1989

Carousel, a process management service to support analog integrated circuit design /

Teresa E. Krieger
Lehigh University

CAROUSEL

A PROCESS MANAGEMENT SERVICE

TO SUPPORT ANALOG INTEGRATED CIRCUIT DESIGN

by

Teresa E. Krieger

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

1988

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Dec 14, 1988

Date

Samuel L. Gulden

Professor Samuel L. Gulden

Lawrence J. Vanneni

Department Chairman

ACKNOWLEDGEMENTS

I would like to thank the management of AT&T Bell Laboratories in Reading, Pennsylvania for the financial and technical support that made this work possible. Aris A. Yiannoulos and Lew E. Miller were, in particular, supportive of the project. Aris, in addition, provided the initial specifications for the Carousel design service and continues to supervise the development of the service. Several of my co-workers deserve credit for the development of parts of the Carousel design service. L. Dreyer, P. Heiser, and R. Wolf were primarily responsible for development, population, and maintenance of the design library. C. Leveroni provided the panel interface. D. Diller and other co-workers mentioned above gave thoughtful input on parts of the system. Several Reading IC designers tested new releases of the service and provided input on future enhancements.

CONTENTS

LIST OF FIGURES

CAROUSEL

A PROCESS MANAGEMENT SERVICE

TO SUPPORT ANALOG INTEGRATED CIRCUIT DESIGN

by Teresa E. Krieger

ABSTRACT

Carousel is a design service developed to support analog integrated circuit (IC) design. The goals of Carousel are first, to reduce IC product development cost by reducing IC design time, and second, to support the "quality by design" philosophy. The Carousel design service meets these goals by creating a customized design environment in which the design process can further be automated. In this environment, Carousel provides fast automatic access to the correct and consistent support information and tools needed at several phases of the IC product development process. Design support information is selectively extracted from a centralized library of tool- and technology-specific functions, data, and utilities.

The Carousel design service is comprised of control code and a database. The database is a hierarchically structured collection of utilities and tool- and technology-specific data and functions needed for analog IC design. The control code provides a clean user view of the system and employs multi-level software switching to attach an appropriate library knowledge base to the current design session. The IC designer therefore, does not need to know where or how the information is stored, nor how to retrieve it.

1. INTRODUCTION

1.1 BACKGROUND

In the past fifteen years, the silicon industry has become highly competitive. Leading edge competitors promote the ability to develop increasingly complex, high quality integrated circuits (ICs) using state-of-the-art technologies while decreasing the design interval. Shorter design intervals in a competitive market assure early market penetration and can yield lower cost end products. Assuming quality has not been compromised by the shortened design interval, the timely introduction of a low cost end product can corner the market. Producing such a product is obviously the ultimate goal of every competitor. The two major goals in IC product development then, are short design intervals and high quality at low cost.

In an effort to secure a competitive position, companies are making great strides toward reducing design intervals and cutting the cost of products while improving quality. Carousel is an example and a component of the systems needed to further this cause.

1.2 OVERVIEW

Quality in IC design relies on quality design information. Design quality can be aided and design intervals cut, by providing fast automatic access to design tools and reliable design libraries. In this thesis, Carousel is described as a service to facilitate sharing design information and automating part of the design process. Following is a description of the problems addressed in building the Carousel

system. These problems and in-depth Carousel solutions are described in detail in this thesis.

The first problem was to define what information the users should share in the system. As a system to be used by a large base of designers using different design technologies, Carousel is implemented with an information base consisting of only the widely accepted technology-specific device and circuit data and commonly used functions and utilities; a small but critical subset of the design information available in the design universe. All information in Carousel is guaranteed to be supported.

The second problem was where and how to store information to allow it to be easily shared. In the Carousel system, design information is collected into a centralized database in the form of a hierarchical tree structure. Information is grouped in tree branches, first by design tool to which it applies, then by technology. Formal naming conventions were established to label branches in the tree structure. These conventions created a parallelism among the branches of the tree, giving order to the structure. This solution eased the task of assuring information integrity, including completeness and correctness. Library administration and distribution became easier, thereby effecting overall system integrity.

The third problem was tool and library access. How can the user easily access the information in the library for a given tool without knowing storage details? The Carousel solution provides control code which allows the user to simply specify the tool and technology he/she wishes to use. Using the parallelism in the tree structure, generic control code is used to access various parts of the library, given this

simple user input. Only design information applicable to the current design session is accessible.

The fourth problem was how to manage versions of library data and tools. In Carousel, version branches were incorporated into the database structure to provide an archival system. The Carousel control code allows the user to specify a library version, then justifies any incompatibilities between the chosen library version and the specified tool version and finally, accesses information in the appropriate version branch.

The fifth problem was how to automate parts of the design process. The Carousel system automatically links the correct tool-, version-, and technology-specific information needed by the designer for the current design session. The user need not know what information is available, what subset of that information he/she should use for the current design session, nor how to access that subset. The user's design environment is customized to fit the parameters of the current design.

The sixth and greatest problem addressed in the Carousel project was how to build a system to support variable numbers and conbinations of tools, tool versions, technologies, and information formats and versions. The goal was to develope generic code that, given minimal user input, could handle all reasonable combinations and could accomodate new combinations without code changes. In other words, the code must be data independent. Carousel achieves this goal through the use of multi-level software switching, in which the input data, when applied to the code, customizes the Carousel process.

Carousel is described here from the product development point of view for several reasons. First, this approach provides a mechanism for fully documenting the progression of Carousel development from conception to release and maintenance. Second, Carousel is a service comprised of several parts. Since the software code and the design library that comprise Carousel were developed concurrently and are critically interdependent, the descriptions of their respective development phases are interwoven in the text. The progressive development approach allows the relationships between the parts and the impact of the resultant system complexity to be better understood. Third, Carousel is a "living" service. It is not a once-and-done release. Given the general goals described in the requirements section, Carousel will be continually modified to further meet those goals. The design and implementation of the first release of the service can have a great impact on the future development of the system.

This thesis includes

- an examination of IC design and the IC design environment.
- identification of some of the user needs in that environment.
- specification of data management problems in that environment.
- a list of the goals and requirements set for the Carousel service.
- an overview of the design strategy used to develop Carousel.

- a full description of the solution to the environment needs as provided by Carousel.
- a full description of the library structure developed to support IC design and capture design methodology.
- a critical review of the Carousel solution.
- discussion of future enhancements and improvements.

2. THE DESIGN ENVIRONMENT

Parts of the analog IC design environment include hardware and operating systems, design tools, design libraries and design experts. The design libraries contain tool- and technology-specific information used by the experts at various stages in the IC design process. Through multi-level software switching, Carousel links all appropriate library information for the tool and technology specified by the design expert and customizes the user environment, thereby automating part of the design process. To fully understand the issue of data management and the Carousel solution in an IC design environment, one must first understand a little bit about the process of IC design and the environment in which it is pursued.

2.1 OPERATING ENVIRONMENT

Carousel was originally designed to run on the Digital Equipment Corporation (DEC) VAX 11/780 series computer running the Virtual Memory Operating System

(VMS). Most of the steps in the IC design process were once done on this machine. Support has been discontinued on the VMS machines and migrated to a series of systems running AT&T's UNIX®* operating system, on which the design tools now run. Since UNIX is the current operating environment for Carousel, only the UNIX implementation of the service will be described here. Supported hardware systems running some flavor of UNIX include DEC VAX 11/780 series running BSD 4.2, mainframe Amdahl running UTS, and SUN Microsystems workstations running SUN OS. Other machine types may be added to the list as the Carousel customer base increases. The only requirement is a UNIX-based operating system environment.

2.2 THE DESIGNER

The people involved in IC product development are typically experts in the physical sciences such as electrical engineering and physics. They are highly trained, highly skilled, and highly paid. They are the most valuable resource to the industry. Optimal use of their time and talent is therefore vital to IC development efficiency. To maximize the output of an IC designer means to maximize use of his/her creative design skills. The Carousel design service aids this charter by removing details of data management from the user and automating part of the design process.

---

* UNIX is a trademark of AT&T Bell Laboratories.

2.3 IC TECHNOLOGIES

Within the analog IC design environment, several technologies are used. They include the complementary bipolar IC (CBIC) technology, the complementary metal oxide semiconductor (CMOS) technology, and the dielectric isolation (DI) technology. Each of these three technologies has from three to six sub-technologies. Each sub-technology is denoted by an appended letter to the technology name. For example, one sub-technology of CBIC is CBICM. Another is CBICR. Listed below are the technologies used for analog IC design at AT&T Bell Laboratories.

- CBIC: CBICL, CBICM, CBICR, CBICS, CBICU, CBICV
- CMOS: CMOSA, CMOSB, CMOSC
- DI: DIA, DIB, DIC

In the rest of this paper, sub-technologies will be referred to simply as technologies.

Different technologies exhibit different speed, power, and size tradeoffs, but the specifics of the technologies are not important here. The important point here is that the characteristics and associated design methodology of each technology can be defined and stored in a database. That technology-specific information then needs to be retrieved each time an IC is designed using that technology.

3. IC DESIGN

The term "IC design" is used rather loosely in the silicon industry. It is used to refer to the IC product realization process, not just the design phase of realization. The actual design of IC's is only part of the IC product realization process although it is certainly a most critical part. The emphasis of the Carousel design service is on support for all phases of design. Hereafter, for the sake of simplicity, the IC product realization process will be referred to simply as the design process.

As noted earlier, a major contributor to the final cost of a product is design time. Engineering hours are usually the highest priced commodity in product design. It is therefore critical to streamline the design process as much as possible while maintaining and, ideally, improving product quality. To understand how the design process can be streamlined, it is necessary to understand the phases of IC design.

3.1 THE DESIGN PROCESS

The IC design process can be divided into several distinct steps. Figure 1 shows a process flow diagram delineating the various steps involved in analog IC design. Loops in the graph indicate an iterative process wherein parameters are fine-tuned in one phase and the changes must be reflected in a previous phase, either because of a data dependency or to force data consistency.

3.2 DESIGN TOOLS

Many powerful CAD/CAT software tools are used during the course of IC product development. There is generally one tool for each phase of development. For the AT&T application, Figure 2 lists the CAD/CAT tools associated with each phase of design shown in Figure 1.

The strength of the tools used at AT&T lies in the fact that most of them were developed by AT&T specifically for its own needs. Some of the tools are interactive graphics tools. Others are data processing tools which, given a set of input data, perform some function on that data and output either data files to interpret or be further processed, or a graphical representation of the processed data.

A characteristic of the tools is that they were developed for a large diverse customer base within AT&T. They are used for IC design, printed circuit board design, and system design. The needs and inherent methodologies of these three user bases are very different. IC design, furthermore, is comprised of digital design, analog design, and mixed digital and analog design. Because the methodologies of these three subdivisions are different, the need exists for a working environment tailored to smaller subsets of the user base. Carousel was specifically developed to present a working environment tailored to analog IC design.

4. WHAT WAS AVAILABLE

4.1 DESIGN LIBRARIES - EVOLUTION

Over the past five years, as design has increasingly come to depend on computer aids, libraries of design information have become extremely important to efficient IC design. Less than seven years ago, the need for such libraries did not even exist. Along with libraries, the need for efficient library administration techniques has grown. What the industry has experienced is as follows.

With the emergence of computer-aided design (CAD) and test (CAT) tools, came the need to create and save files to input to the tools. Creating and storing these files saved entering design data interactively each time a tool was used. A design file might typically contain parameters associated with a component (resistor, capacitor, transistor, etc.) used in a designer's circuit. Another file might contain a description of how several components could be connected to create a particular subcircuit. Yet another file might contain tool-specific procedures to simplify the use of that tool. Since designers working in the same technology and with the same design tools often needed some of the same components, subcircuits, and procedures for their designs, it became commonplace to pass files among designers. Several problems are inherent in this type of information passing.

When several designers maintain copies of the same files, computer storage is wasted. Designers who need a particular set of data may not know that the files they need already exist and therefore waste their time duplicating effort. If an error

is present in a given file, the erroneous file could be propagated to many designers and cause an epidemic of design errors. If an error is found in a file, there is no good way of ensuring that the correction is propagated to all violated files.

As the damage and inefficiency of this unchecked mode of operation became apparent, moves were made to establish centralized libraries of each type of design data. An administrator was assigned to each library and held the responsibility for collecting, checking, and storing design files.[1] Several libraries grew into trusted sources of information; one contained files of components, one contained files of subcircuits, one contained files of device models, and others contained files of setup parameters and procedures. Typically, each library contained data to be used with one specific design tool. As the demand for library data grew, library administrators were faced with the new problem of distributing libraries to more that one machine and maintaining consistency among all machines. Automatic procedures for propagating library changes were soon developed.

Library development for analog IC design needs had come a long way when this point was reached, but several problems still existed. First, to access a file in a given library, the user had to specify the full path to that file. The user therefore had to know the location and structure of the library and the naming conventions used for files in that library. A high-level user view to the database was needed to remove the user from storage details. Second, several library administrators were doing similar administrative tasks but using different and often inefficient methods. Again time was wasted on duplication of effort. One collective centralized library

base was needed. Third, each time a library was updated, whether to fix an error or to improve old information, the risk was run of leaving some designers "high and dry" by destroying the old data. If the designer had started a design using the old data and tried to use the same data again, incompatibilities often occurred between the old and the new data. To avoid this problem, designers began copying needed files from the library into their own environment at the onset of a new design. Consequently, some of the pre-library problems began to re-appear. Library version control needed to be introduced. In summary, a true data management system was needed.

The original charter of Carousel was to alleviate the first of these problems. Given the volume and complexity of design information being stored, a database system, not just a database, was needed. The designer needed to be protected from the details of data storage, access, and consistency.

4.2 DESIGN TOOL ENVIRONMENT - SYSCAD

Each design tool listed in Figure 2 is supported by a different group of tool developers. Until recently, the design tool developers released new versions of the design tools individually as updates and enhancements were made. Each version of each tool had a different name. So, to use a particular version of a tool, the user invoked the tool by using the name associated with that version. For example, for SCHEMA, the user could call SCH or NSCH (new SCHEMA). For ADVICE, the user could call ADVICE, or ADVICEX, or ADVICEXX.

Problems similar to those experienced with the previously mentioned library distributions were often encountered. Namely, incompatibilities existed between new and old versions of the same tool and between different versions of different tools. Since some tools must use the output of or provide the input to other tools, these inconsistencies could not be tolerated. Also, library data used in one tool version did not always play in new versions and the problem arose of keeping different versions of the libraries to match versions of the tools. Users had to keep track of many tool version names. As with design libraries, IC designers should not be required to track tool versions and justify incompatibilities.

In the past year, a new approach to design tool releases has been implemented. The newest versions of the design tools have been packaged into one structure called SYSCAD. The SYSCAD structure has three branches: one for current, fully tested and exercised versions of tools (called SYSCAD), one for new, fully tested but not extensively exercised versions of tools (called NSYSCAD), and one for experimental versions of some tools (called TSYSCAD). This packaged approach eased tracking and distribution.

The SYSCAD and NSYSCAD branches each contain one version of each design tool. Within each branch, all tools are guaranteed to be compatible. To decrease tool version change problems, a strict release strategy was implemented. Version changes are infrequent and are announced far in advance so users can adjust their tool calls appropriately.

To call a particular version of a tool or tool set, the user first specifies SYSCAD or NSYSCAD as the current tool environment. Specification of one of these environments triggers a setup routine which defines the environment. The environment, in turn, restricts tool access to the version associated with that environment. The name of any one tool is identical in either environment, so the user does not have to remember different tool names.

In summary, a disciplined strategy for tool releases has been adopted. And, since information in the design libraries must be closely coupled with the tools, a compatible strategy needed to be developed for library distributions. The customized design environment that Carousel provides is built on top of the design tool environment. It overlays the tool environment with application-specific reference libraries.

5. REQUIREMENTS

The previous sections described the IC design environment and some of the needs in that environment. The environment is constantly growing more complex with the increase in number of technologies and growing sophistication of and reliance upon CAD/CAT tools. Carousel is a means of managing the current complexity and hiding the effects of added complexity.

The requirements for Carousel define the goals of the service, specification of the functionality it is to provide, and priorities for the development of the service.

5.1 GOALS

As mentioned earlier, two essential goals in IC product development are short design intervals and quality at low cost. Carousel's purpose is to help attain these goals. To show how Carousel is helpful, a discussion of the goals themselves and how they can generally be achieved, is given in the following section.

5.1.1 DESIGN INTERVAL AND COST ISSUES

Design cost is a major component of final product cost. It must be contained without sacrificing quality. The introduction of computer aids specifically aims to achieve this, as it also aims to shorten design intervals.

How can design intervals and costs be reduced with computer aids? The largest component of development cost (in time and money), is engineering hours; high-priced hours spent training designers and at working at each phase of product realization, from product specification to product design and documentation. Following is a list of some of the time and money cost components which computer aids can affect.

- learning technology-specific design methods/constraints
- learning to use different types of computer facilities
- learning library structures and access methods

- gathering technology-specific data from other designers and libraries
- learning to use design tools
- moving and reformatting data to input to tools
- running CAD/CAT tools interactively
- writing tool-specific software to make tool usage easier
- interpreting output from tools
- correcting errors due to data inconsistency
- running design iterations through tools
- checking tool results

In what ways can integrated, effectively customized CAD solutions help?

- capture technology-specific design data and methodology in a library
- provide a clean, consistent interface to tools and libraries
- provide a centralized library of commonly used tool functions and design data
- automate steps in the design process where human interaction is not needed
- provide automatic access to tool and technology data in libraries

This list comprises a set of starting goals for the Carousel service. Carousel must provide for these opportunities and must further be designed to easily accommodate future improvements to provide for others.

5.1.2 QUALITY ISSUES

Unreliable design information or unreliable access to such information, undermines the integrity of a design. Design integrity translates into product integrity, which is a major component of product quality. Carousel will make a direct contribution to the commitment to product quality by safeguarding the integrity of information used for design.

Carousel will also make indirect contributions to product quality. A design which requires iteration because the design information used proves unreliable, is a costly design, and a design that will take extra time to complete. If the need for iterations due to corrupt design data cannot arise, quality can be delivered at a lower cost.

In the same spirit, an indirect contribution is made by lowering the cost of "quality by design." Producing quality products requires ongoing effort. An integral part of every good product development project, whether it be software or hardware, is quality control. Recently, greater emphasis has been put on enforcing quality at all phases of product design because production of a high-quality product whose quality is not controlled beginning at the design phase, is a coincidence; production of a high-quality product whose quality is controlled beginning at the design phase is a sure bet. This is the essence of the new "quality by design" philosophy for success.

"Quality by design" requires extra effort during the design process and can multiply the cost of design by a large factor. The efficiencies gained through the design cycle by using the Carousel service can have a significant impact on the cost of this additive effort.

5.2 SPECIFICATIONS

Given the goals of reducing IC product development time and maximizing product quality, specifications for the Carousel design service were defined.

5.2.1 DESIGN LIBRARY

The design libraries are intended to capture technology-specific design methodology, commonly used tool-specific functions, tool- and technology-specific data, utilities, and various representations of design information. The design libraries must contain complete, consistent, and correct design information at all times. Versions of library data must exist to support new and old tool versions and to support new designs and designs already in progress. Library management must be installed to provide security, fast, accurate updates, selective library distribution, and to impose standards on library contents and structure. New library information must be easily added to the libraries without structure or code changes.

5.2.2 DESIGN SERVICE

The Carousel design service must create a customized design environment for analog IC design. The environment must be built upon, and always be compatible

with, the design tool environment. It should encompass the design libraries, design tools and special-purpose utilities to aid the design process. It must exhibit an open architecture wherein support for new tools, utilities, and technologies can be added to the system without extensive code changes or revised data structures. Additions of code and data are obviously unavoidable. The service must function quickly and efficiently.

Functionally, Carousel should automate steps in the design process wherever possible. This includes providing fast automatic access to library information and trimming the design environment to protect the user from version conflicts. All information accessed must be version-consistent. At the same time, it must not decrease the power of the design tools, nor replace the functionality provided by them or by the SYSCAD environment. Carousel is an additive service.

The design service must be portable between UNIX systems. It should therefore contain no hard-coded paths or other machine-specific code.

5.2.2.1 LIBRARY ACCESS

Fast, efficient, automatic access must be provided to the design libraries. What is automatically accessed should be the subset of the database that is specific to the current design process. The accessed subset of the database should be all data that is needed for the current design phase, but not more. The data must match the tool version and technology specified by the user. Library access should be simplified by the standards imposed on library structure. Access routines should relieve the

user from knowing the details of library structure and naming conventions. Finally, the access routines should be independent of the data being accessed.

5.2.2.2 USER INTERFACE

The Carousel design service must provide a clean, user-friendly interface that gives a clear view of the system. It should provide a single point of reference to design files in the libraries and to design tools. In the interface, machine and storage details should be invisible and the view of design tools should be consistent. The interface should allow batch or interactive tool execution.

5.3 PRIORITIES

Priorities for development of the Carousel service were defined as follows. Top priority for Carousel development should be given to imposing standards on the design libraries to achieve data integrity, consistency, and maintainability, and to developing a library autolinking service, first for SCHEMA, then for ADVICE. The library standards would provide a basis for future development of the service. Autolinking routines automate the process of tying design library data to the design tool session.

The first release of Carousel should be implemented as quickly and efficiently as possible to relieve users of data access problems, especially since the library structures will be changing as the Carousel service evolves.

6. DESIGN

The design of the Carousel service is the "how to" realization of the requirements set for the service. It is the creative phase of development where many decisions must be made that are critical to the future of the service.

6.1 CAROUSEL - THE NAME

Carousel is so named because conceptually, the design service resembles a nested series of carousels or wheels. Figure 3 shows the nested levels of environment specification or wheels. When a user accesses Carousel and specifies a tool and technology to be used, Carousel "rotates" the various wheels to match up the appropriate environment definitions to create a customized design environment.

6.2 PARTS OF THE SYSTEM: VIEWS

Carousel, like every database system, is designed to have three layers or views; an internal layer, an external layer, and a conceptual layer. See Figure 4.[2] The lowest level layer is the physical layer or internal view. It represents the way data in the database is physically stored. The external view of the system is the user view. This is the view that the user sees when accessing the system. The conceptual level or view is the level of indirection or mapping between the internal and external views.

Database system views allow a complex database system to appear simple to the user. For Carousel, the internal view consists of the design library and design

tools. This view is complicated by tool and library versions and the interdependencies between them. The external view for Carousel is the user interface to the system. It provides a single point of reference to the complex webs of data in the internal view. The conceptual level of Carousel is the code which maps the external view to the internal view. This code is referred to as the control code.

6.2.1 USER INTERFACE

The user view of Carousel in the form of the user interface, is critical to the success of the design service. If the interface is too complex, unclear, or cumbersome, users simply will not use the service. Studies in Human Factors (Human Engineering) have shown several simple but important points.

- human beings can only absorb and process limited amounts of visual data at one time
- levels of tension and frustration increase when too much data is presented visually
- human beings identify more quickly and easily with pictures (shapes and colors) than with words

These hints are useful in developing a friendly user interface. For Carousel, the user interface must allow the user to present as little information as possible in as simple form as possible to specify what he/she wishes to do with the service.

6.2.2 CONTROL CODE

At the conceptual level of Carousel, the control code is the heart and brain of the system. It has several functions. Generally, it must create a customized design environment for the user. Specifically, it must

- present the user interface
- parse the user input to check for completeness
- evaluate the input to determine correctness
- classify input to determine type and level of support
- determine tool version and match to library version
- set up the environment by creating any needed environment variables or data or control files
- match and feed data to tools or users
- invoke tools
- clean up after control returns from tools
- exit or allow further processing
- provide system help if requested

- respond to errors intelligently

Figure 5 shows a flow diagram for the control code.

6.3 DESIGN LIBRARY

Early in the design of Carousel, it became evident that the design libraries would have to be restructured to clean up the internal system view and meet the goals of the service. Given the unique structure of each library, no generic access or management routines could be employed. The SCHEMA and ADVICE libraries were the first to be revised since support for these two tools was a priority item.

The best approach to library management required packing the libraries into one structure and setting standards for that structure so that other libraries could be integrated easily. The single structure was given the name REFLIB, for reference library. A single library manager was assigned to coordinate restructure, updates, and distribution.

6.4 OPEN ARCHITECTURE

To fulfill the open architecture requirement on Carousel, the control code and the design library (REFLIB) were designed on the basis of modularity and levels of specification. Individual modules consist of function-specific code or a specific class of information. The highest level of specification is the most general. The levels go from general to specific both as one traverses the design library structure, and as one follows the process flow of the control code. Figure 6 shows the levels

of specification for the library and for the control code. Since modular code and structures are employed, levels are distinct and definable.

At every level, the data dependencies are easily distinguished. In the highest level modules, very few data dependencies exist. These modules have the broadest scope. Therefore, code and structure changes at that level are very infrequent. At the lowest levels, several distinct data dependencies exist. The code and data at this level are dependent upon tool version, library version, and technology. Updates are most frequent at this level because this is where the most raw form of data resides.

The Carousel control code also supports the open architecture goal by checking for the existence of directories and data in the system rather than maintaining a list of what is available and checking against the list. Therefore, new code and structures can be added to the system without code changes.

6.5 THE DATA MODEL

REFLIB was designed to fit the hierarchical data model using the UNIX tree structure. The hierarchical data model was chosen for the implementation of REFLIB for several reasons.

First, the hierarchical data model best suits the levels of specification strategy defined for the design service. High levels in the REFLIB UNIX tree structure are the most general; low levels are the most specific. This is clear when each level in the tree structure is thought of as having a data dependency on the level above it.

Second, the individual libraries that were collected to form REFLIB, were already in the form of UNIX tree structures. It was therefore easy to patch the libraries together into a single structure by simply adding a new common level of hierarchy at the root of each library. Consequently, using the structures available was helpful in meeting the fast implementation goal set for Carousel.

6.6 VERSION CONTROL

One of the greatest problems in design library management is version control. The most up-to-date information must be available in the library immediately so that new designs will be developed using the best data and procedures available. Yet, this "fast install" approach poses some serious problems. In many cases, the old library data is not incorrect, but has simply been fine-tuned. The new data can not be substituted for the old because designs in progress may be using the old data and a change would cause havoc when running the design tools. It is therefore necessary to somehow maintain several versions of data when substitutions can not be made. The following questions arise. How many versions should be kept? How long should they be kept? How can they be accessed? How can designers be encouraged to use the new when they can access the old?

In the design of Carousel, these questions were carefully considered. No ideal solution has been found, but an attempt has been made to at least reduce the version conflict problem.

A strategy similar to the SYSCAD tool version approach has been adopted. Three branches were added to the design library tree structure under the tool level. Each branch contains one version of the library for that tool. The branches are c (current), n(new), and o (old) (and in some cases, x for experimental). The current and old branches are relatively static; new and experimental are constantly changing.

The current branch of the library contains data that is tested and correct but current only up to a particular date. It may only be updated when errors are found or data can be added that does not conflict with the other data in that branch. When a "version release" occurs, the current data is propagated to the old branch and is replaced by the data in the new branch.

The old branch contains the data that was last in the current branch. It will never be updated, only replaced by the data in the current branch when a version release occurs. The pre-release old branch data is lost when a version release occurs.

The new branch contains the contents of the current branch plus any new data plus any changes made to the current data. Only tested and approved data is released to the new branch. Using the new branch is somewhat risky because the data in it could change at any unannounced time. Users who want the most up-to-date but tested data will use the new branch. When a version release occurs, a copy of the new branch replaces the current branch, but the new branch remains the same.

The experimental branch is used to store versions of data that are approved but not fully tested. Users who want the most up-to-date data can use the experimental branch at their own risk. The data in the experimental branch can change at any time.

6.6.1 LIBRARY VERSION RELEASES

When a library version release occurs, the new branch becomes the current, the current becomes the old and the old is lost. This propagation should happen as infrequently as possible. When and why a version release occurs are extremely critical to Carousel system integrity.

There are two primary reasons for a version release. First, if a SYSCAD version release occurs and library data is matched to tool versions, either a library version release must be issued or Carousel must match the library version to the tool version in software. Carousel does this matching in the tool switches. Second, when the current library data starts to become outdated and most users are using the new data, a version release should be issued.

When a release is to occur, users must be notified well in advance. Although Carousel matches the library version to the tool version chosen by the user, some users still need to access the library directly. In this case, they need to know which version to access.

6.7 DISTRIBUTED DATABASE

The Carousel design service must be available on several different machines. The distributed database approach was adopted for several reasons. First, this method of data access was the fastest and easiest to implement. No special-purpose networking routines are required and access routines on a local machine are simple and efficient. The only data transfer activities required are updates to the system. Second, access to data on a local machine is faster than through network links. Third, it is easy to restore a corrupted machine by copying data from another machine. No formal archival system is necessary to ensure system integrity. Fourth, network integrity is not heavily relied upon. If the network fails, the system still can operate. Fifth, unreliable data transfer across network links is only a concern for updates, not the frequent transactions of everyday users.

6.8 SYSTEM INDEPENDENCE

Carousel was designed for system independence to meet the portability requirements. In Carousel, system independence is achieved by using relative paths and the UNIX "logdir" function. Since all code and files in UNIX must be referenced using the path name to their location or a symbol equivalent to that pathname, a method was needed for specifying the hardware-dependent root of any path. The "logdir" facility can be used to find the root-level name of any directory. Therefore, in the Carousel control code and design library, when a path needs to be defined or referenced, "logdir" is used. For example when paths to design library data for ADVICE are created, "logdir" and relative path names are used as

"path=‘logdir reflib‘/aref". In this assignment, ‘logdir reflib‘ is evaluated and replaced with the physical location of REFLIB before the path variable is instantiated.

6.9 QUALITY

The quality requirements on Carousel demand that not only must the service meet its functional requirements, but it must also meet performance requirements. To meet the quality requirements on Carousel, good programming techniques were used in the design of the service.

6.9.1 SCOPING

In Carousel, minimal scoping is enforced. Scope is a measure of the life span of a variable (or procedure). It indicates where (in which processes, subroutines, functions, etc.) a variable can be used. In a clean programming application, a variable is available for use only where it is needed. This is called minimal scoping.

At every level of specification in Carousel, variables are permitted to be live and active only if they are needed. In other words, variable scope is kept as close to the problem space as possible. This practice keeps the environment clean and decreases the chances of naming conflicts.

In the UNIX environment, this is done as follows. The user starts a program by calling it. That call starts a UNIX process. Every routine or program called by that process becomes a sub-process. Thus, a hierarchy of parent/child processes is

established. To set up a variable in a process, the user simply initializes it (i.e. var=x). Once a variable is initialized in a process, it can be used in that process. There are two ways to make a variable that is initialized in a process, live in a sub-process.

The first way is to export the variable from the process using the UNIX "export" command (i.e. export var). An exported variable is known to all subprocesses of the exporting process. It can then be referenced and modified by the subprocess but any modification is known only to the subprocess, not the parent process.

The second way to make a variable known to a subprocess is to make the subprocess part of the parent process by calling the subprocess using the ". subprocessname" UNIX call syntax. UNIX treats this type of call as if the subprocess code were installed directly into the process code. There is essentially one process running. Therefore, any changes made to the variable in the subprocess code, are seen by the parent process. This is the only way to make a variable initialized or modified by a subprocess, known to the parent process. Naturally, variables needed only by a subprocess are initialized in that subprocess and thereby have a scope of only that subprocess.

6.9.2 EFFICIENT EXECUTION

Another programming technique designed into the Carousel control code to achieve speed and efficiency, is minimal evaluation. With this strategy, if two or more choices exist at any decision point in the code, the most likely choice is placed first

in the code. Thus, chances are, only the first choice will ever be evaluated, thereby saving execution time.

6.9.3 CLEAN ENVIRONMENT

Another "quality by design" feature of Carousel involves environment order. Several steps in the Carousel process create temporary files in the user's directory. They are created in the user's directory for two reasons. First, since the running Carousel process is owned by the user, it can create and delete files from the current directory (user's directory) without concern for directory permissions and the user can always access files in his/her own directory with no problem. Second, since many users could be using Carousel at the same time, creating files in a centralized location would cause file naming conflicts and require sophisticated code to generate unique file names. The cost of this code would be too great.

After the design tool or the user uses the temporary files, they can be discarded. Carousel is designed to remember the names of all the files it creates by building a list of the names. The last thing the control code does before exiting is clean up the user's environment by deleting the files in the list.

7. IMPLEMENTATION

7.1 DESIGN LIBRARY

As noted in the design section, the libraries for SCHEMA and ADVICE were combined into one library, REFLIB, and a "levels of specification" strategy was adopted

for the structure. Naming standards were also imposed on the levels to make path names to every file more meaningful. These standards are used by the control code for level switching and information location. Figure 7 shows the general structure of REFLIB.

Note that the tool-level directory for the SCHEMA branch was named SREF (for SCHEMA reference) and the tool-level directory for ADVICE was named AREF (for ADVICE reference). Below the tool level, the SCHEMA and ADVICE directory structures are identical down to the sub-technology level. Below the sub-technology level, the branches become unique. Data at that level has dependencies on tool, library version, technology, and sub-technology.

As seen in Figure 7, at most levels of specification of the REFLIB structure, a subdirectory named "com" (for common) was added. The "com" subdirectories were designed to store any information that is common to all other members of that specification level. For example, any data that is common to all technologies in the old branch of SREF, should be stored in the "com" directory at the technology level. This structure provides a means of sharing information among several library branches rather than duplicating it needlessly. As with variable scoping, library information should be available as close to its problem space as possible.

Another type of subdirectory found in the REFLIB structure is the "bin" directory. Each bin directory holds executable code that is applicable to the level of specification at which it is stored. For example, the top-level Carousel control code is stored in the bin directory at the tool level because it is applicable to all tools in

REFLIB. The tool switch for each tool in the library is stored in the bin directory at the version level because it is used for all versions of that tool.

The structure and naming standards applied to REFLIB have given logic and order to the design library. Even if automatic procedures were not available for library access, users would find it easier to locate data in REFLIB than in the old libraries.

7.2 USER INTERFACE

The user view of the Carousel design service takes one of two forms. One is in the form of a command-line invocation and the other takes the form of a SUN panel.

7.2.1 COMMAND-LINE VIEW

The command-line view of Carousel was designed to

- require a minimal number of user keystrokes
- maintain tool call syntax
- allow batch job submittal
- maintain normal UNIX option specification syntax

To invoke the Carousel service via the command-line view, the user simply uses the syntax "car toolname [tool options] -x technology", where tool options are specified only if the user wants to use them or if the tool requires them.

Note that if SYSCAD design tools are needed, the user types SYSCAD or NSYSCAD followed by a carriage return, then issues the Carousel invocation, or the user can type SYSCAD or NSYSCAD followed by the Carousel invocation all on one command line. For example, to use Carousel and the NSYSCAD version of ADVICE with CBICM, the user types "nsyscad car advice t=4014 -x cbicm". The benefit of allowing all specifications on one command line with no further user interaction, is that the tool session can be run in batch mode. The syntax "advice t=4014" is a normal ADVICE tool call. Carousel requires the "-x technologyname" part to define the design environment and link the appropriate design library information.

Versions other than the current (default) version of design library data can be chosen by specifying n, o, or x in front of "technologyname" on the command line. For example, specifying "-x ncbicm" will provide access to the CBICM data in the new branch of the ADVICE directory. Note that if the library version chosen is not available or is incompatible with the tool version chosen, the Carousel control code will select an appropriate library version instead.

The command-line view also allows the user to obtain online help by typing "car" followed by a carriage return or the letter h. In response, Carousel will list on the standard output, the tools supported by Carousel, the syntax for calling them through Carousel, example tool calls, the technologies supported in the design library, and syntax for specifying library versions. Figure 8 shows the help output that the user sees.

If the user calls Carousel with an unsupported tool, Carousel will tell the user that the tool is unsupported, then show the help listing and ask for new input. If the user fails to specify a technology, Carousel indicates that one is required and prompts for input.

When Carousel has acquired all needed input data from the user, the user will see a list of files created by Carousel to be used in the tool (ADVICE only), followed by the start of the requested tool session. When the user exits the tool session, Carousel exits silently and the user will see the system prompt.

7.2.2 PANEL INTERFACE

The panel interface is available only on SUN workstations. It was designed to

- give the user a pictorial view of his/her choices
- use to advantage, the sophisticated capabilities of SUNVIEW software and windows on SUN workstations.
- replace multiple keystrokes with singular key clicks (using mouse)
- allow multiple tool instances simultaneously

To invoke the Carousel service via the panel interface, several conditions must exist. First, the user must be logged directly onto the SUN system where Carousel is to run. Second, the user must be running SUNTOOLS software. Third, the user must type "car" followed by a carriage return (no other specification). Figure 9(a)

shows what the user will see when the panel interface is invoked.

Three types of fields are available on the panel; choice fields, where the user can scroll through the possible choices, keyboard fields, where the user can type in the desired specifications, and button fields, where the user can trigger the indicated action. Figure 9(b) shows the choices available at each choice field. To call a tool through the panel interface, the user fills in the panel accordingly and clicks the execute button. The user will see a list of files created by Carousel to be used in the tool (ADVICE only), followed by the start of the requested tool session. When the user exits the tool session, he/she is returned to the panel to choose another tool or exit.

7.3 CONTROL CODE

The Carousel control code is the software realization of the design strategies specified in the design section. It is written in the UNIX SHELL language, which offers all of the basic constructs available in a high-level language.

The basic functions of the control code were shown in Figure 5. As the control code executes, it incrementally builds the customized design environment by defining environment variables which delimit the scope of the design session. With the scope defined, access to the appropriate subset of the design library information is possible.

The key to the control code execution is multi-level software switching. This technique allows the use of generic code at the highest levels of execution and more

specific code at the lowest levels. All control code levels perform a "match and feed" function on their input data. The match process determines if data is valid and compatible with the chosen design environment and rectifies any inconsistencies. The environment is defined by the user working environment and any previous Carousel environment definitions. The feed process passes the filtered and rectified data to the next step in the process, determining where to feed by using environment definitions. The fed data, in turn, makes each Carousel task application specific. The benefit of this approach is that the more generic levels of code can be shared by many applications. As new applications are added to the system, only small pieces of low-level application-specific code need to be added to the system. In the case of new supported technologies, no code at all needs to be added! This level partitioning also saves storage space by eliminating duplicate procedures that are customized to the application. Instead, the generic procedures are customized to the application "on the fly" by software switching.

Software switching is implemented in Carousel by progressively collecting environment variable assignments for generic variable names, then applying those assignments to achieve an application-specific action. For example, environment variables are used to build pathes to the tool switches and to library files and directories.

In the top-level Carousel control code, the user input is first parsed to determine which tool has been chosen. If, for example, the tool specified by the user is SCHEMA, the variable TOOL is assigned the value SREF. Note that SREF is the name of the branch of the library where all SCHEMA information is stored. Next,

the control code determines which technology and library version have been chosen. If the technology specified is NCBICM, variables GENTECH=CBIC, SPECTECH=M, and TECHVERS=N, are defined. These four assignments are made in the top-level control code because they apply to every Carousel application. Next, Carousel calls the tool-specific switch using the TOOL variable. The path to the switch for every tool in the library is 'logdir reflib'/$TOOL/bin/$TOOL.switch. In this case, the switch will be /usr3/reflib/sref/bin/sref.switch. In the tool switches, the TOOL, GENTECH, SPECTECH, and TECHVERS variables are used to define pathes to library files used by the tool.

The best way to fully describe the execution of the control code, is through pseudocode. Figures 10 and 11 list pseudocode descriptive of the control code flow.

7.4 LIBRARY LINKING

Library linking was the first part of the Carousel service to be implemented. For SCHEMA and ADVICE, a well-defined set of data from the associated library directories needs to be accessed each time the tools are used. It therefore made sense to automate the process.

7.4.1 SCHEMA LINKING

SCHEMA is a graphics tool which allows the designer to capture design intent by building a schematic and descriptive files of the schematic (to be fed to other tools for further processing). Several types of information are available in the SCHEMA

branch of the design library to aid in schematic capture. Some of these include SCHEMA functions to aid tool use, standardized symbols to graphically represent devices, and files to be used for processing in other tools. The information in the library files and directories can be made available for access within SCHEMA by using the SCHEMA command "link group directoryname rea", for directories and "link func pathtofunctionfile", for function files.

To automate this process, a file with the SCHEMA syntax and the paths to all needed files and directories is built by the SCHEMA switch and stored in the user's directory. Figure 12 shows an example of a file built by the SCHEMA switch for CBICM. The file is given the name of the technology (i.e. CBICM). SCHEMA is then invoked using the "-x technologyname" option. This option specifies to SCHEMA that the file, "technologyname", is to be executed in SCHEMA. SCHEMA tells the user which functions and directories have been linked. The effect is the same as if the user had typed the series of "link" commands interactively within SCHEMA.

7.4.2 ADVICE LINKING

ADVICE is a circuit simulation tool which allows device-level simulations of the electrical characteristics of circuits. In the ADVICE branch of the design library, device models and case files are available to use in simulations. The case files specify a list of files with variations on the parametric data applied to the models. Figure 13(a) shows a sample case file for CMOSA. In their stored form, the case files are not complete. One requirement of Carousel is that no hard-coded paths be

stored or referenced. The Carousel ADVICE switch therefore, builds complete case files using the stored case files and the "logdir" facility. One completed case file for CMOSA is shown in Figure 13(b).

The Carousel ADVICE switch builds all case files for the user-specified technology in the user's directory and writes to standard output, the names of the files built. File names are "co, c1, ..., cx" for CBIC and "cc0, cc1, ..., ccx" and "ac0, ac1, ..., acx" for CMOS and DI, where x is the number of case files in the library. Carousel then invokes ADVICE. To use the case files, the user specifies the ADVICE syntax ".lib lib1=cx" (or ccx or acx), where x is the case file number desired. When this command is issued, ADVICE makes cx one of it's internal library files. The data in the file can then be used for simulation.

8. SUPPORT

Like any other software product, Carousel is only as good as its support. The Carousel administrators are committed to ensuring the integrity of the service on all supported machines. Carousel support takes several forms.

- Alpha and Beta testing with new releases
- bug report logging and fast fix on a priority basis
- distribution of updates and new releases

- solicitation and logging of feedback on current system performance and new features
- counseling on system use
- system enhancement

Several individuals are responsible for different aspects of the Carousel system support. Each has a well defined problem space in which he/she takes ownership of support. The support team members also work closely together to find the cause of problems and to coordinate solutions that best benefit the overall system performance.

Each support team member is an expert in the area for which he/she is responsible. Team members include a Carousel administrator, a librarian, an interface developer, and a tool expert for each supported tool.

8.1 TESTING

The integrity of the Carousel design service depends heavily upon pre-release testing. Two testing phases are exercised before Carousel software is released: Alpha testing and Beta testing.

8.1.1 ALPHA TESTING

Alpha testing is testing done by the software developer to ensure that all specifications set for the software are met in the end product. This means that the

software must not only perform the function for which it was designed, but must also meet quality, speed, and other performance requirements defined at the onset of the project.

For Carousel, the functional requirement specified that the user must be given a customized design environment for his/her current design session. This means that the user must see the appropriate user interface for the hardware he/she is using, proper environment variables must be set up to provide access to the correct utilities, tools, and data, and the correct knowledge base from the design library must be automatically linked. These items can easily be checked.

The speed requirements for the Carousel design service are not rigid. They simply require that the service perform its actions in a "reasonable" amount of time: an amount that is not annoying to the user. Obviously, meeting the speed requirement demands a judgement call.

The quality requirement on Carousel specifies that not only should Carousel perform the given function, but it must do so well by using smart and accepted Computer Science technics. Carousel meets this specification by for example, cleaning up after itself (removing temporary files), using minimal scoping, responding to errors intelligently, and, in general, presenting a user friendly system view to the user.

To fully Alpha test a version of Carousel, every possible combination of hardware, tool version, technology version, and call option would have to be run. Since there

are several hundreds of combinations of these variables, it would be unrealistic to try to test them all. Depending upon the nature of the change to the Alpha version, a minimal set of tests can be defined to approximately span the subset of the combination space that might have been affected by the change. If for example, only the SCHEMA tool switch has been modified, the other design tool switches do not need to be tested.

In summary, the Alpha tester must understand the entire problem space and be able to map software changes onto that space efficiently and run tests accordingly.

8.1.2 BETA TESTING

Beta testing is software testing done by actual design service customers. The benefit of Beta testing is great. Carousel users are apt to try to use the design service in ways that were not considered by the developers. They also provide feedback on the performance of the service: not only on whether or not it works, but also on whether or not it is what they really need and how it can be further developed to service them better. Many Carousel revisions have been the outcome of Beta testing feedback.

8.2 DESIGN SERVICE CUSTOMERS

Several types of customers have been identified for the Carousel design service. Each has different needs, different support priority, and requires different procedures for receiving library distributions. The person distributing the libraries is responsible for knowing the customer class of a customer and distributing libraries

appropriately.

Type one customers are users in the design laboratory where Carousel was developed. They work on UTS machines and SUN workstations and have top priority in receiving releases. They receive service automatically. They should always have access to the maximum capabilities of Carousel. In particular, they should be able to access all tool versions, all library versions, and all libraries. They should receive continuous service.

Type two customers are internal AT&T system-level customers, OEM strategic partners, and AT&T design centers. They typically have SYSCAD tools available on their machines. They should receive only the new or the current version of the design library. They are to receive update support for one year on a contract basis. They must request service and provide a specific development case number to obtain support.

Type three customers are all other OEM customers. They usually need to receive design tools as well as support libraries. They should receive only the new or current version of the design library. They should receive only the tools and portions of the library licensed to them. They must request service, provide a development case number, and specify a date for termination of service to receive service. Updates will be provided only up to that date.

8.3 DISTRIBUTION

Carousel supports the distributed database approach to data access. Therefore, every machine where Carousel is to provide service, must have all needed parts of the database resident on that machine. Since the Carousel control and interface code must also be locally resident, they have been stored in the design database itself. This provides several advantages. Namely, all Carousel-related code and data is contained in one tree structure (directory) on any given machine. It is therefore easy to find all system parts. Updates are easier because the structure is identical on every machine. Distribution is easier because the structure can be bundled, shipped, and unbundled as one unit. Given these common attributes, highly automated procedures can be used to update and distribute the database. Finally, since the structure is controlled by the Carousel administrators, and not the hardware system administrators, the relationships between code and data locations are known and constant. These relationships are used to advantage in the Carousel environment definition and tool switches.

The person responsible for library distribution has several duties. When a request for service is received, he/she must follow a very strict procedure to determine the variables for distribution. First, he/she must determine if the request should be granted. There are formal procedures for making a request which include obtaining proper approval for service. These procedures ensure that proprietary information is not released to unapproved sites.

Second, if service is to be granted, the person doing distribution must evaluate and classify the target machine by answering the following questions:

- Which type of customer has made the request?
- Which tool supports are they requesting?
- Given the answers to the first two questions, which parts of the design service should the customer receive and for how long?
- Are the required design tools available on the machine?
- How must the service be sent? (i.e. via network links, on tape, etc.)

When the distribution variables have been defined, the distributor must set up the operating environment for Carousel on the target machine. This typically means obtaining a computer account from the system administrator, ensuring that sufficient disc space for the database is available, and setting up any global symbols needed by Carousel (such as "car").

Next, the person doing distribution packages the appropriate parts of the service and sends them to the customer for installation. Once this has been done, the distributor has responsibility for keeping track of the status of the Carousel account, making updates when needed (and approved), and terminating service if scheduled.

8.4 DATABASE ARCHIVAL

As with any database system, some form of data archival is critical to the integrity of the system. Archival is critical, first, to facilitate version control, and second, to prevent information loss with system failure. Version control is discussed in Section 6.6. Formal database archival procedures have not been installed for Carousel because it has been implemented as a distributed database system. One advantage of using a distributed database is that if the data on one hardware system is lost or violated, it can be restored from one of the other systems in the supported group. Therefore, integrity is not at stake.

9. CAROUSEL EVALUATION

Although the Carousel design service conception is only two years old, it has succeeded in achieving its goals of reducing the design interval and maintaining product quality. The areas of design library management and automatically linking library data to design sessions have been especially successful. The design library is now a very reliable source of information that is easy to access.

Some of the strategies used in the development of the service emphasized fast, easy implementation, rather than optimal efficiency. Now that the service has been installed and used, those same strategies can be reviewed and improved upon. Following is an evaluation of Carousel; a description of the good points, the bad points, the insufficiencies, and the inefficiencies. Future improvements and enhancements slated for the service are also described.

9.1 ADVANTAGES

Carousel provides several advantages over pre-Carousel methods of tool and data access. It provides a single point of reference to the design tools and the design library. The library is a much cleaner, more reliable source of information than before. Users no longer need to know library structure and naming conventions to access library data. They also save time by typing fewer keystrokes since no full path names need to be specified. The data that is accessed by the service is only as much as the user needs, and no more. The user therefore does not have to sort out which data applies to his/her design process. Versions of library data are now supported and matched with tool versions automatically by Carousel. The interface is clean and simple, both in the command-line form and the panel form. The system provides an open architecture so other tools and libraries can easily be incorporated into the support group with few or no code changes. The foundation for a sophisticated design automation tool has thus been developed.

9.2 DISADVANTAGES

The current implementation of the Carousel design service has several disadvantages. Carousel allows specification of only one technology in the design environment. In some few cases, designers need to access data for more than one technology. Carousel should allow a larger scope of information in the design environment. Since the SCHEMA and ADVICE switches build files in the user's current directory, the danger exists of naming conflicts. Carousel does not overwrite their files, but if a conflict exists, no linking is done. The SCHEMA switch unlinks all

linked files and directories before it links new ones to avoid conflicts. In some cases, users have files linked that they do not want unlinked. Carousel should unlink selectively. Carousel spawns several subprocesses as it executes. The requested design tool is in fact run as a subprocess. This is not the most efficient way to execute tools. The Carousel control code is dependent on the structure of the library. The panel interface is only available on the SUN workstations. A comparable interface should exist on the other systems as well.

9.3 ENHANCEMENTS

Several system improvements are slated for Carousel in the near future. The GRED (graphics editor) design tool will soon be supported in Carousel. For ADVICE, some users need to create customized case files rather than using the ones in the library. The ADVICE switch will be improved to give this flexibility. The SCHEMA switch must be altered to allow more that one technology. This will broaden the environment scope. It must also unlink files selectively. Facilities for building a panel user interface on systems other than SUN are now available. The interfaces should appear the same on all systems. A bug reporting facility will be installed to log problems. This system encourages follow-through procedures when a problem is reported. Some method of improving version control needs to be developed. More than three versions should be supported to satisfy user needs.

10. SUMMARY

Carousel is a process management service designed to improve the productivity of the analog IC design process. It has successfully reduced the design interval and helped to improve product quality and lower its cost.

In this thesis, the environment, goals, development stages and evaluation of the service were discussed. Carousel was described from the product development point of view to document clearly why the project was undertaken and how it developed from definition of a design environment need to installation of a clean, useful design product.

The first releases of Carousel, as described in this paper, are of limited scope. Carousel provides supports for only two of the design tools in the tool cadre. It has none-the-less built the environment and set the stage for further development toward design automation. Standards for both the design library and the control code were defined and applied to the initial releases. Given these standards, further tool supports can now be added to the system easily and process flow automation can be integrated into the control code.

Carousel was also described in the light of formal database theory. Specifically, database system views and data models were reviewed. Further, concerns from the science of "human factors" were incorporated into the design. This consideration yielded a highly user friendly end product: an attribute vital to product success.

In summary, Carousel has both practical and theoretical foundations. It is the product, not only of application specific requirements but also of formal academic theory from the disciplines of computer science, electrical engineering, human factors, database theory and quality assessment studies. The result is a design service that has already paid for itself at the very least, in saved engineering hours.

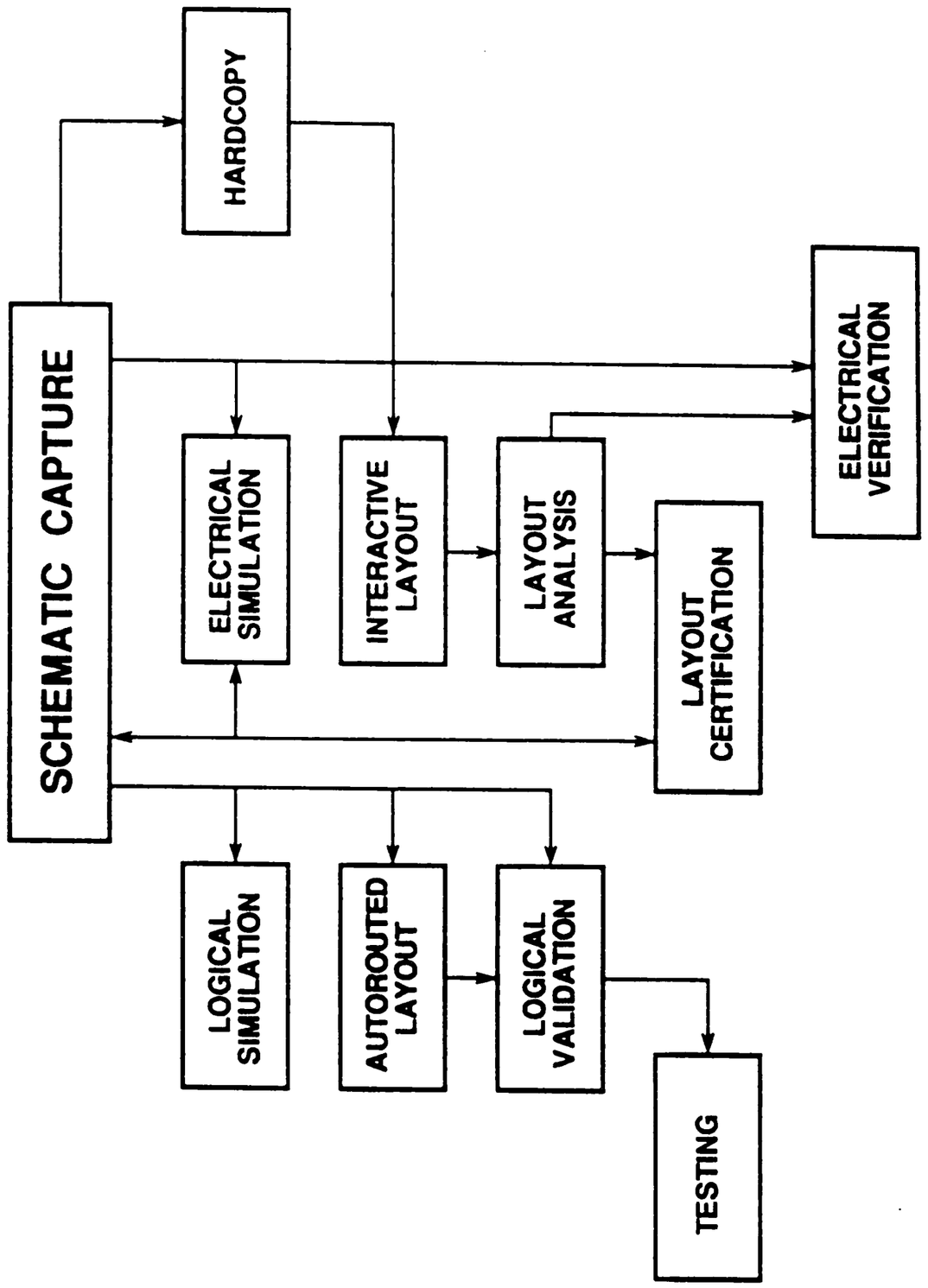


Figure 1. Process Flow for Analog IC Design

TOOL	FUNCTION
SCHEMA	schematic capture
MOTIS	logical simulation
LTX2	autorouted layout
MOTIS	logical validation
TPG2	testing
ADVICE	electrical simulation
GRED	interactive layout
HCAP	layout analysis
GEMINI	layout certification
ADVICE	electrical verification

Figure 2. Design Tools and Their Functions

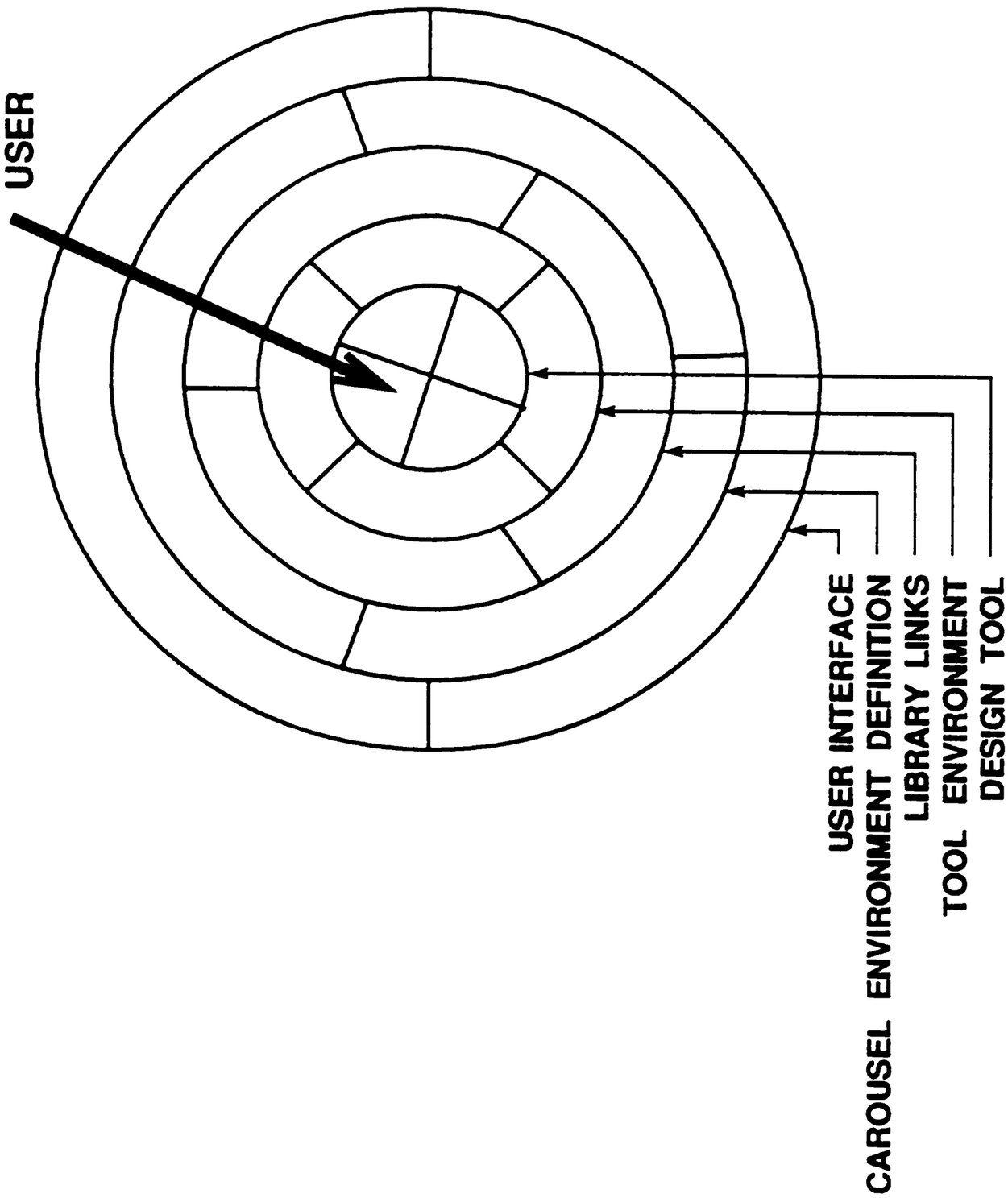


Figure 3. Levels of Specification in the Carousel Environment

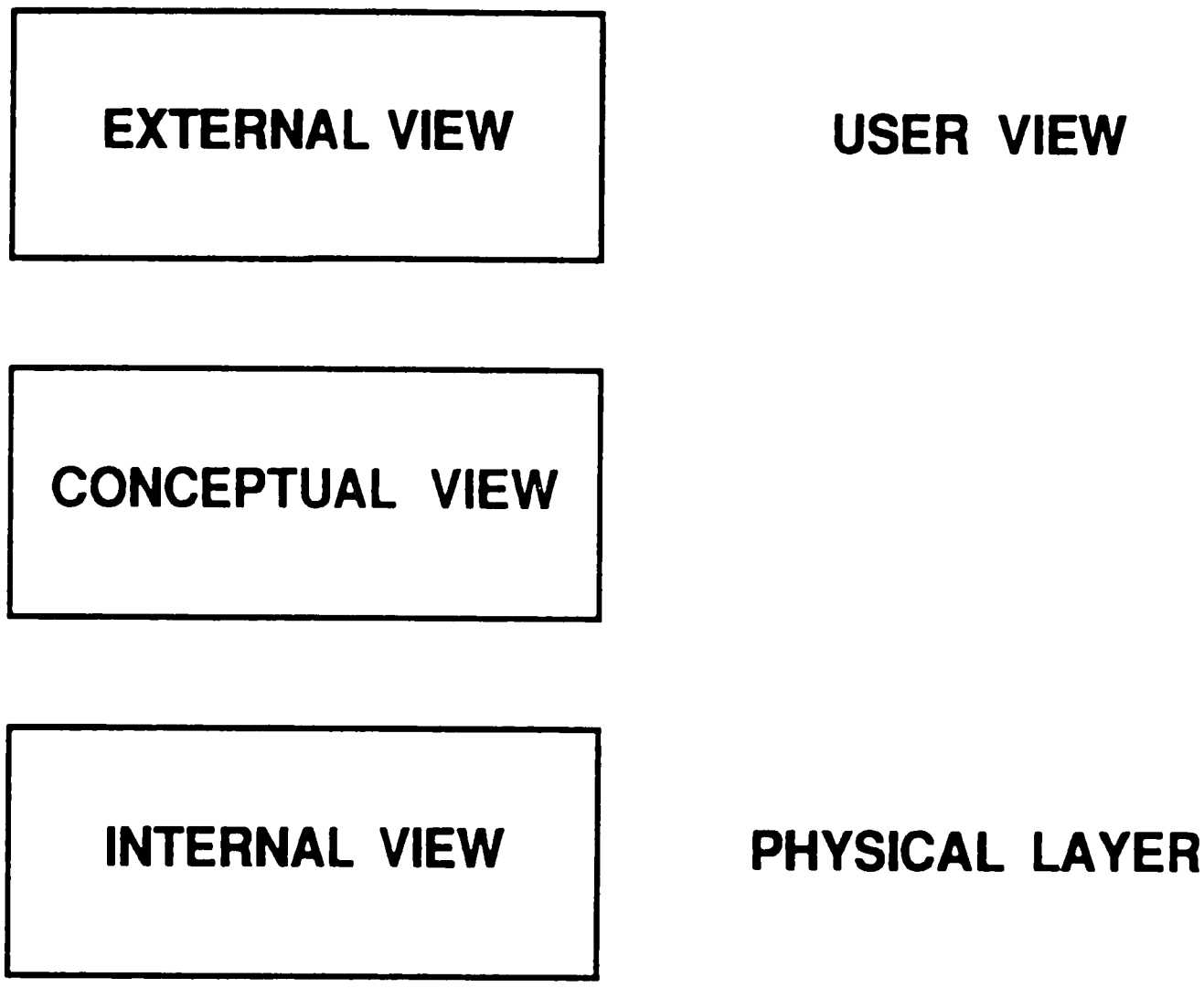


Figure 4. Database System Views

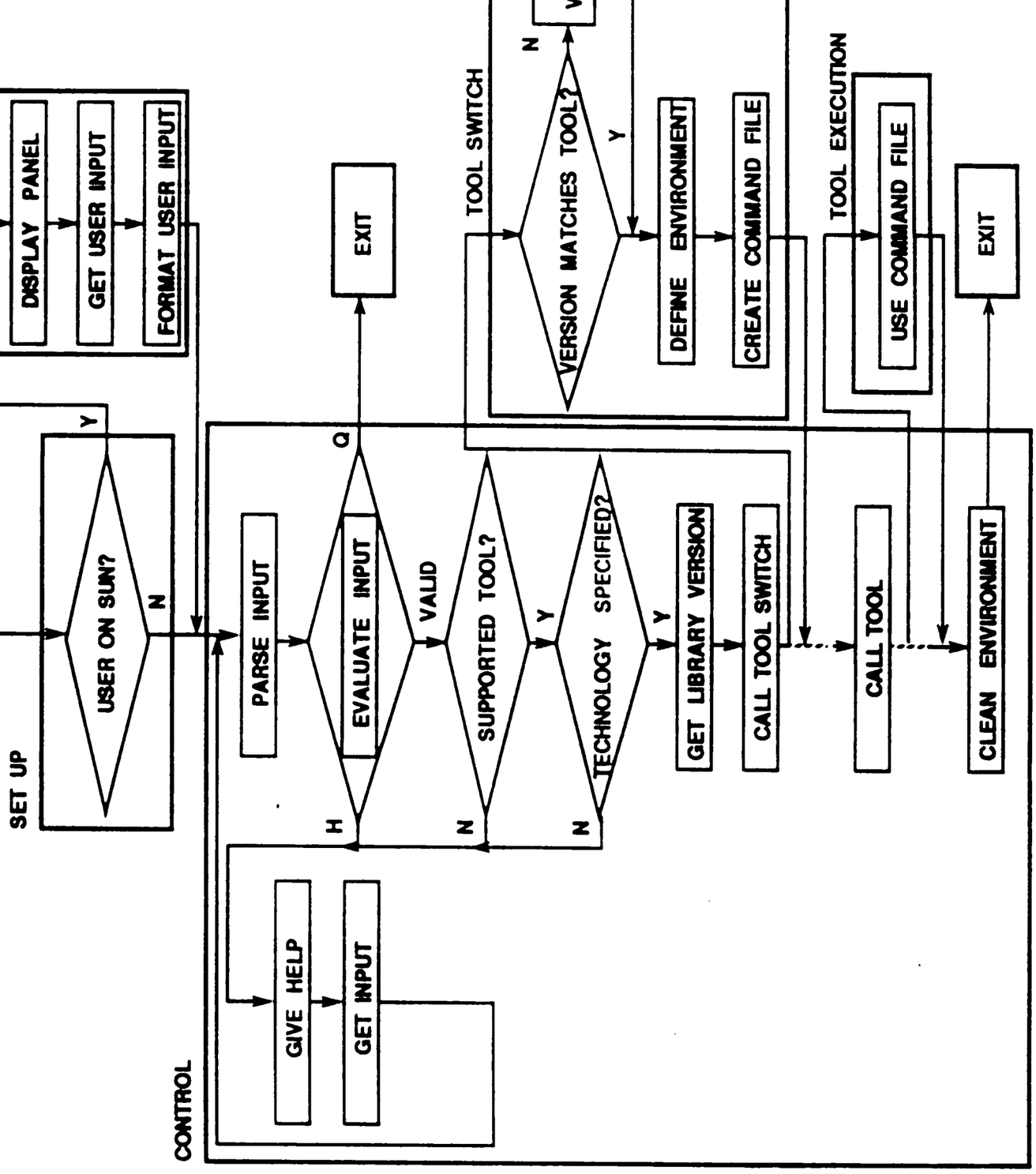


Figure 5. Flow Diagram for Carousel Control Code

	LIBRARY	CONTROL CODE
HIGH	TOOL	OPERATING ENVIRONMENT
•	LIBRARY VERSION	TOOL ENVIRONMENT
•	TECHNOLOGY	TOOL
•	SUB-TECHNOLOGY	TECHNOLOGY
LOW	DATA	

Figure 6. Levels of Specification for the Library and Control Code

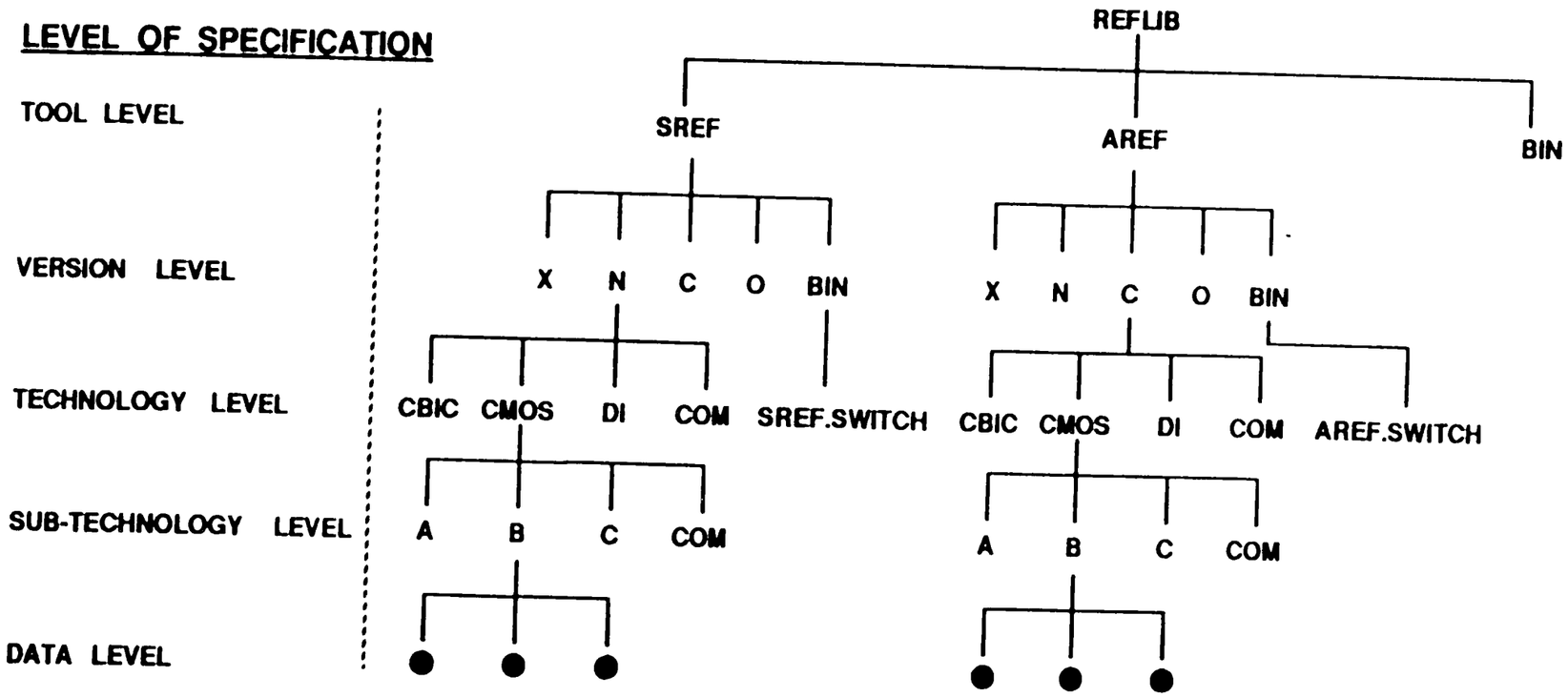
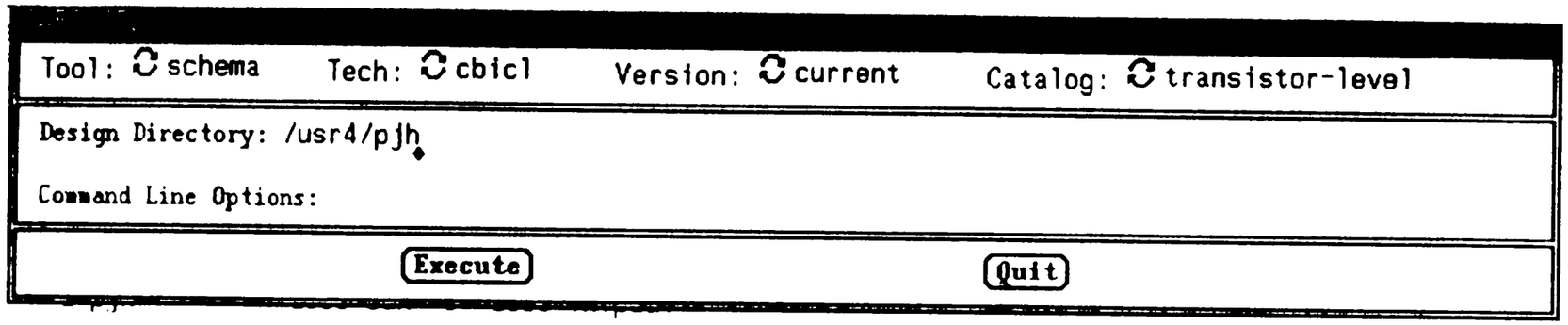


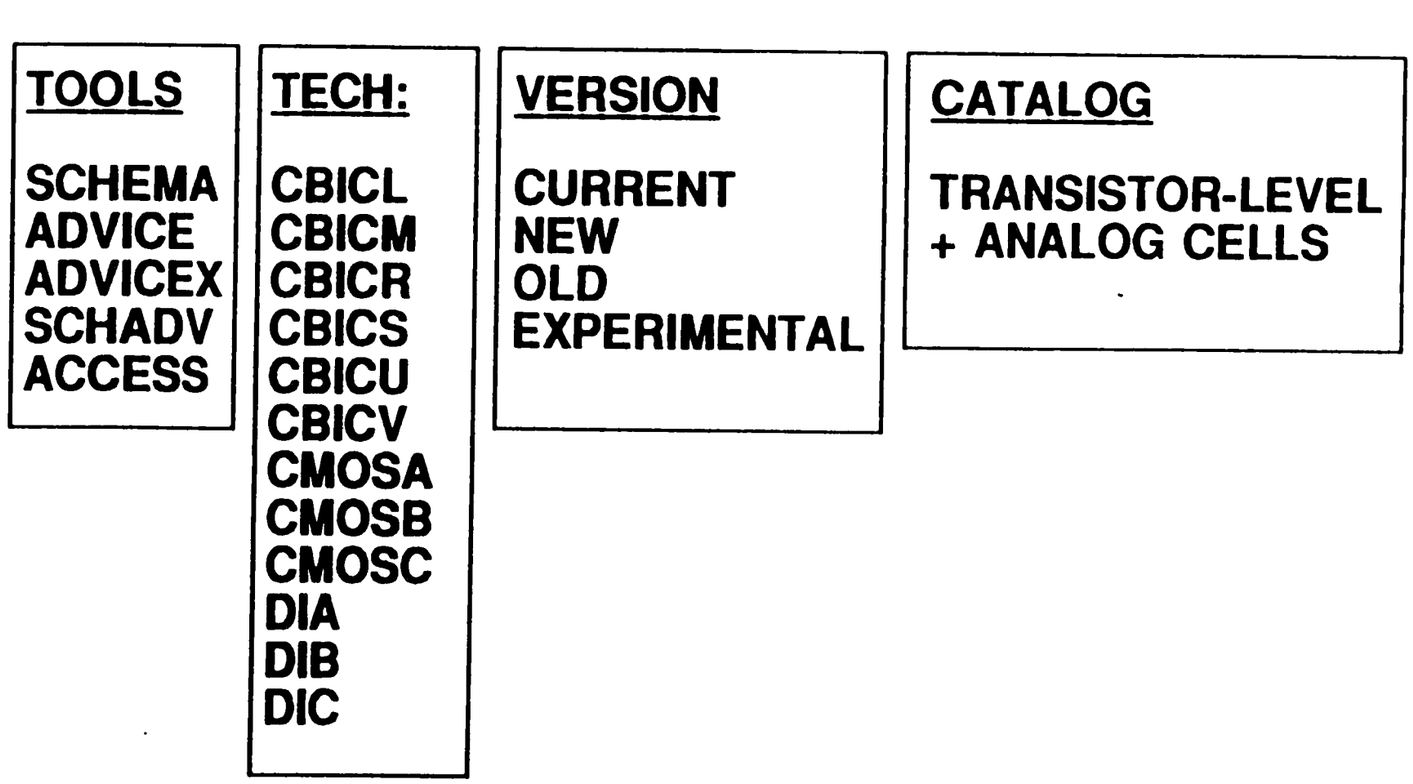
Figure 7. REFLIB Structure

```
The tools currently supported by carousel are:

     schema
          format:  schema (options) -x technology (options)
          example:  schema -d ic -t 4014 -x cbicu

     advice
          format: advice (options) -x technology (options)
          example: advice t=4014  -x cmosb

     access
          format: access [options]
          example: access

     schadv
          format: schadv color -x technology
          example: schadv green -x cbicm

Supported technologies include:
     cbicl, cbicm, cbicr, cbics, cbicu, cbicv, cmosa(cmos3.5),
     cmosb(cmos1.5), cmosc(cmos0.9), dia, dib, dic.

Library versions:
     to specify a particular library version, type
     xtechnology for experimental (ie. xcmosa)
     ntechnology for new (ie. ncbicr)
     technology for current or the default (ie. cbicm)
     otechnology for old (ie. ocmosb)
     Note: If the specified library version does not exist or is
           incompatible with the specified tool version, you will
           automatically be given a default library version.

Enter the command line for the tool and options you wish to use
or h for help
or q to quit the carousel.
```

Figure 8. Carousel Help Output



(a)



(b)

Figure 9. Carousel Panel Interface

```
call CAR [ tool options -x technology ]

CAR [ tool options -x technology ]
   look at user environment
   if user on SUN workstation and no arguments issued
     then call PANEL INTERFACE
     else call CAROUSEL [tool options -x technology]

PANEL INTERFACE
   show screen of panel options with defaults
   read user choices from panels
   set up call line for CAROUSEL using panel choices
   call CAROUSEL ( tool options -x technology )

CAROUSEL [ tool options -x technology ]
   set environment variable REFLIBPATH to location of library
    (ex. REFLIBPATH='logdir reflib'=/usr3/reflib)
A: look at user or panel input
   if valid tool specified
     then set environment variable TOOL
      (ex. TOOL=SREF for SCHEMA, TOOL=AREF for ADVICE)
     elseif H specified
     then give help (see Figure 8)
          read new input
          goto A
     elseif Q specified
     then QUIT (EXIT)
     else tell user "input not valid"
          give help
          read new input
          goto A
   if a technology is specified
     then set environment variables GENTECH and SPECTECH
      (ex. GENTECH=CMOS, SPECTECH=A for CMOSA)
     else tell user "technology required"
          read new input
          goto A
   if a library version is specified
     then set environment variable TECHVERS
      (ex. TECHVERS=N for new if technology specified as NCMOSA)
      else set environment variable TECHVERS to default value
   call .$REFLIBPATH/$TOOL/bin/$TOOL.switch ($var=value of var)
    (ex. /usr3/reflib/sref/bin/sref.switch (SCHEMA soft. switch))
   on return from switch, call TOOL [ options]
    (ex. SCHEMA -t 4014 -x CMOSA)
   on return from TOOL, clean up files created by CAROUSEL
   EXIT

NOTES:[] means arguments may or may not exist.
  "." in "call .pathtofile" means execute in the current shell
```

Figure 10. Pseudo Code for Carousel Control Code

```
$reflibpath/sref/bin/sref.switch
(SCHEMA software switch)
  check tool environment
  if lib. version not compatible with tool version in tool envir.
    then reset environment variable TECHVERS to SCHEMA default
  build a file in the user's current directory named
   $TECHVERS$GENTECH$SPECTECH (ex. NCMOSA)
   (contents of file that is built is a series of lines of form
   link gro /usr3/reflib/sref/n/cmos/a/com rea (see Figure 12))
   (contents of switch that builds this file are lines of form
   echo "link gro "$REFLIBPATH"/sref/"$TECHVERS"/"$GENTECH"/"
   $SPECTECH"/com rea" >> $TECHVERS$GENTECH$SPECTECH
   (where ">>" means append))
  EXIT
```

```
$reflibpath/aref/bin/aref.switch
(ADVICE software switch)
  check existance of lib. files for specified library version
  if no files exist
    then not a supported version
         reset environment variable TECHVERS to ADVICE default
  for each file in $REFLIBPATH/aref/$TECHVERS/$GENTECH/$SPECTECH
   /w2 directory, build a file with the same name in the user's
   current directory of the form shown in Figure 13(b)
   (built files are called c0, c1, c2, or cc0, cc1, cc2, or ac0,
   ac1, ac2, depending on the technology)
   (files in the w2 directory are of form shown in Figure 13(a))
   (editing commands are used on w2 files to create built files)
  set up environment for ADVICE to run
  EXIT
```

Figure 11. Pseudo Code for Carousel Control Code Switches

```
unlink all nosave
unlink fun all
link gro /usr4/syscad/nsyscad/lib/schema/groups rea
link gro . wri
link gro /usr4/reflib/sref/n/cbic/com rea
link gro /usr4/reflib/sref/n/com rea
link fun /usr4/reflib/sref/n/cbic/m/tech.cic
link fun /usr4/reflib/sref/n/com/tnspec.cic
link fun /usr4/reflib/sref/n/com/tspec.cic
link fun schmff
link fun schadv
```

Figure 12. CBICM File Built by SCHEMA Switch

```
* cc0    3.5um    case 0    nominal
/nch/nom
/pch/nom
/nchd/nom
/nplus/nom
/pplus/nom
/ptub/nom
/pcap/nom
/npn/nom
/par/nom
/w2/cdev
```

(a)

```
* cc0    3.5um    case 0    nominal
.use "/usr4/reflib/aref/c/cmos/a/nch/nom"
.use "/usr4/reflib/aref/c/cmos/a/pch/nom"
.use "/usr4/reflib/aref/c/cmos/a/nchd/nom"
.use "/usr4/reflib/aref/c/cmos/a/nplus/nom"
.use "/usr4/reflib/aref/c/cmos/a/pplus/nom"
.use "/usr4/reflib/aref/c/cmos/a/ptub/nom"
.use "/usr4/reflib/aref/c/cmos/a/pcap/nom"
.use "/usr4/reflib/aref/c/cmos/a/npn/nom"
.use "/usr4/reflib/aref/c/cmos/a/par/nom"
.use "/usr4/reflib/aref/c/cmos/a/w2/cdev"
```

(b)

Figure 13. ADVICE CMOSA File

REFERENCES

1. Luke, D. C. "A Comprehensive CAD Data Management System to Support Integrated Circuit Design and Development" Master's Thesis, Lehigh University, Bethlehem, Pennsylvania, 1984.

2. Date, C. J. An Introduction to Database Systems. Volume 1. 4th ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1987.

AUTHOR'S BIOGRAPHY

Teresa Elizabeth Krieger was born March 22, 1959 in Wilmington, Delaware to Paul O. Krieger and Kathleen M. (Redmond) Krieger. She attended Susquenita Elementary and Junior High Schools in Duncannon, Pennsylvania and Trinity High School in Shiremanstown, Pennsylvania. Teresa received her Bachelor of Arts degree in Mathematics in May of 1981 from Rosemont College in Rosemont, Pennsylvania. In 1982, Teresa enrolled at Lehigh University in the Department of Electrical and Computer Engineering. While at Lehigh, she completed several undergraduate courses in Electrical Engineering and Computer Science, then began graduate work for a degree in Computer Science. A Master's Degree in Computer Science will be awarded in January of 1989.

On June 1, 1981, she started working for AT&T Bell Laboratories as a Technical Editor, publishing IC design manuals. In October of 1983, she took an assignment at AT&T as a System Manager for three VAX 11/780 computer systems (VAX is a trademark of Digital Equipment Corporation). In 1985, she became part of the Design Automation Group at AT&T as a software developer. The system described in this thesis is one product of that assignment.