work to a different environment; be it a hardware platform, operating system, or simulation or synthesis software. There are several problems faced during a design cycle that are related to the hardware and operating system used for running design tools. Typical problems include:

Nonstandard operating system commands: Certain system commands for identification of hostname, hostid, and Ethernet id are different across different versions of UNIX. Similarly, the commands for creation and expansion of swap space and for accessing remote file systems vary across platforms. This lack of standardization makes system administration harder to perform.

Office / home computing incompatibilities: Portability of scripts from one software platform to another platform is limited. For example, if an engineer is using a UNIX workstation at his office and a personal computer at home, he require two sets of scripts to run simulations and to copy files back and forth. Scripts may even not be portable between platforms running different flavors of Unix. Furthermore, the engineer's preferred simulator or waveform display program may not be available on both his office and home platforms, forcing him to learn two user interfaces. These problems reduce the productivity gains which could otherwise be realized by telecommuting.

Tool version skew: Even if a CAD vendor has ported a tool to all of the platforms in use on a design project, the vendor may not support all platforms equally. Bug fixes and new tool releases sometimes take weeks to propagate across all of the platforms a vendor supports. Before purchasing a tool, the user should verify the vendor's track record in supporting the platforms the user will be using.

Hardware interfacing: The interface required between a workstation and a special purpose hardware box such as a Quickturn emulator or an IKOS hardware accelerator is different for different vendors. These interfaces differ in cabling, connectors, device drivers, installation, and administration. They also differ in their user interfaces. These differences makes it harder to change the hardware and/or software computing environment during a project.

Extension languages: Different hardware and software vendors provide different C or C++ compilers and libraries. Verilog simulators provide a PLI (programming language interface), which allows the user to link custom C language modules to the simulator. Compiling and linking these modules into a Verilog simulation requires the user to be familiar with the compiler for his computing platform, and with the linking procedure for his simulator.

3.5 Tool Command Languages:

There is no standardization on the language used to integrate tools and manage workflows. TCL, Skill, Perl, and Unix shell are all in widespread use. Unless a company adopts and enforces a standard for an integration language, sharing and reuse of design methodologies within that company will be limited. Section 5 describes how to choose a workflow management tool which can promote consistency in a company's design methodology.

4. IC Floorplanning and Place and Route Tools

This section describes interoperability problems between floorplanners and P&R (place and route) tools used for physical design of integrated circuits.

Today's process technologies provide designers with the ability to develop very complex, transistor rich designs that can operate at frequencies above 200MHz. However, capitalizing on the potential offered by these process technologies is not easy. Time to market pressures and increasing design complexity require that more and more of the design be done with CAD tools. Signal delay and integrity are now significantly affected by interconnect topologies, adding complexity to the design process and requiring a higher degree of input control to the P&R tools. Design floorplanning is now a requirement for managing signal delays. However, floorplanning is effective only if the floorplanner's results can be used to control the P&R tools which will be used to complete the physical implementation. Frequently, the problem of communication between floorplanners and P&R tools is complicated by the designer's use of multiple P&R tools on a single design.

High Level Design Systems provides the designer with multiple levels of floorplanning capabilities which can drive directly into a place and route backplane. Almost all of today's P&R tools are supported from this backplane. In developing this backplane, HLD has had to deal with the lack of interoperability between P&R tools. For example, there are no common languages, syntaxes, or semantics between these tools. While there have been efforts to create standards such as PDEF to support some timing related placement, there is currently no effort to define a common language to deal with the semantics of both control and data input into the various P&R tools. HLD's P&R backplane is the best attempt to at least map the semantics and controls from one tool to the next. The lack of interoperability between these tools decreases the designers' ability to properly influence the P&R tools. Interoperability problems in cell definition, block floorplanning, and interconnect topology are described below.

Cell definition: All P&R tools require an abstract view/definition of the design cells or blocks that they are to assemble. These abstract views consist of many parts including cell/block boundaries, site types, legal orientations, a complex (and sometimes comprehensive) set of pin data, and routing blockages. How this data is defined and input is different for most P&R tools. (While there are groups of tools that support some commonality, there is minimal consistency over all tools). To demonstrate the complexity that the designer must deal with, let's examine pin definitions.

Pins are the connection targets for the router. The parts of a pin are: a name, location, shape, layer, and a set of connection properties. The connection properties include access direction, multiple connect, equivalent connect, must connect, and connect by abutment. Each P&R tool supports a slightly different set of input data requirements. For instance, some tools read access direction as a property, while others try to determine it from the routing blockages. The latter approach adds a level of complexity to how blockages are defined for each tool. Connection types are also not uniformly supported. Some tools read connection types as a set of literal properties on the pin, others require an external file, and a few have no predefined support for some connection types.

Block floorplanning: Communicating area constraints from a floorplanner to a P&R tool is also, unfortunately, an exacting process. During floorplanning, a designer makes decisions on block aspect ratios and size, general and literal pin locations, and special blockages marking keep out zones. He also defines the general routing strategies for global signals such as power, ground and clock. Once the designer is satisfied with the floorplan, he must then convey all of the appropriate information to the P&R tools. Though HLD's P&R backplane conveys as much as possible to the various P&R tools, each tool requires a specific set of constraints.

Interconnect topology: Interconnect topology has a large impact on design performance and functional integrity. Consequently, the designer must pass topology control information between the floorplanner and the P&R tools. Coupling capacitance can causes all sorts of problems, but can be controlled by shortening wire length, increasing spacing, or even by shielding. Minimum metal widths are also only appropriate for typical drive currents; wider widths must be used for nets with larger currents. Therefore, routers should be able to accept width specifications for selected nets. Some tools can not support these requirements, while those that do are inconsistent in their language or semantics.

There are many other issues with communicating constraints into P&R route tools, but the above issues are sufficient to demonstrate the global issue of interoperability. The designer is really dealing with two separate problems. First, the feature sets of the various tools are sometimes significantly different. Though vendors will argue that these features competitively differentiate their tool from another tool, it can also be argued that there is a required set of features that must be understood by all tools. Second, when such features are supported, there is no standard as to how they should be defined and presented to the designer.

HLD's place and route backplane addresses many of these issues. However, designers, floorplanner vendors, and P&R vendors need to cooperate to standardize on a complete set of requirements and semantics for properly feeding design constraints forward through the design process. This standardization can be done without compromising the competitive features and performance of the various CAD products.

5. Workflow Management Tools

An integrated design environment must include more than just the ability for one tool to share data with another tool. There is a need to address the "process gaps" that also typically affect the efficiency and robustness of the design process. Companies have found that simply adding better point tools, no matter how well integrated, does not provide the same incremental productivity gain as in the past. What is now required to achieve and maintain a competitive advantage in the market is a focus on the "process" - whether that process includes gathering product requirements, design, manufacturing or maintenance. The type of workflow management support required goes far beyond the simple "tool invoker/tool sequencer" approach currently found in many process management tools.

Creating a workflow involves first capturing the structure of the flow graphically. Next, the work that occurs within the flow as the process is followed is specified. Once the workflow is captured and specified, the resulting workflow template is deployed across the organization. Each instance of the captured process is derived from the same template, providing process consistency. As the workflow

progresses, status is collected and reported to the end-user and to management as required. These collected metrics can later be analyzed and used to tune the process, providing a closed-loop, continuously improving process environment.

Mentor Graphics has found that a workflow product suite must have the following characteristics to successfully deliver a workflow-enabled, integrated design environment:

Environment independence: No vendor has a monopoly on the tools included in the design environment. A successful workflow management environment must support any combination of tools from multiple vendors. The workflow management tool itself must be framework independent such that it does not require a particular vendor framework be resident (i.e. purchased!) to operate. The same integration facilities and process should be used to create the workflow independent of the tools being used in the process.

Open language environment: Many processes are controlled currently via a series of shell scripts and other procedures that are held together by the user's own experience about what the procedures do and the order in which they are to be executed. This experience can be leveraged if the workflow system maintains a clear architectural separation between the process description and the process actions. This means that the actions invoked from the process description can be implemented in any programming language desired by the flow developer - UNIX shell scripts, PERL, TCL/TK, C-language, etc. This openness allows any existing programs, executable from the UNIX command line, to be attached as actions to a workflow without the use of special compilers, proprietary languages or wrappers, or other tool-specific means.

Flexible tool management: A workflow and a set of integrated tools can co-exist in a number of different ways depending on the type of process flow and the capabilities of the individual tools. For example, a workflow may consist of a number of separate steps, each of which causes a separate tool to invoke. Another workflow may consist of the same number of steps, but in this case each of the steps causes a separate feature of a single tool to be executed. In the first case, each tool is invoked as a separate process and the return value (or other means) is used to determine the success or failure of the step. In the second case, the first step in the sequence invokes the tool (if not already invoked), then subsequent steps communicate to the already- running tool via inter-process communication or RPC protocols (both of which can be used within the workflow).

Default behavior, not built-in policies: There is a fine architectural line between providing an extremely programmable system and a system that is extremely difficult to program. During the course of workflow capture, if the developer must tell the system exactly what to do in every case, and always account for even the most commonly-expected behavior, then creating a workflow can become tedious and error-prone. For many tool integrations, a return status of "zero" from the tool will indicate successful execution; a return of "non-zero" status will indicate failure. For this common case, defaults must be built into the workflow system to react appropriately. For example, a tool invoked from a workflow step that returns zero status will be assumed to have completed successfully, and the workflow status for that task will be updated appropriately by default, without the developer having to explicitly set the task state to "completed successfully". However, if a more complex integration is required, support is provided in the API to set the state of a step to an explicit value based on whatever criteria is necessary to determine the completion status of the tool invoked from that step. This type of flexible default behavior allows any tool to be successfully integrated into the workflow environment.

Support for hierarchical design: Hierarchical design is a given, and tools integrated into a workflow-managed environment must be allowed to operate on hierarchical data if designed to do so. The process flow must adjust dynamically to the changing hierarchy of the design (especially in the early "what-if" stages). Each design block in the hierarchy can be developed using the same sub-flow template, but the data and process status is kept separate for each block. Blocks can be named and organized within the workflow such that a natural design hierarchy is visible and obvious to the user.

Open and flexible support for data management: A default design data management structure can simplify the design process and allow designers to concentrate on design instead of data management. Most designers do not care where their data is stored, as long as they can find it easily and their tools operate on it correctly. A default data storage structure can be embodied in the workflow, built by running an initial setup action, and then added to and/or re-used as the workflow progresses. This approach isolates the designer from the details of data location and access.

Architectural separation of workflow and data management: Although data management is a crucial component of workflow management, services that provide each should not be too tightly linked. It should be possible to build a flow that contains as much data management as is required - but no more than is required. By separating the two services, workflow does not dictate that the same data management facilities be used for every flow. In some cases, UNIX-based utilities such as SCCS, RCS and make can provide an adequate level of data management. In other cases, a much more sophisticated level of data management, such as is provided by commercial data repositories and product data management (PDM) systems is required. This decision should be left to the flow developer, not the workflow system provider.

Flexible dependency management: A rich and robust dependency management system must control the workflow engine such that tool and data prerequisites and other conditions can be modeled and presented to the end-user via the flow in an acceptable manner. For example, certain events might trigger the availability of certain data or tasks for use by downstream steps in the flow ("start dependencies"). Other events might be used to insure that a task does not complete too soon ("finish dependencies"). Conditions must also be allowed that support controlled decisions regarding "When can I reset and rerun this step?", "Do I have the necessary permissions to execute this task?", etc.

Tools are integrated such that checks can be made on their data to determine flow state. File existence, date/time stamps, file contents and other means can be used to determine data maturity, and these maturity results can be use to control progress through the workflow. Data variables in the workflow can serve as proxies for one or more design data items, allowing information about the data state and/or value to be stored as metadata separate from the design data. Workflow procedures can be automatically triggered based on design data-related events that occur. Communication between the integrated tools and the workflow is performed via the workflow application procedural interface (API), which allows the tool to exchange (set/get) metadata (task state, data variable state and value) with the workflow.

Trigger-based procedures provide the ability to notify the user when something has changed in the design that does, or might, require them to rework some of their steps. Features that detect changes, notify downstream process steps, capture information about the change, and allow the user to determine the best course of action must be provided.

Each of the characteristics described previously contributes to the goal of creating an integrated, workflow-enabled design environment. This goal is achieved if the user of the workflow system becomes (and feels) more productive, while at the same time deriving the cost, quality and time-to-market benefits gained through following consistent design processes to produce successful designs.

6. Interoperability Research

This section describes research into a system level CAD software design process for developing truly interoperable CAD software that solves real user problems.

Past efforts at making CAD tools interoperate met with limited technical success because the interoperability problem was being approached from the bottom up. This approach was analogous to a microprocessor designer trying to design a state of the art processor starting with off the shelf components. Although the designer could probably find all the parts he needed, they would not work well together because they had not been designed from a system perspective

System design approach: The key to the successful design of a complete system-to-mask CAD software system is to treat the design as a system design problem. This section outlines an interoperability analysis methodology that combines object oriented software design with top down hardware design. There are three distinct parts of the methodology: system specification, system analysis, and system optimization. Detailed implementation follows from these steps and will not be addressed here.

System specification: In both hardware and software system design, the single most important step from the perspective of quality and timeliness of results is specification of the problem to be solved. This methodology uses a "user task" oriented way of specifying high level system requirements. The basic approach is to model the CAD user's design methodology as a set of well defined tasks. A task consists of a textual description of what work is performed, the set of inputs required in order to perform the task, and the set of outputs produced by the task. Note that tasks are defined in a tool independent way. So, an example task might be developing RTL level models. A task would NOT be, push button A on interface B to produce a netlist that is used by C. Tasks are defined for the entire design methodology to be supported by the CAD system. In our experience, we found that it takes approximately 200 tasks to describe a cell based design methodology that spans from product specification to final mask tapeout. During the task development process, it is important that task inputs and outputs be normalized. Normalization means that the fundamental information being consumed or produced is identified, rather than the file format which some tool may use to represent it.

Once tasks have been defined, the resulting information represents the major design creation, analysis, and validation steps as well as a complete information model of the user's design methodology in a tool independent fashion. Tasks are represented as nodes in a directed graph which are linked together through the specified inputs and outputs. Interestingly, task graphs more faithfully represent the designer's choices in what steps to do next at a given point in the design process. In contrast, the normal tool specific design flow descriptions do not represent what designers actually do, because they simplify the problem to one which is linear in nature.

After tasks have been specified, then a set of scenarios is defined. A scenario is a set of boundary conditions to be applied to the set of tasks previously defined. A scenario typically includes: end user profile (team size, experience, etc.), tools that must be used (already purchased or developed), and end user driving functions (product cost, size, performance, and technology to be used). Scenarios should represent the set of unique contexts in which the CAD system will used. The purpose of the scenarios is to prune the task graph, and reduce the number of interactions the tasks have with each other to a practical subset.

The combination of tasks and scenarios represents the system specification that the CAD system is designed to.

System analysis: The CAD system is now analyzed using real tools. The purpose of the system analysis phase is to evaluate data and control flow during the design process as defined by the tasks and scenarios. The first step in the analysis is to perform a task to tool mapping. During this step each scenario is analyzed with a specific set of tools. For example, a broad based CAD vendor may perform one analysis with only its tools and a second with key third party tools included. An internal CAD organization may perform two or three mappings on a combination of external and internal tools. The result of this step is a mapping of tools to tasks. Typically, this is the first point where holes and overlaps of functionality are identified.

When the task/tool map is complete, then tool models need to be developed for each tool. A tool model is similar in structure to the user task. It contains a description of the function, data inputs, data outputs, control inputs, and control outputs. Data input and output is classified into four parts, persistence, behavioral semantics, structural model, and namespace. Control is defined as a set of interfaces. This interface model is analogous to the software component models like Corba and Com.

Once models have been developed, then data flow and control flow diagrams are created for the entire task/tool map. These diagrams are then analyzed. In our experience, this analysis clearly identifies the classic interoperability problems (performance, name mapping, structure mapping, semantic interpretation errors, and tool control). This level of analysis is typically the most important for CAD organizations as they typically have to deal with tools as black boxes that cannot be optimized in and of themselves. The analysis results are captured and used as input for the system optimization step.

System optimization: In the system optimization step, the pieces of the system are modified to improve overall performance of the system. There are three ways of improving this performance. The first way is to repartition the boundaries of tools. This type of improvement can typically be performed only by CAD vendors or on internal tools. The idea is that by peeling back the tool's general purpose interface, there is typically a level where a lower overhead interchange of data and control can take place. The second type of improvement comes from improvements in data interoperability. In this case, analysis results will lead to things like internal naming conventions, bus usage conventions, etc. that improve the interpretation of data throughout the flow. The final type of improvement is through technological innovation. This is where new technologies (such as formal logic verification) replace a large number of tasks with a single task in the overall flow.

7. Conclusion

This tutorial paper has described interoperability problems in four classes of tools: schematic capture tools, simulators and synthesizers, IC design tools, and workflow management tools. Using the examples cited in this paper, a new CAD user can carefully examine his design process, the tools required to execute that design process, and the limitations and interactions of such tools. Current research may allow seamless interoperation of future tools. For now though, the authors hope that this paper has forewarned and forearmed new CAD users against the interoperability problems they will certainly face!

Acknowledgments

Ates Gurcan and Jim Sunde of Exar Corporation provided support and review for the section on schematic interoperability issues. Mike Murray thanks Rhonda and Steven Murray for their patience and understanding. All product or service names mentioned herein are trademarks of their respective owners.