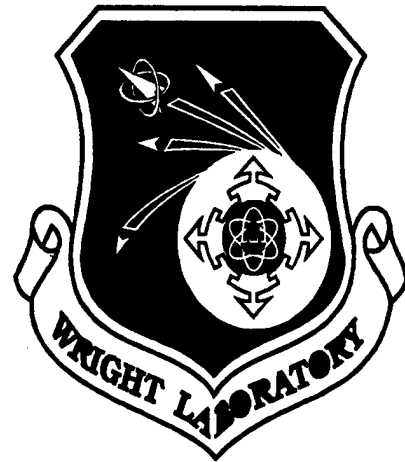


WL-TR-96-1089

**ADABRA – A Rapid Prototyping
Environment for Electronic Packaging
Design**



Sowmitri Swamy
Surendra Burman

Vista Technologies, Inc

AUGUST 1995

FINAL REPORT FOR PERIOD AUGUST 1993 – AUGUST 1995

19981026 053

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

NOTICE

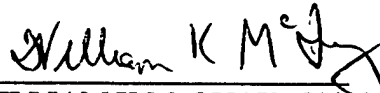
USING GOVERNMENT DRAWINGS, SPECIFICATIONS, OR OTHER DATA INCLUDED IN THIS DOCUMENT FOR ANY PURPOSE OTHER THAN GOVERNMENT PROCUREMENT DOES NOT IN ANY WAY OBLIGATE THE US GOVERNMENT. THE FACT THAT THE GOVERNMENT FORMULATED OR SUPPLIED THE DRAWINGS, SPECIFICATIONS, OR OTHER DATA DOES NOT LICENSE THE HOLDER OR ANY OTHER PERSON OR CORPORATION; OR CONVEY ANY RIGHTS OR PERMISSION TO MANUFACTURE, USE, OR SELL ANY PATENTED INVENTION THAT MAY RELATE TO THEM.

THIS REPORT IS RELEASABLE TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS). AT NTIS, IT WILL BE AVAILABLE TO THE GENERAL PUBLIC, INCLUDING FOREIGN NATIONS.

THIS TECHNICAL REPORT HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION.



LUIS M CONCHA
Program Manager



WILLIAM K McQUAY, Chief
Avionics Simulation Technology Branch
Avionics Directorate



STANLEY E WAGNER, Chief
System Concepts & Simulation Division
Avionics Directorate

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE August 31, 1995	3. REPORT TYPE AND DATES COVERED FINAL AUG 93 - AUG 95		
4. TITLE AND SUBTITLE ADABRA - A RAPID PROTOTYPING ENVIRONMENT FOR ELECTRONIC PACKAGING DESIGN		5. FUNDING NUMBERS F33615-93-C-1362 PE 62204 PR 6096 TA 60 WU 02		
6. AUTHOR(S) Sowmitri Swamy and Surendra Burman				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Vista Technologies Inc 1100 Woodfield Rd., Suite 437 Schaumburg IL 60173		8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Information Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson AFB OH 45433-7334 POC: Luis Concha AFRL/IFSD, 937-255-1902 ext 3578		10. SPONSORING/MONITORING AGENCY REPORT NUMBER WL-TR-96-1089		
11. SUPPLEMENTARY NOTES This is a Small Business Innovation Research (SBIR) Phase 3 report.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited.		12b. DISTRIBUTION CODE		
13. ABSTRACT (Maximum 200 words) Adabra is a tool that evaluates design alternatives for Multi-Chip Modules (MCM) used in electronic packaging. It is used to select a set MCM technology parameters that best fit a specific MCM design problem. The tool consists of Propagation modules and a packaging database. Each propagation module evaluates the design with respect to one technology parameter, e.g. metallization, technology, heat dissipation, etc. As technology changes, more modules can be added. A special correlator module correlates all the outputs and presents the result as a table of parameter values each of which represents a feasible solution. The technology database contains MCM technology data embedded in a PCTE (Portable Common Tool Environment) object base.				
14. SUBJECT TERMS		15. NUMBER OF PAGES 97		16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT SAR	

TABLE OF CONTENTS

Adabra: A Rapid Prototyping Environment for Electronic Packaging	1
An Outline of Adabra Architecture	2
Tool Integrating Framework	2
Packaging Design Tools	3
The PPN Computation Model	3
MCM Design and Brokerage Service for the US Electronic Industry	5
Business Opportunity	5
Technology and Business Development Plans	6
Phase III work	9
Productization of Phase II prototype:	9
Robustization of the tool	10
Technology modifications/improvements	10
Functional improvements	13
Testing and Training	14
Study of foundry processes	20
Business development activities	26
Adabra Tool Interface	53
Architecture of Adabra Tool Interface	53
The Tool Interface Module	54
The ToolAddress Table	54
The Callbacks Module	55
The Communication Channel Module	55
PPN Propagator	56
Propagator Module	56
Requirements as a client in the rapid prototyping environment	56
Requirements in terms of the functionality	57
Requirements for each sub-module in the Propagator	58
Communication Channel Requirements	58
Initializer Requirements	59
The Master Table	60
The Scheduler	60
PPN Integrator Requirements	61

PPN Compiler Requirements	61
Review of Requirements	62
Propagator Design	64
Architecture of the Propagator	64
Design of Propagator Sub-Modules	71
Communication Channel Module	71
Initializer Module	73
Channel Table Module	74
MasterTable Module	74
Scheduler Module	75
Multiple Propagators	78
PPN Redo	79
PPN Design	83
Introduction	83
Introduction to the PPN model	83
Definitions	84
Notes about the Requirements	85
PPN Nodes	93
Operators	94

LIST OF FIGURES

FIGURE 1. Architecture of the rapid prototyping environment	4
FIGURE 2. Computation for die-attach selection	12
FIGURE 3. Multibus II design steps	15
FIGURE 4. Multibus II padstack definitions	16
FIGURE 5. Symbols used in Multibus II design	17
FIGURE 6. Steps in layout preparation	18
FIGURE 7. Gerber and punch files generated	19
FIGURE 8. MCM-C fabricating process	24
FIGURE 9. Variants of the PCB to MCM conversion problem	41
FIGURE 10. Design flow for PCB to MCM conversion	42
FIGURE 11. MCM TCAD processes	47
FIGURE 12. Architecture of Adabra Tool Interface.....	53
FIGURE 13. Architecture of Adabra Tool Interface.....	63
FIGURE 14. Architecture of the Propagator.....	65
FIGURE 15. Action taken by the ChannelTable on receiving UpdateChannel message	67
FIGURE 16. Actions taken by a Communication channel on receiving special messages	68
FIGURE 17. Actions taken by the Initializer on receiving a start message.....	69
FIGURE 18. Actions taken by the MasteChannel on receiving CreateMoreChannels message.....	69
FIGURE 19. Propagator classes and their methods	70
FIGURE 20. Methods of Communication Channel.....	72
FIGURE 21. Methods of the Initializer.....	73
FIGURE 22. Methods of ChannelTable.....	74
FIGURE 23. Methods of MasterTable.....	75
FIGURE 24. Interaction between Scheduler and other Modules in the Propagator	76
FIGURE 25. Control Flow for the Toplevel Module of thePropagator	77
FIGURE 26. Redo execution	80
FIGURE 27. Methods of MasterTable.....	82
FIGURE 28. A PPN depicting interdependence between parameters	84

1. Adabra: A Rapid Prototyping Environment for Electronic Packaging

Vista Technologies, Inc.
1100 Woodfield Road
Schaumburg IL 60173-5124
(708) 706-9300 (P)
(708) 706-9317 (F)

Adabra is a rapid prototyping environment for Electronic Packaging. Adabra integrates many design tools that already exist to carry out design steps such as placement, routing, electrical analysis and simulation, etc. A database which stores technology specific data is also integrated in Adabra. The key feature of the environment is the use of a computation model called Parameter Propagation Network (PPN) that facilitates the selection of package design parameters. A PPN represents causal relationships between design parameters and thus allows a change in one design parameter to percolate through others. Parameters are represented by nodes in the network and their inter-relationships by arcs. Each node computes the value of a single parameter. A PPN computation, as carried out in Adabra, calculates various parameter values by interacting with design tools in the environment or with other PPNs. The concept of a PPN can be extended to form a network of PPNs. Several PPNs may be tied together to form an independent module, generically described as a *propagator*, which can be used to solve a specific packaging subproblem. A complete solution to a packaging problem may involve execution of a set of propagators which coordinate the interaction among different design tools utilizing technology data from the database. Two propagators for PCB component selection and MCM technology selection are currently integrated in Adabra.

Our focus in this project has been on developing a rapid prototyping environment for electronic packaging, laying the groundwork and developing technology leading to an eventual *packaging compiler*. The environment consists of a set of packaging tools that interact closely to generate a packaging solution. There are many individual tools that support package design at the component, board and system levels; they could tremendously benefit the user when tightly integrated under a common framework specifically designed for packaging applications.

Package design has characteristics that make it particularly suitable to rapid prototyping. It covers a wide variety of technologies and applications, but is amenable to a systematic approach to design. Moreover a number of design tools to carry out individual design steps already exist. If the tools are part of a package design-specific framework, the proper usage of tools, coordination among tools, and data exchanges between tools can all be sys-

tematically carried out by the framework with the active participation of the user. In this sense, the framework provides an environment that is greater than the sum of its parts.

Our rapid prototyping environment for electronic package design, Adabra, provides these features. The power of the environment lies in the PPN computation model which facilitates the design process. The model enforces a causal relationship between design parameters that underlie the design process. A PPN computation consists of the autonomous (meaning, not started by the user or any tool) propagation of parameter values to other parameter computations in a manner defined by a network. Constraints can be placed on parameter values in the network. While computing values for the design parameters with the given constraints, the PPN uses other application specific packaging tools that have been integrated in the framework.

1.1. An Outline of Adabra Architecture

Adabra consists of three components:

- the tool integrating framework,
- a set of packaging design tools for simulation, layout and analysis, and
- tool integration technology enforced by a PPN computation model

The Parameter Propagation Network is a computation model used to represent the inter-relationship between design parameters that are involved in electronic packaging. Each parameter is represented by a node in the network; their inter-relationships by arcs. During execution, the model interacts with tools in the framework, exchanging parameter values.

1.1.1. Tool Integrating Framework

Tool integration and inter-tool communication is provided in the framework by a client-server model. In this model, the various tools in the framework are clients that request services from an anonymous server. Tools request services in their native mode, that is correctly interpreted by the server. Besides, each tool operates independently and is unaware of the identities of other tools. The subcomponents of the tool integrating framework are:

- a Server that can be thought of as the hub of a star network. Clients are connected to it via persistent socket connections, and may communicate with each other indirectly through the Server. Though the PPN model is an integral part of the framework, it can be considered as a special client that uses the same services of the Server that other clients require.

- a client registry, which is a list of all client identities that are currently active. All the clients that the server connects to have an entry in the client registry.

1.1.2. Packaging Design Tools

A large number (~100) packaging design tools for simulation, layout and analysis are available from vendors. A representative selection of these tools from vendors and Universities can be selected and integrated in an environment like Adabra. The Server, described in a following section, is the tool that provides integration services to all packaging tools. The Server views all tools in the framework as it's clients and maintains a current list of active clients in a data structure called the client registry. Of particular interest is a client called the Propagator that is the computation engine for the PPN model. In a sense, it drives the tools in the environment towards a packaging solution with the help of the Server.

1.1.3. The PPN Computation Model

The salient features of the PPN computation model include computation of parametric values, asynchronous propagation of parameters to other parameter, computation and recomputation of parameters by backtracking when parameter constraints are not satisfied. Complex parameter constraints and interactions can be effectively handled by a computing paradigm called *parameter propagation*. In this approach, a network of computational elements or *nodes* is constructed. Each node computes the value of a single parameter. The interconnections between nodes reflect the interdependencies between corresponding parameters. The network, called a *parameter propagation network* or PPN, is typically a *directed acyclic graph* (or *level graph*) to ensure that its computation terminates in finite time.

To begin the computation, the values of the nodes at the lowest level of the PPN level graph are initialized. The remaining nodes then perform their computations asynchronously, updating their outputs whenever one of their inputs changes. A constraint may be specified at a certain node: if the result of that node computation violates the constraint, the network initiates a backtracking procedure through the level graph, exploring the space of parameter values until the constraint is satisfied. Thus, parameter values can be visualized as propagating back and forth through the network, influencing, and being influenced by, the computation of other parameter values.

The network is said to have completed its computation when the output of each node remains constant. At this stage, the set of parameter values at the node outputs represent a solution which satisfies all the parameter interactions and constraints, if such a solution exists.

The concept of a PPN can be extended to that of a *PPN network*, in which several PPNs are connected up in a manner similar to the way nodes are connected in a PPN. Each PPN

has one or more *output nodes* which compute values of parameters of interest to the user, and *internal nodes* which compute values of parameters that are at a level of detail hidden from the user. Typically, one constraint may be specified per PPN, and backtracking does not occur across PPNs, since each PPN is expected to be a relatively independent computational unit. Output nodes of PPNs may propagate *lists* of output values, instead of a single value, so that the network as a whole can generate a set of valid solutions.

Figure 1 below shows the various sub-components of our rapid prototyping environment. The server initializes, terminates, and communicates with tool instances that are registered in the client registry. The PPN computation model is executed by the module called the Propagator. The Propagator's interface to the framework is just as any other tool: i.e., the server initiates, terminates, and communicates with the Propagator in the same manner that it does with a general tool. A common user interface provides the means by

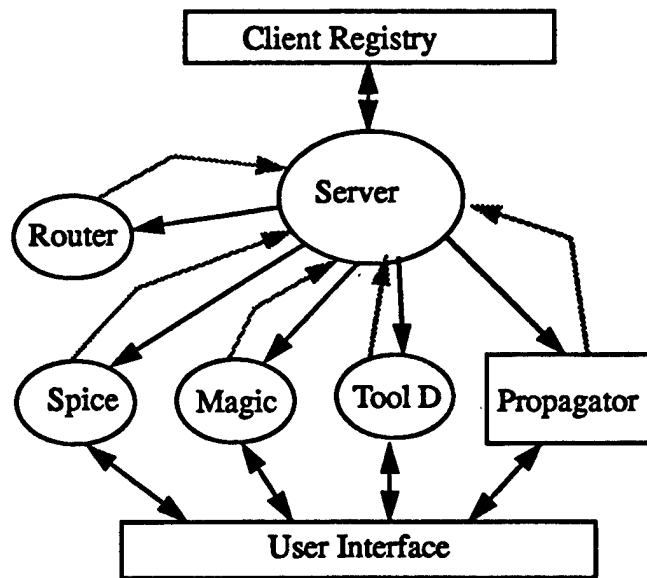


FIGURE 1. Architecture of the rapid prototyping environment

which the user can graphically interact with the Propagator. In most instances, the user does not directly interact with the Server. Tools may have their own private graphical user interfaces that they do not share with other tools in the framework.

2. MCM Design and Brokerage Service for the US Electronic Industry

2.1. Business Opportunity

This section describes the steps we have taken in commercializing the package design technology that has been developed under a Phase II SBIR program. Our commercialization activity has focused on the Multichip Module design market. This market is capable of supporting high technology products and services, and has the potential to rapidly grow into one of the dominant packaging technologies of the nineties. With the introduction of many bare chip suppliers in the market, MCM market is poised for a significant growth. New applications such as cellular phones, personal digital assistants, PCMCIA cards etc. are beginning to provide the required volume production, the MCM market has been waiting for. A self-sustaining market in MCM products and services is expected to commence very soon.

Under the SBIR Phase II project discussed earlier, we developed enabling technology for MCM design. The highlight of the effort is the development focused around a formal model of parameter interaction (PPN model) involving packaging parameters. Using this formalism, we now have the capability of rapidly developing design advisors which can enforce a wide variety of parameter interaction induced by physical, electrical, thermal, material, manufacturing, or other reasons. In addition, we have developed the infrastructure necessary for such tools to function in a design environment. This consists of the Adabra tool integration framework built on an object base called PCTE. The environment consists of several packaging object bases, several design tools, including Propagators, and tool interaction technology that can be used for tools that are expected to be part of the product environment.

We believe that there is a very good opportunity for successful commercialization of our technology as an MCM design and brokerage service. We now offer full MCM design service (L, C, D, and the newer hybrids) and are able to perform foundry selection based on customer needs. Our competitive edge is derived from the fact that our technology is used to evaluate materials, processes, and design rules in order to select the best match with customer requirements. It is an imaginative plan to provide a "one-stop-shop" solution to many medium and small hardware manufacturers that cannot afford the in-house expertise or investment to use MCM's. We encourage small and medium hardware manufacturers to use MCM technology by offering them bias-free advice on MCM usage (we can run our decision tools in their presence), through lower NRE costs and, hopefully, lower manufacturing costs through broker induced consolidation of orders.

The business plan that we have developed and described below has several advantages. The business plan makes use of Vista's existing infrastructure and experience. Our unique perspective, as a foundry-less MCM design house enables us to assess foundry processes

strengths and weaknesses to the benefit of the customer. It adds to the infrastructure and facilities necessary for MCM technology to reach critical mass.

It should be noted that many companies offer design services tied to the specific architecture and manufacturing processes that they support. Our advantage and uniqueness are our technology that allows us to be selective in both process and design to the benefit of our customers.

2.2. Technology and Business Development Plans

The technology development plan has been carefully formulated to support our business development plan, while still being in the mainstream of technology development. We refocused Adabra environment to emphasize three areas: MCM technology selection, design, and process selection.

MCM technology selection

MCM packaging is available in many different technologies, and the problem of choosing a good match for a packaging application, based on user requirements, is very important. In fact, what is needed is a multitude of solutions which the brokerage service can grade based on technical, economic and manufacturing criteria. We implemented an MCM technology selection Propagator during Phase II, and we are satisfied with its performance. In addition to this, we have enhanced its capabilities in several ways, most notably in incorporating several new MCM technologies which straddle more "traditional" L, C, and D categories. We have also improved the quality of its recommendations by implementing several enhancements to the basic Propagator and PPN computation model.

MCM design

We are enhancing the support capabilities of the Adabra environment for MCM design in two ways. We are building a new Propagator that will address design parameter selection and that will work in tandem with the following tools: layout, electrical and thermal analysis and simulation tools, and the technology selection Propagator. It will explore the design space between various MCM technologies, and consider trade-offs using MCM design tools integrated in the design environment.

It might be recalled, that during Phase 2 we consciously (and correctly) made the decision not to implement tools already available in the market such as MCM routing and analysis tools. To complete the Adabra environment, we have selected and acquired Cadence MCM design tools (Allegro-MCM) for integration.

Process selection

We have developed good knowledge of process and fabrication parameters as well as process advantages and idiosyncrasies of few foundries. We are in the process of completing a new database of process parameters and foundry parameters, and will develop guidelines on how they affect design and system level parameters. Process selection and foundry selection will be manual, but will heavily depend on the Propagators, design tools, and the process database. In the long run, process selection will involve process simulation along the lines of IC process simulation.

Our business plan consists of the following steps:

- We established technology non-disclosure agreements with two foundries: Acsist Associates (Minneapolis) and CTS Microelectronics (Indiana).

This entails selection of a representative list of foundries, initiating preliminary discussion leading to the agreement to provide data. We have found some foundries to be reluctant to provide data, while others will have no problem providing data under non-disclosure. The choice of foundries will be representative of both technology and market. This will include the following:

L, C, and D technologies

Low volume/high-end applications

-aerospace and military

High volume/low cost

-consumer electronics

Special applications

-automotive

-telecommunications

-computers

Other

-niche technologies- e.g. diamond substrate

-custom form-factor

- We are currently working on establishing the modalities of a working relationship with each foundry, including demarcation of responsibility, formats for technical disclosure, and financial arrangements

Since foundries do not have standardized technical non-disclosure formats we are trying to understand foundry specific terminology, formats and measurement methods (this part will be made easier if ARPA sponsored standardization gets finalized and adopted industry wide). With each foundry we will agree on who is responsible to which degree on what, and agree on a cost schedule for services.

- We are also in the process of establishing supply relationships with bare-chip vendors including KGD methodologies. With the introduction of Intel's SmartDie program, KGD supplies have improved considerably. Some foundries can supply KGD from their own sources, others may want us to provide KGD. Cost and delivery deadlines may drive choice. Relationships with KGD sources may be a critical factor in business success

Vista's MCM design services is now open for business. We have successfully completed two trial MCM-C (LTCC) based designs. We have been actively trying to establish contacts with other MCM foundries and other potential MCM users.

3. Phase III work

The importance of Phase III in the development of the business was *critical* because it provided the bridge between technology development in a laboratory-like environment and its implementation as a technology to solve problems in the real world.

No new technology was developed during Phase III. Instead, the prototype developed during Phase II was refined, tested, and integrated with other MCM design tools.

The Phase III work can be characterized as a trial period during which a maturing process for the technology was realized. Phase III work can be described in terms of three activities: *Productizing the Phase II prototype; implementing an extensive training and testing program during which real MCM designs were implemented from start to finish; and a business development phase where the details necessary for doing MCM design and brokerage business were implemented prior to opening for business.*

3.1. Productization of Phase II prototype:

Productization work focused on three aspects that needed improvement: performance of the tool, elimination of software bugs, and improving and augmenting technology selection algorithms to improve the quality of the output of the tool.

1. Performance improvements

The prototype at the end of Phase II was slow, but during Phase III we made very significant improvements in response times. We quickly found that the main cause of the slowness rested in the response time of the object base to the tool's queries. A lot of effort was expended on improving the format of the queries, changing the schemas that governed the data, and modifying the query engine. Improvements were obtained in response time, but they were not dramatic. The PCTE object base that contained the database for the operation of the tool was carefully examined and we concluded that the complex structure of the object base made further improvements at that level unlikely.

During the course of trial runs of the tool, we discovered that the tool was being used repeatedly with minor changes in parameters; yet each of the queries that were formulated were handled in the same manner by the object base resulting in a predictably slow response. The obvious solution of providing a data cache was implemented, resulting in a dramatic improvement in performance. Surprisingly the data cache also improved the performance in general of the tool, even when different sets of data were run. Further

improvements in performance were obtained by simplifying many of the control structures of the propagator, while still retaining an interactive mode of operation.

Another technique that resulted in significant improvements in performance was to simplify the query presented to the object base by performing more query related processing within the propagator. In effect, part of the functionality of the query engine was transferred to the PPN's themselves in the form of more extensive node computations.

As a result of these improvements, it is now feasible to make several runs of the tool in an hour.

3.2. Robustization of the tool

The principal focus of this activity consisted of the following: elimination of critical bugs that would render the tool inoperable, minimizing memory leaks and reducing the run-time memory usage, installing better diagnostics, and providing more type checking of input and computed parameters.

Software bugs that resulted in segmentation violations were targeted and removed to the extent that they were detected. Minimizing memory leaks was a significant effort since memory allocation was reasonably common throughout the operation of the tool. Memory leaks were significantly reduced through eliminating larger leaks. Not all memory leaks were eliminated, since the benefits would not be proportionate to the cost incurred in doing so. Many run-time diagnostics were introduced and error and warning messages were improved. To further robustize the tool, careful type and range checking for technology parameters was introduced. This was needed because the object base can be put into a non-responsive state if the query parameters do not match the type specified in the schema definitions.

3.3. Technology modifications/improvements

The technology selection propagator was subjected to an extensive review that covered its PPN structure, parameter computations (algorithms), and modifications to the object base. Practically every PPN was modified in the light of greater experience with the technology selection process. Several new parameters such as "Die availability", "Number of dies of a given type", "Module Thickness", etc. were found to be significant factors in technology selection and were introduced into the various PPN's. In addition, two new PPN's -- Signal Layers and Module Attach were added to the network. These changes provided significant qualitative improvements to the tool's evaluation of various MCM technologies.

Here is a brief summary of the technology improvements:

PPN: MCM Signal Layers

This PPN estimates the number of signal layers required by the netlist prior to actual routing. Previous estimates used either a constant value or Rent's rule. A more sophisticated technique involved converting all the multi-terminal nets into two terminal nets following which a geometrical calculation combined with statistical data provides a good measure of the number of signal layers. The parameters used in the computation to produce the estimate include the number of chips, and the number of nets, and the total number of package IO's

PPN: MCM Technology Selection

Extensive trials were conducted to obtain qualitative improvements in the performance of this PPN which selects a specific MCM technology for implementation. The number of technology choices was expanded to cover: MCM-L, TFC (MCM-C), LTCC (MCM-C), T-Tape, HTCC, Ceramic, metal, Silicon, GE-Chips First technology, and MCM-L/D (IBM). The criteria to select among these technologies was expanded to include several new categories. They now include: Thermal properties, Geometrical properties, Reliability, performance, and Miscellaneous. A total of 13 parameters were used to represent these criteria whose relative importance was reflected in the selection algorithm.

PPN: MCM Die Attach

The improvements made to this PPN are extensive. First, we considered die attachment at two distinct levels: the die and the module, and found that these are related choices that need to be separately evaluated. Thus the original die attach PPN was split into separate PPN's for die attach and module die attach.

The die attach techniques were expanded to include a new technique called Overlay(HDI) in which die are glued into recesses and the interconnection layers are laid over the die. New parameters that were included in the selection criterion included die availability, testability, burn-in, electrical characteristics (inductance, mutual inductance, capacitance, transmission line capability), mechanical robustness, etc. Then, a two step selection algorithm was implemented.

PPN: MCM Module Attach

This is a new PPN and its purpose is to evaluate the selection of die attach for individual dies in the context of the overall module. At the module level, various combinations of individual die attach solutions to be assembled on the same module are evaluated on the basis of four parameters: Silicon density (measures how close the die can be sited on the module), manufacturing complexity (a rough measure of how hard it is to manufacture the design), manufacturing flexibility (the ability of the die attach technique selected to adapt to a process parameter change), and Module die attach cost.

An example of the computations is shown below for a four die design.

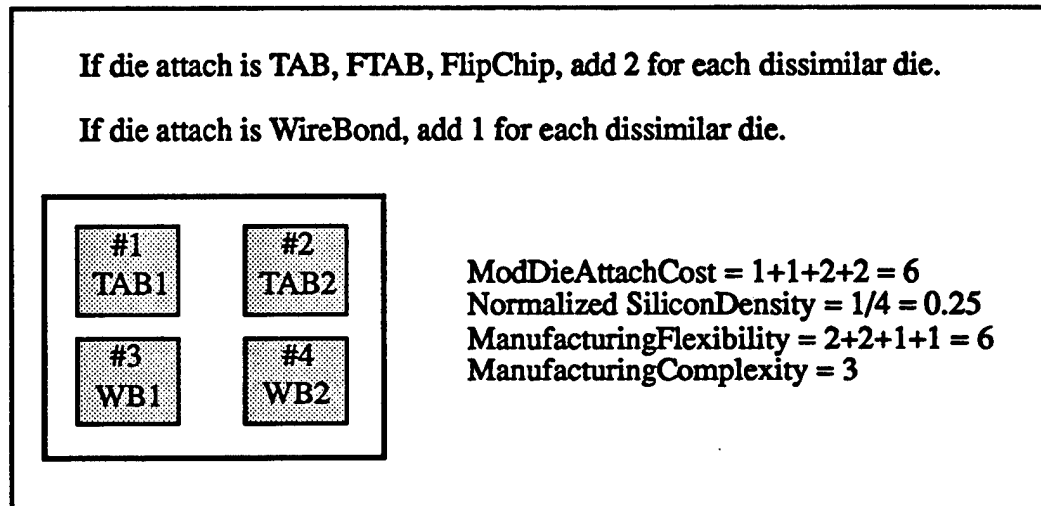


FIGURE 2. Computation for die-attach selection

Wire bonding can be adapted for components with various die I/O, die geometry, package geometry and production process parameters, by programming the bonding machine. Thus, this interconnection technique does not require any hardware changes for devices of various different parameters. TAB and flip chip require hardware changes to accommodate devices of various design and process parameters, since their tooling orders are customized to a specific die and package and require different equipment. However, for multichip applications, a module may consist of die attach methods of different types and the overall manufacturing flexibility would refer to the combination of all the die attachments in the module. Since manufacturing flexibility refers to the adaptability for different design and process parameters, the overall flexibility for a module with mixed interconnections can be determined by considering the dissimilarities (nature) of different dies.

PPN: MCM Metallization

This is a new PPN that was introduced into the technology selection process. It estimates the metallization required in the module based on the results of previous technology selec-

tion such as substrate, dielectric, technology, etc. The major characteristics used to select metallization include adhesion to substrate, low resistivity (high conductivity), corrosion resistance and chemical inertness, and CTE match with the substrate.

3.4. Functional improvements

Several functional improvements were also made to the Propagator. These include p-list propagation and range inputs. P-list propagating is a technical enhancement that needed extensive changes in the PPN architecture. It allows the propagation of multiple values across PPN's. This is important because it allows the parameter selection algorithm to provide a set of multiple values in response to a set of single values at its inputs. Thus multiple solutions are produced resulting in a better match of the chosen solution with the design parameters. The quality of the solution is greatly enhanced.

Another enhancement is the introduction of range values. This allows input parameter specifications to be a range rather than individual values. An application of this feature can be seen in the module attach PPN where a number of independent solutions are simultaneously specified, each representing a die attach solution for one specific die.

We also implemented a feature that allows constraints to be set on parameter propagation. Constraints are settable on only one node in a PPN, i.e., the output node. The constraint consists of a logical expression in an interpretive language (X lisp) that uses the Hyper-Web server to evaluate the expression. The expression can be changed at any time without the need for recompilation. The expression is persistent in that it can be re-used at a later time. Special nodes in the object base are used to store the constraints as an X-Lisp expression.

Constraints allow the user to screen out parameter data values that may not be required. A common use of constraints is in reducing the number of feasible solutions (since examining all the solutions may not be practical) and in tailoring the solution set to a specific set of parameter values that a foundry may be able to implement.

Filer commands now allow the reading of file inputs, writing to file, saving and restarting the current user session, crash recovery, etc. It is a very useful feature for extended and repeated runs of the technology selection propagator.

A feature that we added to greatly improve the presentation of the tool is the correlation of the outputs of each PPN in a tabular manner. The rows of the table reflect entries that are individually correlated with the rest of the row entries from other PPN's and therefore cor-

respond to the outcomes of individual wavefronts. The correlator also provides a table driven user interface for selection of specific parameter values.

3.5. Testing and Training

An extensive testing and training program was initiated as part of the business development strategy. Testing was made difficult by the fact that test data were not readily available. Some test data were gathered from published technical papers. Others were developed in consultation with the foundry industry.

It was absolutely essential that we get acquainted with the nitty gritty of designing MCM's using real examples. First, we had to settle on an MCM design suite of tools to complement our tool. Acquisition, installation and evaluation of several toolsets in the market was tried and we settled on the Cadence MCM Allegro toolset.

The training program involved actual MCM designs, but these were difficult to obtain due to their proprietary nature.

The actual training took several months to achieve a level of technical skill that would be needed to service customer needs. Here is an example description of the type of design work that was performed as part of the training. This sample was the outcome of several iterations caused by our lack of experience, the inadequacies of the Allegro toolset, and the changing conditions imposed by the customer.

Example: Multibus II design

Design Description

The Multibus II design consists of nine chips and ten discrete components. The design has approximately 750 net pins and about 250 nets. The netlist includes 7 critical nets which need to be routed on a separate layer to meet high speed timing constraints. The chips include a Microcontroller MR87C51FB-16, one MPC MQ82389, one FPGA MMBT2222A and six resistor networks. The discrete components include two NPN transistors MMBT2222A, one resistor of 250 ohm M55342k06B250DM and seven capacitors. The MCM package is a PLCC type having 224 pins. In addition to these, the design includes six test pads each of 15 mils x15 mils to be attached on the back of the seal ring. The design is to be completed using 4 signal layers, 3 ground planes, 1 voltage plane and a

separate layer for routing critical nets. The design steps used are shown in the figure below.

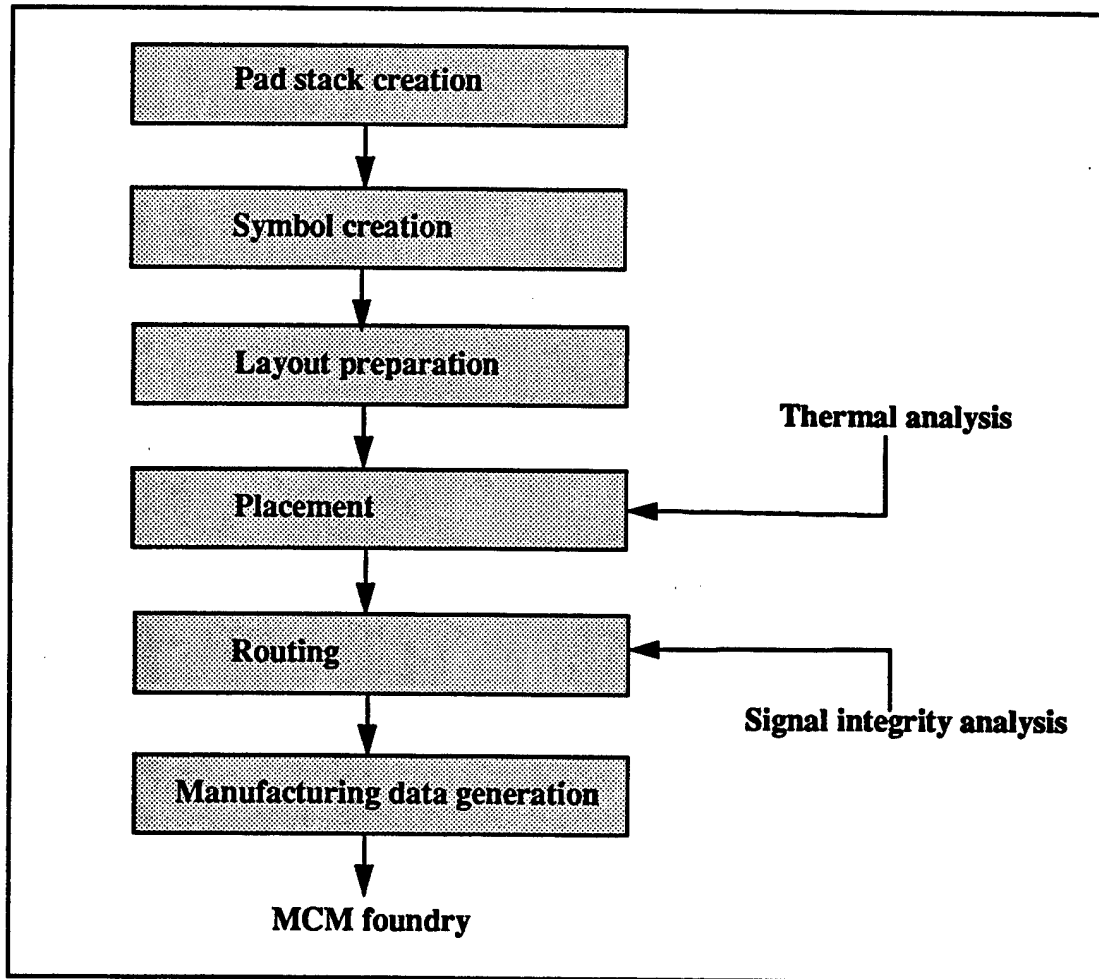


FIGURE 3. Multibus II design steps

Padstack Creation

Padstacks were used to define all the pins, vias, die pads and bond pad information for the layout. For Multibus design, pad stacks were defined for die pads, bonding pads, buried/blind vias, and I/O pins. The definitions for different padstacks are shown below.






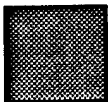
	20 mils x 6 mils	bonding pad
	4 mils x 4 mils	diepad
	4 mils diameter	blind/buried via
	50 mils x 12 mils	I/O pad
	50 mils x 24 mils	I/O pad
	15 mils x 15 mils	test pad

FIGURE 4. Multibus II padstack definitions

Symbol Creation

Symbols corresponding to the components in the design were created using the Symbol Editor in our design suite. There are nine types of symbols which were created for this design. The symbols for the chips were created for wire bonding assembly technique. The pad stacks which were created in the previous step are used in the symbol creation process. Basically symbols consist of part geometry and pad stacks corresponding to the die and bonding pads for a given component. In case of discrete components, the symbols used pads for the surface mount assembly. The main challenge which was encountered in symbol creation process was to create bond pads for the high I/O chips. All the symbols for the chips were created using single row bonding pads to facilitate the process. The bonding pads were created to be accommodated in a single row. Because of this single row

limitation, one symbol (U1) became unusually large. A standard symbol for SOT-23 (Q1 and Q2) was used for transistors. A device file containing electrical information for the pins was used for each symbol to verify one-to-one correspondence between pins and the pads in the symbols. The symbols used in the Multibus design are listed below.

Symbols for chips -----
die2002.psm die4010.psm die82389.psm die87C51.psm
Symbols for discrete components -----
cap01uf.psm res250.psm sot23.psm
Symbol for the I/Os -----
io.psm
Symbol for test pads -----
ecp.psm

FIGURE 5. Symbols used in Multibus II design

Layout Preparation

After all the symbols have been created and saved in the library, the layout was prepared for placement and routing. The layout preparation involves defining design rules (properties and constraints), defining layers (cross section), creating substrate geometry, part keepins and keepouts and route keepins and keepouts. Once the layout has been defined, the netlist is loaded and then blind/buried vias are created. Our design tool set allows creation of all possible blind/buried via for a given cross section. The suggested stackup by the manufacturer was used for the layout cross section. For Multibus design, a substrate geometry of 710 mils x 710 mils, route keepin of 700 mils x 700 mils and part keepin of 600 x 600 mils were used. The different steps for preparing the layout for this design are shown in below.

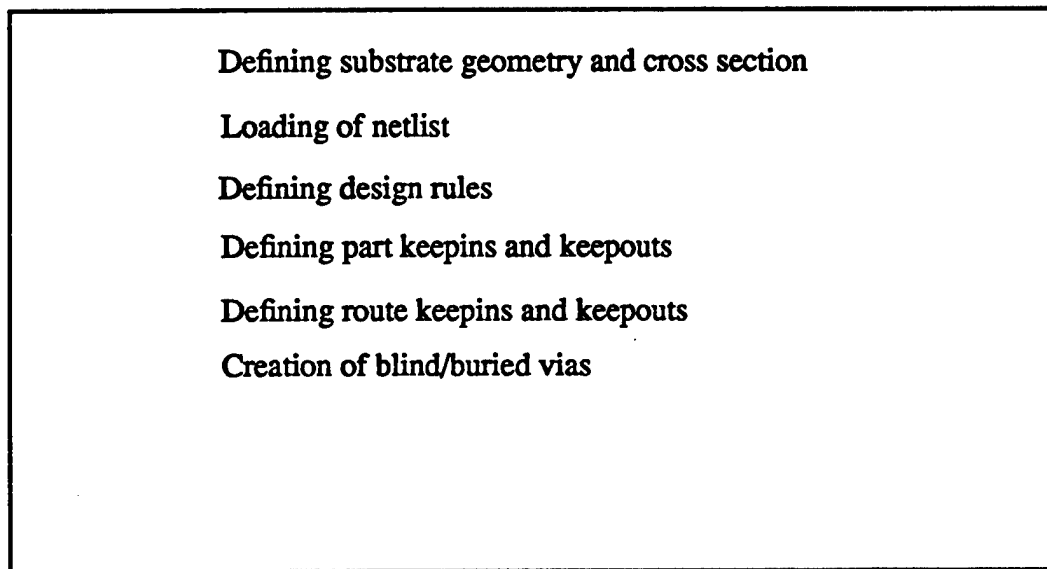


FIGURE 6. Steps in layout preparation

Placement

Placement and routing are done using the Layout Editor. The placement is similar to the one suggested by the manufacturer. Our design suite provides tools for placement evaluation and thermal analysis (Thermax) at placement level. Our design suite also provides autoplacement capability. In this design, no routing is allowed on the top layer within the seal ring area.

Routing

After the placement was finalized, routing was undertaken. First, vias were created for I/O pins and bonding pads since no routing was allowed on the top layer. In this design, vias for the bonding pads are required to be in a staggered fashion. This necessitated the creation of vias for each bonding pad to signal layers. A routing utility called *zrouter* to create vias for each I/O pins was used. Via creation for the I/O pins was easy. However, vias for the bonding pads were created separately for each row or column. To facilitate the routing completion, vias from the top layer were dropped to x1 and y1 layers and distributed evenly in such a way that vertical vias are on the y1 layer and horizontal vias are on x1 layer. After the initial via creation, critical nets were routed on *critical signal layer*. Power and ground nets were assigned NO_ROUTE properties because these nets were to be routed after the signal routing. Autorouting was used to complete a majority of the nets. We were able to achieve about 72 percent routing (20% P/G nets) using autorouting. How-

ever, the remaining 8 percent nets were routed manually. A very few nets had to be routed on critical signal layer in order to achieve 100 percent routing completion for the signal nets. Since the film type is positive, power and ground nets were routed after the signal routing. The design has three ground and one voltage planes. There are 57 ground connections and 37 voltage connections. Our design suite provides glossing routines to clean up the routing before the artwork generation is done.

Manufacturing Data

There were two types of manufacturing data that were needed: Gerber data for each conductor layer, and punch files for via informations. Gerber data were generated using artwork generation capability of the design tool set whereas punch files are generated using NC Drill programs. However, the format of punch files required by the manufacturer is different than what is generated by the design tool. Therefore, we have used post-processing routines to convert the punch files into the right format. The Gerber files and punch files are listed in Figure 7.

Gerber files	Punch files
filmx1.art	viax1.dat
filmy1.art	viay1.dat
filmx2.art	viax2.dat
filmy2.art	viay2.dat
filmg1.art	viag1.dat
filmg2.art	viag2.dat
filmg3.art	viag3.dat
filmv1.art	viav1.dat
filmcs.art	viacs.dat

FIGURE 7. Gerber and punch files generated

In addition to the Gerber and punch files, the pen plots of different parts of the design have also been generated.

Report Generation

Our MCM design suite provides capability to generate a wide range of design reports. The following types of reports have been generated:

Design Rule Checks
Net List
Placed Components
Summary Drawing Report
Unconnected Pins
Unplace Component
Bill of materials
Component list

One of the goals of the training program was to achieve a high rate of productivity in the design process. Our first round of training with a real MCM design took 12 weeks not counting the preparation time. When in business, this rate would not be economical since the design would have to be iterated over several times. Towards the end of training, we were able to improve productivity by 40%, but the actual time will vary widely depending on the complexity of the design.

3.6. Study of foundry processes

During the training period a major effort was undertaken to study fabrication processes prevailing in the industry to evaluate their impact on technology selection. This work involved a detailed understanding of the process variations across foundries and across technologies. The intent was to survey to what extent we needed to structure our design process to take into account foundry process variations.

This was a complex study of foundry processes because of the lack of standard terminology and limited disclosure of process details. Examples of MCM-L and C processes and their variations is shown below.

MCM -L fabrication process for multilayer MCM-Ls

- 1. Prepare copper clad laminate**

2. Apply photo sensitive resist, expose, develop, etch, remove resist

3. Inspect layer pair

4. Treat copper surface to improve adhesion

5. Laminate

6. Drill vias

7. Clean via holes

8. Metalize via holes with copper

9. Prepare copper surface of the panel.

10. Apply photo sensitive resist, expose and develop

11. Selectively metalize with copper, nickel, gold, etc.

12. Remove resist.

13. Etch off unneeded copper.

14. Remove individual MCM parts from panel.

15. Misc. operations--solder mask, nomenclature add pins, special metallization, etc.

16. Electrical test

17. Final clean

Impact on business development:

The impact of the study of foundry processes revealed that the business could split up MCM manufacturing into two or three separate activities (Substrate manufacture, assembly, and test) which could be contracted out to individual firms that specialized in these activities. As a result we formed a strategic partnership with a California MCM assembly firm, and identified a Minnesota firm for the substrate manufacture part (Unfortunately the firm went out of business shortly thereafter).

Impact of process steps on design

The impact of the design steps on the process steps are depicted in the table on the following page. Foundry processes have a lot of variation involving these steps, and the technical data regarding these steps are trade secrets.

The major components of the MCM-C process flow are depicted in the diagram following the table, after which a mapping is provided between the lower level substrate formation process flow and the top level process.

Design step	Layout	Routing	Discrete	Technology choice	Electrical	System design	Requirement	Cost
Fabrication step								
Cutting of green tape	x							x
Via punching		x						x
Metallization			x		p.delay			
Testing								
Stacking/Lamination		x						x
Sintering				x				
Ni/Gold plating					x			
Substrate elec test								x
Pin brazing						x	thermal rework testability volume	x
Chip attachment								
Module elec. test								x
Capping								x

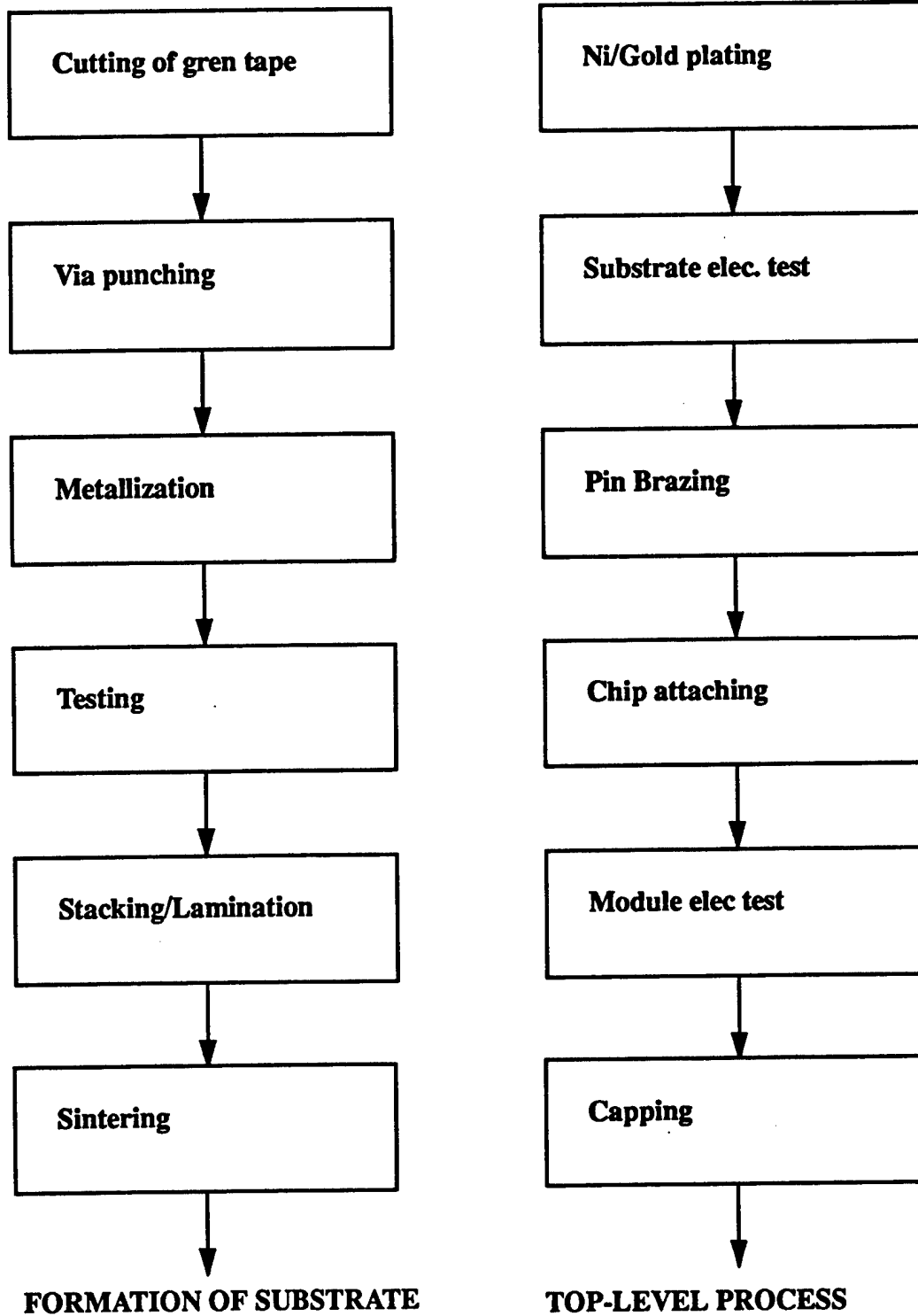
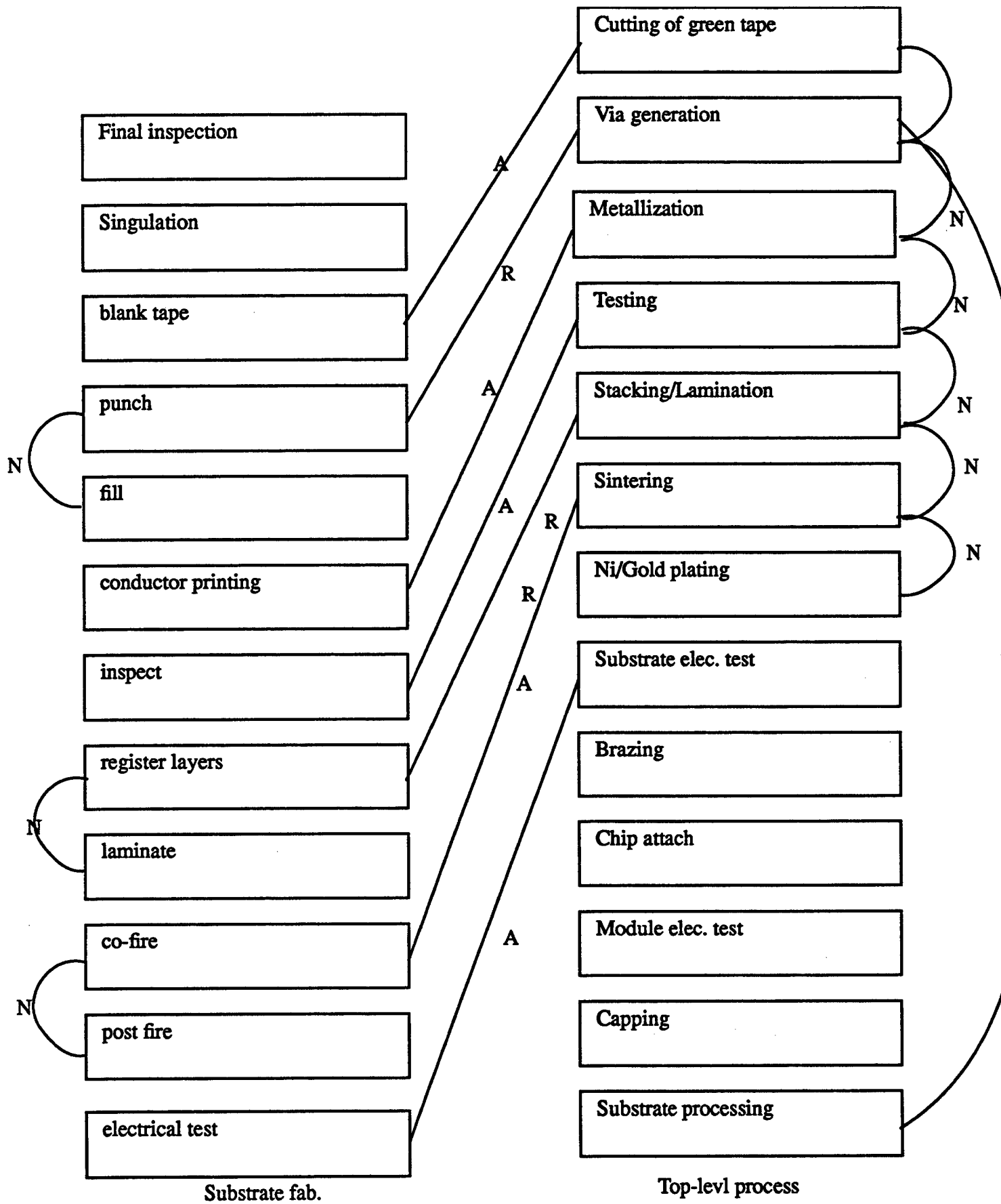


FIGURE 8. MCM-C fabricating process



3.7. Business development activities

We had extensive business development activity during Phase III. These activities can be described as: business setup, launching a publicity campaign, pursuing strategic alliances, and defining the competitive edge that we would have over other companies in the future.

The MCM design service was a new business model that required a clear assessment of the services that we could offer at a reasonable cost. We described our services to our potential customers in a clear and succinct manner as follows:

The design services include technology selection, layout design, foundry selection and interface to various MCM manufacturing facilities. The key feature of the services is to offer an unbiased evaluation of manufacturing capabilities of different MCM foundries in order to ensure a cost-effective and competitive MCM product. In addition it requires a commitment to reduce implementation risks and solve customer's MCM problems by evaluating alternative technologies, creating multiple designs and through strategic partnerships with a broad manufacturing base.

Technology selection

The availability of a broad range of MCM technologies combined with recent variations among these technologies, and the multiplicity of the fabrication choices, make the task of technology selection a challenging one. Vista has developed tools to carry out a thorough trade-off analysis between the application requirements and different technology and materials choices available. These tools guide the designer in making the selection of appropriate technologies based on application requirements specified in the form of input parameters. This step involves selection of substrate, dielectric, and metallization materials, die attach techniques, heat removal techniques, etc. The technology selection tools provide a simple and flexible mechanism to specify application requirements such as circuit performance, size, weight, environmental conditions, cost, etc., and simple user interaction to analyze partial results in order to converge to a suitable set of technology parameters.

Foundry selection

The selection of an appropriate MCM vendor is an important step in the successful implementation of MCM technology. Based on the technology selection results, Vista can use its expertise to make the selection of a suitable MCM foundry. Being an independent design service bureau, Vista can provide an unbiased evaluation about the manufacturing capabilities of different MCM foundries existing in the MCM market today. Vista has already established contacts with many qualified MCM manufacturers in order to collaborate with their manufacturing capabilities to meet its customers' MCM needs.

Layout design

MCM designs being process-driven, it is required that the design be performed based on the technology selected and design rules of the selected foundry. Vista can closely work with the MCM foundry in the design phase in specifying the design rules and other process specific data and to complete the MCM design.

Manufacturing interface

Vista design services include providing MCM foundry interface for substrate fabrication, assembly, test and rework. Since Vista is not tied to a single foundry, it can provide manufacturability analysis of different MCM foundry capabilities. In order to provide a cost-effective and competitive product, Vista can investigate the possibilities of subcontracting different components of manufacturing phase to separate MCM vendors. For example substrate fabrication and assembly can be subcontracted to two separate vendors. This can be done to use the manufacturing expertise of MCM foundries in some particular areas and at the same time provide customers a better product at a competitive cost.

Business setup

The business setup activities consisted of infrastructure development consisting of setting up the modalities of doing business with the customer, with bare die vendors, and with substrate manufacturing and assembly firms. A database of vendors was developed and telephonic contact was established to introduce our services.

Several bare die vendors were contacted and price and delivery time tables were discussed. An experiment was conducted in which price quotes for a set of four bare die were requested. The mean time to obtain a price quote was 4 days. One of the chips did not appear on any vendors die list that we contacted; and procuring the die would take a couple of months. An early lesson we learnt was that bare die vendors are geared for mass market bare die and appear to be reluctant to service requests for other bare die.

The design process from customer contact to hand over was defined, legal liabilities were determined, and technical data interchange with customer was also determined. Examples of how data are interchanged with customers and vendors are depicted in the following pages.

The first feasibility analysis report is used to record the status of the design service with respect to the customer. Eight milestones are defined and the start and completion dates are recorded in the report. The importance of this minutiae lies in the fact that it acts as a productivity measure. As the business grows there will be a need to hone our processes to reduce the time of completion of each step.

The second form is used to obtain technical data from the customer who often may not be aware of the extent of information needed to develop an MCM that best suits their application. The form includes a component list that the customer needs to provide. Some of the components may be packaged components for which bare die availability will have to be researched as part of our design services offerings. The third form depicts a typical request

for quotation that we use once design is complete. Of interest is the period of performance clause that we added following our experiences. Apparently, quantity per quarter is a strong determinant of price and some foundries will not provide price quotations without this data.

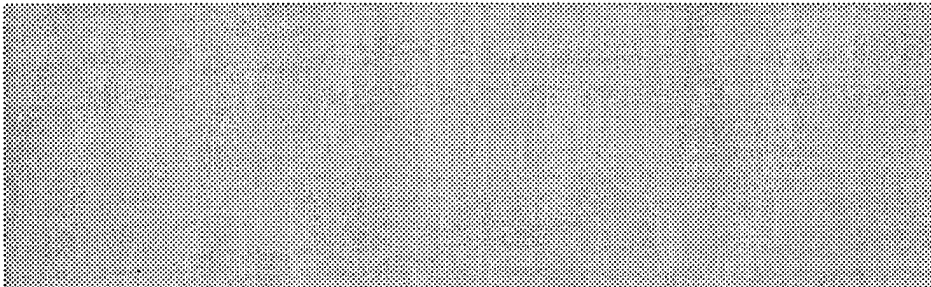
FEASIBILITY ANALYSIS

STATUS REPORT

Customer name:



Customer address:



Start date

Date Completed

1. Sign NDA with customer
2. Get input from customer
3. Check availability of bare die
4. Develop feasibility solutions
5. Send data to foundries
6. Prepare feasibility report
7. Conference with customer
8. Service completed

FEASIBILITY ANALYSIS

CUSTOMER INPUT FORM

Customer name:

Customer address:

1. Netlist

Format:

Filename:

2. Application (*check all that apply*):

1. consumer

2. computers

3. communication

4. military/avionics

3. Period of performance

When do you desire the first prototype:

Desired start date of manufacture:

Anticipated manufacturing schedule: *Fill table below*

Year or Quarter from start date	Quantity

4. Is there a specific technology (or technologies) that you prefer? *(check one or more)*

- 1. MCM -L
- 2. MCM-C - (HTCC)
- 3. MCM-C - (LTCC)
- 4. MCM -D
- 5. MCM-C/D
- 6. Other *(specify)*
- 7. No preference

5. Level of testing required* *(check one or more, except 8 and 9):*

- 1. Standard ASIC
- 2. Assembly JTAG
- 3. Assembly ICT
- 4. System Test
- 5. Performance testing
- 6. Functional testing
- 7. Limited functional test
- 8. Will perform own testing
- 9. Do not know, need guidance

* Some of these tests may require you to supply test vectors.

6. Will you supply your own die (*choose one*): Yes No Partially yes

7. MCM packaging: (*Choose one of the following*)

1. LCC

2. PGA

3. QFP

4. BGA

5. Custom

6. No preference

8. Maximum operating temp: degrees F

9. Operating frequency (Clock): Mhz

10. Special requirements:

Hermetic sealing required (*check one*): Yes No

Size (inch x inch):

Volume:

Shape:

11. Special operating conditions (*choose the first or either one or both of the rest*):

1. No special conditions

2. Will operate under high thermal stress:

Temp range:

Number of thermal cycles during lifetime:

3. Will operate under high mechanical stresses:

Will you perform your own acceleration testing (*select one*): Yes No

Request for Quotation

To _____

Address _____

Tel. & Fax _____

Please provide quotation for item(s) listed below:

Qty.	Description of item	Date required
1	Multi Chip Module per attached spec. Please provide cost breakdown for design (NRE), fabrication, assembly, and test (RE)	ASAP

Ship To _____

Address _____

Tel. & Fax _____

Date:

Purchasing Agent

This is not a purchase order

Specification:

Number of nets: 240

Module area: 2. sq.inch

Number of layers: 3 gnd., 1 power, 2 signal pairs, 1 critical net layer

Number of critical nets: 11

Operating frequency: 30 Mhz

Operating voltage: 5 V

Power dissipation: 3 W

Mounting: on PWB with approx. 200 I/O's.

Functional testing needed.

Anticipated award date: 1Q '95

Period of performance: approx. 400 units per year over 1995 and 1996

Component description:

Desc.	Part Number / Price	Qty.	I/O
Processor	Intel MQ82389 (\$265 each)	1	164
Controller	MR87C51FB-16 (\$240 each)	1	40
FPGA	XC4010 -5MQ208C (\$470 for 300)	1	191
Resistor (die)	PRN111242002G (\$22.40 each)	6	24
Resistor	M55342K06B249DM (60 cents each)	1	2
Cap.	250R11B103KV4 (25 cents each)	7	2
Transistor	MMBT2222A (47 cents each)	2	3

The business setup activities included visits to nearby foundries and fabrication companies to describe our business and to assess an industry standard pricing structure for our offering.

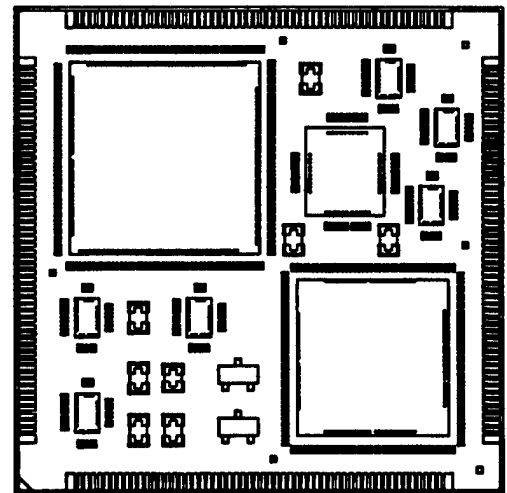
Publicity Campaign:

We collected a mailing list of about 200 names of companies who might serve as potential customers. This list included a number of bare die vendors, MCM foundries, electronic companies, PCB manufacturers, etc. We also obtained a list of addressees from the SBA which provided a list of contact names and phone numbers. These included mainly large defense corporations (Northrop, Singer, Hughes, etc.). In each case mailings were targeted at the divisional level. Letters were written to individual people after preliminary contact had been established and where their interest warranted such an approach. Follow-ups were conducted over the telephone. A sample letter is shown on the following page.

We also developed advertisements and brochures. A sample advertisement is shown below:

MCM design and brokerage services are offered to customers that need one stop assistance in choosing the right MCM technology, designing the MCM, and in interfacing to the MCM foundry. This new service addresses the needs of customers who wish to consider using MCM's to meet their packaging requirements of high performance, high density, and low cost. Services include MCM feasibility analysis, technology selection (L, C, D, C/D, substrate, die attach, thermal management, packaging, etc.), foundry selection, physical design, electrical and thermal analysis, bare die and package procurement, and interface to manufacturing. Feasibility analysis provides a quick first cut technology and cost analysis and is ideally suited for a management decision to proceed or pass. H

specialized custom tools are used at various stages to find best fit solutions for your requirements. The design service includes routing, signal integrity analysis, thermal analysis, design rules check and manufacturing rules check. For more information, contact Vista Technologies, Inc., 1100 Woodfield Road, Schaumburg, IL, 60173-5124. (708) 706-9300, e-mail: MCMinfo@VistaTech.com



Sample Brochure

November 29, 1994

Joe K.
M... Computer Corp.
199 R.... Rd.
Chelmsford MA 01824

Dear Joe:

It was nice talking to you on the phone. Our MCM design services group is now open for business. Our business model envisages customers coming to us to get advice on MCM technology, foundries, etc. We evaluate the customer's requirements, choose the technology, do the design, and contract out manufacture, assembly, and test to foundries that we help chose on the basis of competitive bidding. Or, if you prefer, we simply do the design work based on the design rules of the foundry of your choice. We work hard to keep abreast of the rapidly changing domain of MCMs so that we can provide you with the performance edge that you expect of MCMs.

As you requested, I have also enclosed a corporate bio. I have also enclosed some Vista product literature We would appreciate your putting us on your mailing list for your RFQ's.

Sincerely,

Dr. Sowmitri Swamy
Director, Packaging Design Group

Encl.

Strategic Alliances:

Another technique to increase business exposure was through strategic alliances. Three such alliances were pursued: one for substrate manufacture (ACSIST Associates), another for assembly and test (Amtech Electronics), and a third with a full range foundry (CTS Corp.) with a large defense clientele. Of these three, ACSIST is no longer in business.

Excerpts from the Amtech Electronics agreement are provided below to indicate their scope.

Excerpts (articles 1 and 4 only):

“1. Generating business opportunities:

Amtech agrees that it will generate business opportunities for Vista in the area or areas of Vista expertise related but not limited to MCM design. Vista agrees that it will generate business opportunities for Amtech in the area or areas of Amtech expertise related but not limited to MCM assembly.

4. Exchange of Technical data

Amtech agrees to provide technical data, under non-disclosure, regarding its assembly process to enable Vista to provide design services suitable for assembly by Amtech. Furthermore, Amtech agrees that it will inform Vista in a timely manner of any changes to its assembly process, capabilities, etc. which may materially affect Vista's ability to render its services.”

Maintaining the competitive edge

As part of the business development strategy, we are concerned about the viability of the business in the medium term due to the following reasons:

- a. The MCM market, though increasing, was and is a specialized niche market.
- b. Low cost and volume are the driving forces of the electronics industry, but MCM technology is not a low cost technology. Volume manufacturing has not been tried out.
- c. Our competitive edge based on technology selection can erode over time.

To further maintain our competitive edge we examined the vast PCB market. PCB manufacturers thrive on low cost manufacturing and high volume. Most electronics product cycles consist of several upgrades, and one the ideas that emerged was techniques to convert existing PCB designs into MCM designs.

PCB to MCM conversion

The MCM Design Service introduced by Vista offers customers technical solutions to packaging problems involving Multi-Chip Module technology. Current services include MCM technology selection, foundry selection, and substrate design services. The market for such services is rapidly expanding as the number of new MCM technologies, processes and foundries increases. The request for these services comes from customers with new designs; but enquiries from customers exploring the possibility of converting their existing Printed Circuit Board (PCB) designs to MCM designs is also increasing. Some vendors have already come up with MCM upgrades of the PCB designs as described in Section 4. This is due to the fact that MCM technology potentially offers a way to update existing PCB-based designs without re-designing the entire system. However, the factors governing MCM design are radically different from those governing PCB design. Converting an existing PCB design to an MCM implementation is complex, and involves extensive feasibility analysis. As a result, the conversion process results in very high NRE cost and design time. Consequently, a potentially large MCM market segment remains reticent about investing in MCM technology. Servicing this large potential customer base with cost and time effective conversion technology will strengthen MCM infrastructure and will further reduce MCM costs by broadening the market.

With current technology, the NRE labor costs of the PCB to MCM conversion process flow, described in Section 2, are very high. The steps in the process flow, not counting the design of individual MCMs, are estimated to take 15 man-weeks. The design of individual MCMs takes an average of 6 man-weeks per MCM. Thus for a typical PCB design conversion that involves the development of two MCM's, the total labor is estimated to be $15 + 6 + 6$ or 27 man-weeks. With the technology development that we propose below, we believe that the estimate of total effort for the same example can be cut down to 14 man-weeks. The improved productivity target can be met by the development of a small set of tools and utilities to improve circuit modeling, electrical simulation and estimation, and partitioning of the PCB netlist.

Technology development for PCB to MCM conversion will complement our current set of tools and enable us to provide a new cost effective design conversion service.

Design Flow for PCB to MCM conversion

Figure 1 depicts the three variants of the PCB to MCM conversion problem presented to the design service. These are: a. converting a single PCB to MCM's for performance improvement, b. converting multiple PCB's to MCM's for decreasing volume, or c. replacing part of a PCB with MCM's to make room on the PCB for additional chips. The key component of the conversion process in all cases is to substitute an interconnection of single chip-based packages with an electrically equivalent combination of multichip modules implemented in one or more technologies under severe form-factor, compatibility and cost constraints. In all three cases we expect to follow the process flow that is shown in Figure 2. Starting at the top left hand corner of the diagram, the PCB netlist is modified by replacing packaged components with bare die. A chip replacement strategy is used to handle the difficult cases arising due to non-availability of bare-die at a given test level, and

substitution of (passive) discrete components. The electrical characteristics of the circuit will have changed due to changes in the underlying parasitics of the un-packaged die and changes in the trace structure and substrate material of the MCM's. Since the latter are determined at a future step in the process by the MCM implementation, the simulation model is developed and simulated at various extrema. This way, the actual values will lie within the simplex described by these extrema. The result of the simulation is to generate a set of critical nets that cover the simplex. The netlist is then partitioned in the following step.

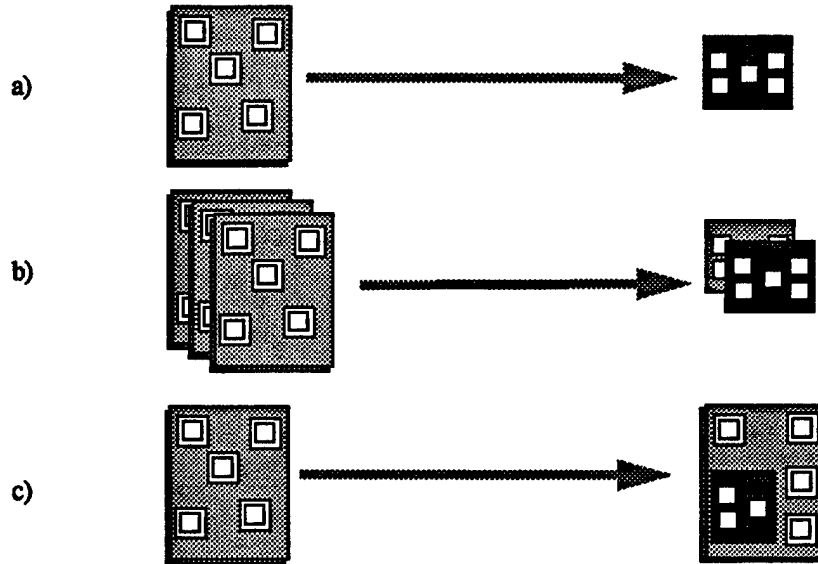


FIGURE 9. Variants of the PCB to MCM conversion problem

The partitions are developed taking into account bare die geometry, external I/O, testability levels, critical nets, and customer specified factors. The output of this step is a partition of the original netlist and technology and performance indices for each resulting MCM. There are several technology indices that are calculated, such as estimated module power, average circuits per chip, estimated module size, and average die area, etc. The technology indices form the input to the technology selection tool that is used to select the MCM tech-

nology (L, C, D, C/D, L/D etc.), substrate, dielectric, die attach, power dissipation and other relevant parameters.

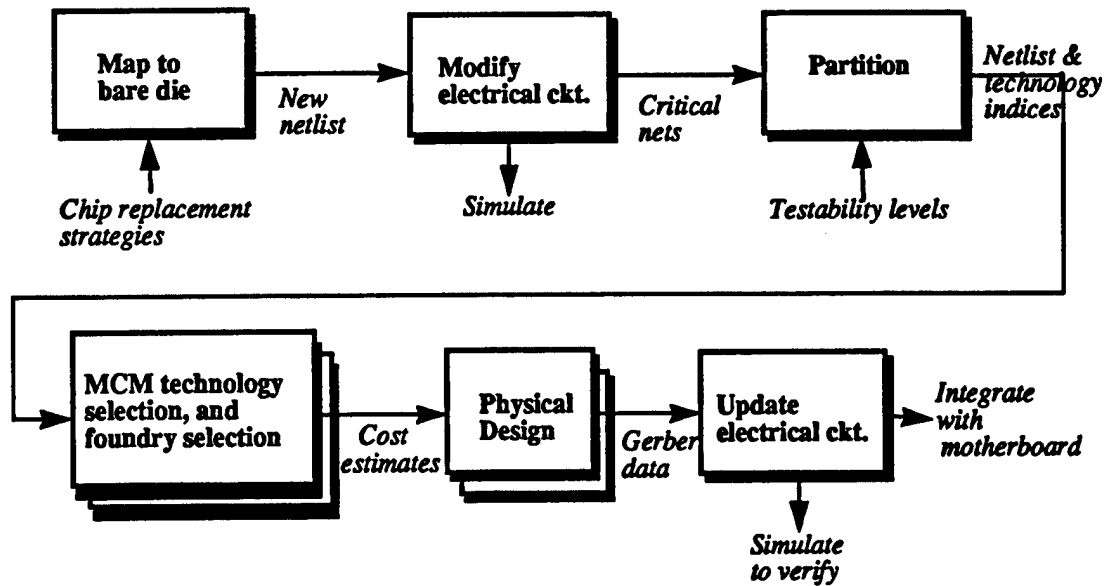


FIGURE 10. Design flow for PCB to MCM conversion

The technology indices and the MCM partitions drive the MCM Technology Selection and Foundry Selection tool as shown in the figure. This tool outputs a variety of technology and foundry choices that can be evaluated to obtain independent cost estimates. Technology selection and foundry selection are followed by physical design (placement, routing, thermal and electrical analysis) of each MCM individually. The next to final step in the process flow is the updating of the electrical model using the Gerber data generated during physical design. The updates include the use of accurate trace delays and the modeling of inter-module net delays including clock skew across modules. The electrical simulation of this model verifies the timing requirements set by the earlier simulation. The final step is the MCM package design for pin compatibility or integration with the motherboard. These steps may be iterated over several times when multiple solutions of technology indices are generated.

Technology development

The technology development effort needed to support the PCB to MCM design conversion process flow is described below:

Bare die analysis, electrical re-modeling and simulation

The first step in the feasibility analysis for PCB to MCM conversion is analyzing bare chip availability. In most cases, the availability coverage is substantially less than 100%, especially glue logic and low volume special purpose chips. Glue logic chips may be generally re-targeted to a new implementation strategy such as manufacturer-specific programmable

logic devices for which bare-die availability is assured. Special purpose chips may have to be replaced with components of equivalent functionality for which bare die is available. In cases involving performance upgrades, the replacement chip should be available as bare-die. The electrical model of the netlist is revised and re-simulated to ensure that the original timing requirements are met or exceeded. Such electrical models are critical in high frequency (> 60 MHz) applications. The result of the simulation is to redefine the critical nets in the design.

Research: Development of robust pre- and post-design electrical models of netlist to reflect transition to MCM's from original PCB.

Technology development: We need to develop a tool for quick electrical model generation of MCM netlist, The electrical model should account for the reduced inductances due to elimination of component package leads, change in trace structure and trace lengths, component replacements, and material changes. It should compute (the simplex of) extremum parameter values and be integrated with SPICE or other circuit simulators.

We also need to develop a database of bare-chip data that includes bare-chip equivalents, testability levels, physical data and suppliers

Partitioning, 3-D MCM implementations, and electrical simulation

This involves partitioning the netlist so that each part can be implemented on a separate MCM. For high density requirements coupled with low power dissipation, the partitioning step will consider 3-D MCM implementations as well. Partitioning may be necessary either to satisfy real estate needs or by a need to improve yields and thus lower costs. Partitioning will be based on the following criteria: form-factor requirements, chip I/O, maintaining integrity of critical nets, and testability levels of the bare-die. Passives/discrete components will be handled as part of the form factor constraints. Their small contribution to I/O will be neglected as a factor in I/O-based partitioning. Factors such as thermal management will be handled "downstream" as part of the physical design step.

The testability requirements are the only non - topological factor influencing the partition, but they are very critical since they directly affect yield. Bare die come with various quality levels reflecting the extent of testing (and thus, influencing the bare die price). At Vista's MCM design service, we have six categories of tested die: Bare die, Known tested die, Known good die, DC Probe tested, DC probe with LAT, AC/DC hot temperature probe with LAT, 100% die-level test with burn in. Since not all dies are fully tested, the partition must not freely mingle expensive known good die with inexpensive partially tested die in order to maintain acceptable first pass yield. Furthermore, rework and repair strategies to improve yields are direct correlated with the testability levels of individual die on the MCM.

One of the special applications of the partitioning tool will be the development of design techniques for 3-D MCM implementations. Currently, 3-D technology is being employed in design conversions of memory PCB's to MCM's (See Section 4). The tool will develop partitions to support stacked substrate MCM architectures in a manner that will minimize

connectivity between non-adjacent layers (with the exception of I/O). The advent of low power designs makes the power dissipation problem less acute, and our partition tool will allow stacked substrate architectures for non-memory PCB's to be designed.

A second phase of electrical modeling whose focus is the impact of partitioning on the electrical characteristics of inter-MCM and motherboard nets will be needed. Partitioning impacts the electrical characteristics in several ways: through the enhanced need for drivers to drive off-MCM or multiple MCM loads, through increased fanout, and through introduction of heterogenous trace structures in several nets. We will use the modeling tool developed in the previous section for this second phase of simulation.

Technology development: We need to develop a partitioning tool that partitions PCB netlists across several MCM's based on the four factors listed above, namely: form-factor requirements, chip I/O, maintaining signal integrity of critical nets, and testability levels of the bare-die. In addition, we need to develop a special partitioning technique for 3-D MCM's using the stacked substrate architecture.

Technology/foundry selection and physical design

Technology selection in the MCM conversion involves the selection of technology parameters for a given MCM partition. This module takes input parameters such as EstModuleSize, NumberOfChips, EstModuleSpeed etc. for a given MCM partition and outputs the suitable technology, materials, die attach technique etc. The technology selection tool has already been developed by Vista as part of the MCM design and fabrication services.

Technology needed: Current tool support is adequate

Related work and case studies

There are several examples of manufacturers marketing MCM upgrades of PCB designs. These are one-of-a-kind designs aimed at the PC market. Others have developed MCM design advisor tools that perform trade-off analysis at system level in order to determine packaging technologies for input components. The technologies covered by these design advisor tools include traditional and fine-line PCBs and MCM's. A brief description of related work is listed below.

Computer upgrades by Austek Microsystems

Austek Microsystems and MicroModule Systems have devised a solution to upgrading the aging laptop computers running on Intel 286 and 386SX processors. Using MCM technology, they have designed and assembled a five-component TI486SLC-based upgrades with the footprint of the original '286' or '386SX'. The modules have been designed to be field-upgradable by making them pin-compatible with the existing 286 and 386SX processors. The upgrading MCM includes 486SLC chip from Texas Instruments, Austek's cache controller chip with 16 kbytes of cache, a 25-MHz clock generator and a synchronizer. The MCM comes in two packages, one that plugs into a 286 socket and one that clamps over a 386SX and disables it. The latter configuration is necessary, since 386SX chips are usually soldered to circuit boards. The new modules can enhance performance by a factor

of six. This is possible by boosting the clock rate of the devices on the MCM and due to the shorter distances between the devices.

Dual C40 DSP by Texas Instruments

Texas Instruments Military Products Division developed a standard multichip module based on two SMJ320C40 digital signal processors in the first quarter of 1993. The new Dual C40 MCM reduced the board area by approximately 40 percent and power dissipation up to 15 percent over discretely packaged components. Size reduction was made possible by the assembly of multiple components in a single package and by limiting external package pins. About 500 connections remain internal to the module, so that a Dual C40 MCM requires only 408 pins. This can be compared with a single SMJ320C40 having 325 active pins. The new dual module also saves board space in its package dimensions. A 320C40 packaged in a ceramic quad flatpack measures 2 inches on a side. The Dual C40 MCM, with 10 active components measures 2.65 inches on a side.

Multichip System Design Advisor from MCC

MSDA (Multichip Systems Design Advisor) is a software tool designed to enhance the manufacturability and decrease the design risk associated with the selection of packaging technologies for integrated circuits. It performs trade-off analysis at system level for the selection of packaging technologies for input components. Trade-off results are output for various technologies including traditional, fine-line PCBs and MCMs. The input consists of two types: chip class and partition class. The chip class includes bare chip and other discrete components data whereas partition class includes the partition data and the netlists. The analysis is performed by concurrently computing different design parameters. These design parameters include thermal (internal and external resistances), electrical delays, physical (size, weight, interconnect routing requirements), reliability, testability, and cost performance metrics for multichip systems. The different technologies considered by MSDA are traditional and fine-line PCBs, low temperature co-fired ceramic, and thin-film.

3-D MCM Implementations

Three-dimensional packaging techniques have been employed in many memory as well as non-memory applications. There are three types of 3-D assembly techniques which have been used so far: packaged chips, bare chips, and multichip modules. By stacking chips on top of one another, designers can exceed 100% silicon efficiency compared to 2-20% for DIPs, PGAs and surface mount packages. Space and defense applications have been the main targets driving the developments of 3-D technology. The major applications for 3-D MCMs are all related to memory stacking: secondary cache memory, SIMM replacements for DRAM storage, solid state disk drives, and high density PCMCIA memory cards. Sun Microsystems has recently developed processor boards with cache memory stacks. Texas Instruments uses memory stacks on its Aladdin processor MCMs and dual C30 DSP modules. Irvine Sensors, nCUBE and NASA's Jet Propulsion Lab have developed a memory stack that is integrated into a compact node for a massively parallel processor computer.

Matsushita has developed a process using stacked TAB technology to build memory cards.

Another path that could be pursued in the still longer term is to incorporate the impact of processes on the MCM design through an extension to the current tool set. This involves implementing a Fabrication Advisor. The MCM market and technology is still in its infancy to make this idea viable in the near term, but with increased clock frequency the Fabrication Advisor could become indispensable to the design services business. the usefulness of this technology

Technology CAD

Technology CAD (TCAD) aims at predicting the impact of process and manufacturing technologies on semiconductor devices. It has long been used in IC design to achieve enhanced performance, reliability, and manufacturability. TCAD has been very useful in transitioning IC manufacturing to higher densities through improved device characterization leading to accurate device modeling and simulation. The new packaging technology revolution being ushered in by multichip modules (MCM) will include, among others, the same semiconductor processes used in IC manufacturing. The requirement of highly reliable design, significant process dependencies of the design, and the tremendous pressure to lower manufacturing cost while improving yield, are very critical to successful MCM manufacturing. Yet, TCAD development for MCM's has been very limited. This paper investigates the role of TCAD in MCM design and manufacturing processes. It establishes that TCAD can improve the reliability and manufacturability of MCM designs, by being able to predict the effect of specific processes on reliability, module yield, and estimated turnaround time, and will be very useful in the development of cost-effective module designs.

Our approach to TCAD for MCM's differs in many respects from IC manufacturing TCAD. We will heavily emphasize the impact of process and manufacturing steps on MCM design and technology selection. This is because MCM technology covers a large range of process and fabrication choices, each of which impacts design differently. We will focus on ways to have TCAD impact MCM's during the design phase. Our vehicle for doing so is the Fabrication Advisor. The key feature of the Fabrication Advisor is the ability to analyze the impact of process parameters on design parameters such as reliability, and optimize the match between technology data, user-specific requirements, design, and the available fabrication processing capabilities. It will perform a detailed fabrication feasibility analysis for a given set of design and technology data, early in the design process to ensure a reliable, cost-effective, high yield and highly manufacturable process-driven design.

MCM Technology CAD

Technology CAD has been used in the design of integrated circuits and technologies in order to achieve high performance, reduced cycle time, enhanced reliability, and increased

manufacturability [3, 7, 9]. Traditionally, the term TCAD has meant a set of tools that provide data of the physical characteristics and topography of the IC by simulating fabrication. Various TCAD tools such as process and device simulators [2, 4, 5] have been developed and are being used in many applications of IC design and technology developments. These TCAD tools were developed to minimize the physical experimentation time and to develop new IC technologies and have contributed to significant improvements in design and fabrication. We wish to investigate the role of TCAD in MCM designs in order to achieve similar benefits.

MCM's, being process and technology driven, are very prone to variations in these factors from design to design. MCM TCAD can address these process and technology dependencies in order to ensure high reliability, yield, and cost effectiveness. Properly designed and deployed, MCM TCAD can delineate the differences in processes, technologies, and materials on a specific design, and can enable the designer to make intelligent choices from the plethora of competing technologies, materials, and process alternatives, as illustrated in Figure 1. We view TCAD as an essential part of the MCM technology infrastructure that needs to be developed.

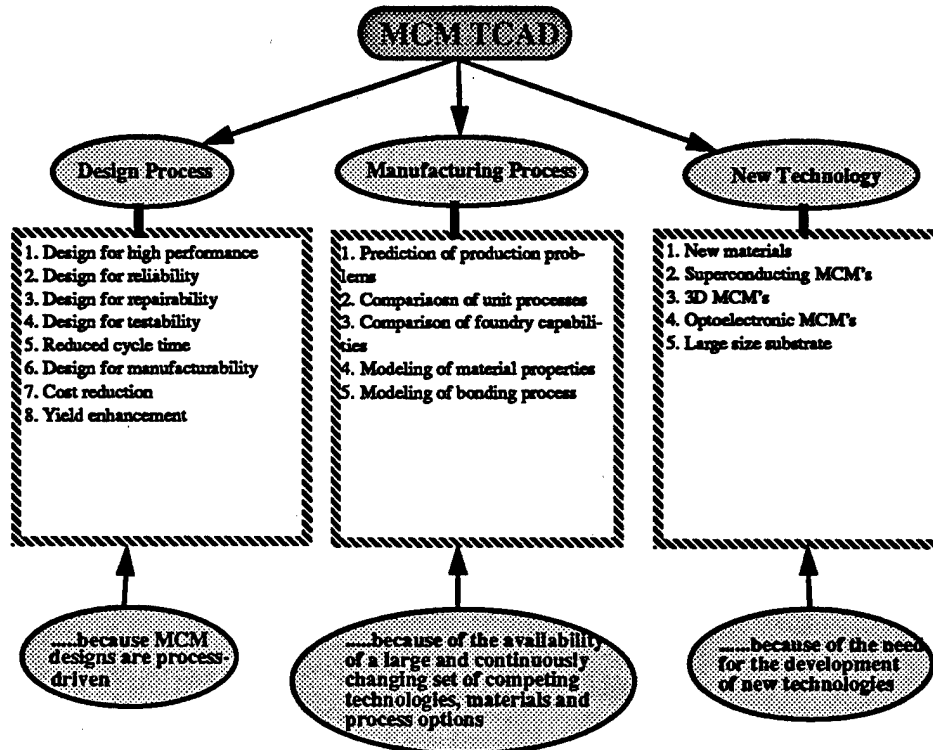


FIGURE 11. MCM TCAD processes

The three areas in which MCM TCAD can play important roles are:

1. Design Process

2. Manufacturing Process

3. New Technology Development

MCM TCAD IN THE DESIGN PROCESS

TCAD can be used to improve a number of design parameters such as performance, reliability, and repairability, as discussed in the following paragraphs:

Design of high performance MCM-based systems require a careful consideration of electrical and thermal issues. Usually, electrical and thermal designs are carried out by simulating MCM layout. However, data provided by the layout alone is not adequate for simulation, as it lacks the topography based information. By simulating fabrication, MCM TCAD can provide accurate topography information to be used in electrical and thermal simulations resulting in enhanced performance. Moreover, by providing physical based information on 3D substrate and layer models, a better control on the signal integrity problems such as crosstalk and skin-effect can be achieved.

Enhancement in performance must also be accompanied by a reliable design. MCM TCAD provides the ability to predict module reliability associated with a particular design before going to the fab. It can be used to predict the failure mechanisms of process related materials and mechanical stress induced due to thermal mismatch. Early analysis of these failure mechanisms in multichip module designs by using TCAD can increase the reliability factor by a significant margin.

In addition to enhancement in module reliability, TCAD can play a critical role in the design for repairability, which is considered one of the major factors for cost-effective products. TCAD can be used to analyze the substrate and interconnect structures to evaluate if the replacement or the repair of defective parts will be possible or not in a post-fabrication phase. Since repairability of a defective part in an MCM depends on the process employed, TCAD can efficiently be used to provide good estimation of the layout margins necessary for the repair of defective components after the MCM has been fabricated.

MCM TCAD can play equally important role in the design for testability (DFT), manufacturability and yield improvements. The technical evolution in MCM's allows the design of products so complex that they are virtually impossible to test. MCM technology can severely limit the available electrical test access. Therefore, design for testability is the key to cost-effective testing. Usually, testing has been divided into three categories namely parametric, functional and performance testing. The circuitry and the method employed for each of these has its own implementation requirements depending on the particular fabrication process. TCAD can be used to analyze the adaptability of particular test method in an existing fabrication process.

Even when the MCM technology which satisfies the application requirements has been designed, the actual fabrication of substrate and interconnect structures are subject to manufacturing variations. TCAD can evaluate process sensitivities on design parameters leading to a better estimation of the final module yield at a particular performance level.

TCAD can be used to generate foundry-specific design rules earlier based on design and process anomalies reducing design cycle time. TCAD can also be used to investigate the impact of process modifications to enhance the performance and capability of a technology resulting in manufacturable designs. Improvement in design, minimization of physical experimentation, and early feasibility analysis of fabrication processes can lead to a substantial yield improvements and cost reductions.

We can thus conclude that in the design process

TCAD predictions should be design- specific.

A corollary of this statement is that MCM TCAD has something useful to say for each individual design. This goal is significantly different from IC design, where predictions are device based. Any design that uses that device would have the same predictions. This critical capability makes interaction between MCM design software and TCAD extremely useful for the system designer.

MCM TCAD IN THE MANUFACTURING PROCESS

The availability of a large and continuously changing set of competing technologies, materials and process alternatives, make process selection a challenging task. MCM TCAD can be used to accurately predict production problems, analyze unit processes, and simulate a particular foundry-specific process flow without actually going to the fab, resulting in improved yields, turnaround time and considerable cost-savings. In this section, we briefly survey the role TCAD can play in the manufacturing process.

1. Accurate prediction of fabrication specific production problems (fabrication feasibility analysis early in the overall design process)

e.g. stacked via of x micron cannot be fabricated in y number of layers

An immature infrastructure, lack of standards and novel fabrication techniques, sometimes lead to an MCM final product which is either costly or does not meet the required specifications. MCM TCAD can accurately predict anticipated production problems by employing process simulation concepts. TCAD enables the designer to modify the design, or the foundry its target technology to avoid production problems. Manufacturing-related problems have been one of the main causes for costly final products. For example, a particular fabrication process may not allow the formation of stacked vias beyond a certain number of layers and this can be predicted well in advance by using TCAD tools.

2. Objective comparison of Unit Processes

e.g. MCM-L (lamine process) vs. MCM-D (deposited process)

MCM TCAD can be used to compare unit processes from different technology alternatives. Since there are various fabrication processes which exist for different MCM technologies, it becomes important to evaluate one unit process against the other. For example,

deposition of interconnect metallization may vary in different processes (MCM-L, MCM-C and MCM-D). This helps to optimize process related parameters by analyzing the effects of unit processes in different process flows.

3. Objective comparison of foundry-specific processing capabilities

e.g. MMS vs. nChip. The table on the left depicts the same patterning process implemented in two different ways.

Patterning using polyimide		
	Tooling	
Process step	Foundry 1	Foundry 2
<i>Apply</i>	Spin	Spray
<i>Soft bake</i>	Oven	IR conveyor
<i>Expose</i>	Contact	Projection
<i>Develop</i>	Spray	Ultrasonic
<i>Cure</i>	Furnace	Microwave

MCM TCAD can play a significant role in comparison of different fabrication processing capabilities. The number of fabrication processing capabilities for manufacturing MCM's is expected to grow as MCM's gain widespread applicability. The dependencies of the fabrication process on the foundry-specific equipments and materials necessitate a systematic comparison of foundry processing capabilities for a given design and user requirements. A given design can be manufactured in different MCM foundries resulting in different characteristics of the final product. A given process flow can be implemented in different foundries using slight variations of materials and foundry-specific equipments.

Use of TCAD in this application can result in process optimization for a given design and user requirements and consequently cost savings and enhanced manufacturability. MCM TCAD obviates the need to go to the actual fab and hence saving in time-to-market and revenues.

4. Accurate modeling of substrate, dielectric, interconnect structures for use in electrical/thermal simulations models

MCM designs require a thorough analysis of electrical and thermal behaviors. The electrical and thermal simulators available provide this information based on the layout. These simulation tools cannot provide the topography based information which is needed for accurate simulations models. MCM TCAD enables the designers to carry out the electrical and thermal simulations with accurate physically based information, thus resulting in better performance for the final module product.

5. Bonding process modeling.

The bonding techniques can have significant impact on the yield and the manufacturability of the final products. This becomes more important in view of the limited supply of known

good die (KGD) and repairability of the fabricated MCM. TCAD can be used to estimate the effectiveness and suitability of the different bonding techniques depending on the design and application requirements.

MCM TCAD IN NEW TECHNOLOGY DEVELOPMENTS

TCAD can also play a significant role in the design and development of new technologies using concepts such as process optimization and process enhancements. The actual fabrication and experimentation of new materials and processes involve substantial costs. TCAD can obviate the need for actual experimentation and aid in the developments of new technologies such as new substrates (e.g. diamonds), new assembly techniques (e.g. 3D MCM's), superconducting and optical interconnects. Some of the potential areas where TCAD can be used in order to design and develop new technologies are:

1. New substrate materials
2. Superconducting MCM's
3. 3D MCM's
4. Optoelectronic MCM's
5. Large size substrates

Key Component in MCM Design Services

In addition to providing a complementary role in Adabra framework environment, FA will also become an essential part of our proposed MCM design services bureau. FA automates the process of MCM fabrication facility selection, a key requirement in the overall MCM designs and an important business offering in an MCM design services. This will provide a bias-free evaluation of fabrication facilities for a given design by employing process simulation technique such as {nem design-driven process simulation}.

The use of TCAD in the design of integrated circuits is known for few years and have proved to provide many advantages. However, TCAD can provide even more significant benefits in MCM designs. This is true because of the fact that MCM designs are process-driven. Several technology choices and existence of multiple fabrication facilities make the use of TCAD in MCMs imperative. We have also proposed the development of a TCAD tool known as Fabrication Advisor, for use in MCM designs. The key function of Fabrication Advisor is to optimize the match between the technology design, user-specific requirements, design and the available fabrication facilities. Moreover, it predicts the fabrication difficulties for a given input technology design early in the design process. Thus, it helps the selection of a particular MCM fabrication process, being an integral part of an MCM design services bureau. This ensures a cost-effective, high yield and highly manufacturable process-driven design. Fabrication Advisor can play the role of a complementary tool in our Adabra environment and also a key component in the proposed MCM design services bureau.

Bibliography

CDB91 G. Chin, W. Dietrich, Jr., D. Boning, A. Wong, A. Neureuther, and R. Dutton, Linking TCAD to EDA - Benefits and Issues, Proceedings of the 28th ACM/IEEE Design Automation Conference, pp. 573-578, 1991.

AHW78 D. A. Antonodais, S. E. Hansen, and R. W. Dutton, SUPREM II-A program for IC process modeling and simulation, Stanford University, Integrated Circuits Lab. Tech Report 5019-2, SEL, 78-020, June 1978.

CLC C. L. Chu, Private Communication.

HH83 C. P. Ho and S. E. Hansen, SUPREM III-A program for IC process modeling and simulation, Stanford University, Integrated Circuits Lab. Tech Report 5019-2, SEL 83-001, July 1983.

DIA92 Daniel I. Amey, Overview of MCM Technologies; MCM-C, Proceedings of the International Symposium on Microelectronics, pp. 225-234, 1992.

DVD79 D. C. D'Avanzo, M. Vanzi, and R. W. Dutton, One-dimensional semiconductor analysis (SEDAN), Stanford University, Integrated Circuits Lab Technical Report G-201-5, October 1979.

LSN83 K. Lee, Y. Sakai and A. R. Neureuther, Topography Dependent Electrical Parameter Simulation for VLSI Design, IEEE Transactions of Electron Devices, vol. 30, pp. 1469-1474, 1983.

LMH92 Leo M. Higgins III, Material, Manufacturing, and Assembly Considerations for Laminate Substrate-Based Multichip Modules (MCM-L), Proceedings of the International Symposium on Microelectronics, pp. 216-224, 1992.

NSD84 S. R. Nassif, A. J. Strojwas and S. W. Director, FABRICS II: A statistically based IC fabrication process simulator, IEEE Transactions on Computer-Aided Design, vol. CAD-5, pp. 40-46, 1984.

ONSR80 W. G. Oldham, A. R. Neureuther, C. Sung, J. L. Reynolds, and S. N. Nandgankar, A general simulator for VLSI lithography and etching processes, IEEE Transactions of Electron Devices, vol. ED-27, August 1980.

PRD84 M. R. Pinto, C. S. Rafferty, and R. W. Dutton, PISCES II: Poisson and continuity equation solver, Stanford University, Integrated Circuits Lab Technical Report, September 1984.

SSP80 S. Selberherr, A. Schuz, and H. W. Potzl, MINIMOS-A two dimensional MOS transistor analyzer, IEEE Journal of Solid State Circuits, SC-15, no. 4, August 1980.

4. Adabra Tool Interface

This document describes the design of Adabra Tool Interface. Section 4.1 describes the architecture of Adabra Tool Interface. Section 4.2 details the methods of each module described in section 4.1.

4.1. Architecture of Adabra Tool Interface

- Figure 1 below shows the architectural layout of Adabra Tool Interface. Adabra Tool Interface consists of four modules, "ToolInterface", "Communication Channel", "Callbacks" and "ToolAddressTable" module. The dotted lines in Figure 1 indicate that the "tool interface" module contains the other three modules.

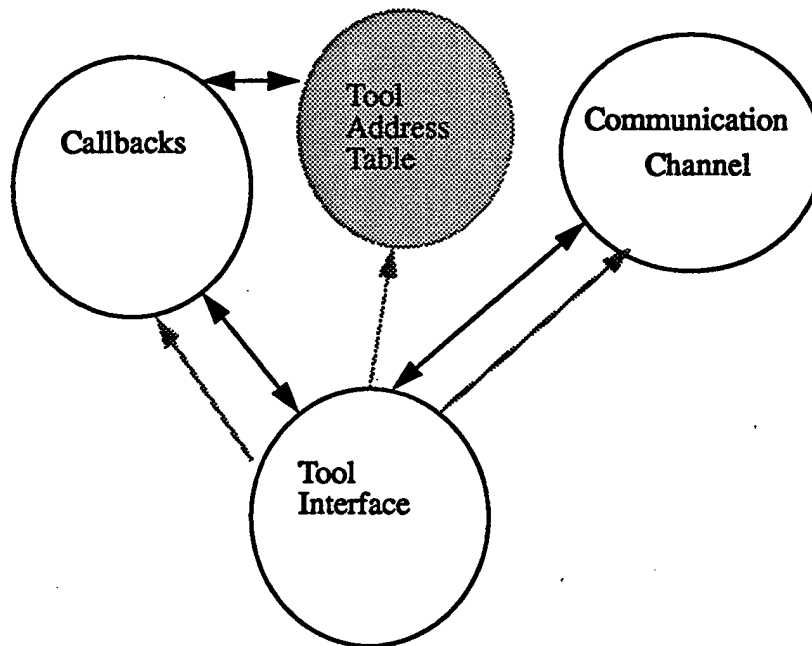


FIGURE 12. Architecture of Adabra Tool Interface

The Tool Interface module, is responsible for creating the motif widgets, and processing X events with the help of the other two modules contained in it.

The Communication module is responsible for communication between Adabra Tool Interface and Adabra Server. It is through this module that Adabra Tool Interface informs

the server to start clients, stopclients. Through the communication module, the tool interface module gets the configuration information of the tools running in Adabra.

The Callbacks module has the functions required to process the Xevents.

4.2. The Tool Interface Module

ToolInterface Methods

1. Create -

- Create the Communication Channel module, the ToolAddress Table and the allbacks module in that order.
- Create and Initialize the DataMembers

2. Delete -

- Delete the Communication channel module and the Callbacks module

3. Get glossiest -

- Get list of tools running in Adabra

4. Create and Display Widgets-

- Get the list of tools running in Adabra by sending a message to Adabra Server
- Create the Widget to be displayed using this list of tools and display the widget.

5. ToolInterfaceLoop

- Catch the X events and Process them.

Tool Interface Data Members

1. ToolAddressTable

2. CommunicationChannel

3. Callbacks

4.2.1. The ToolAddress Table

The ToolAddress Table is a table with two columns. The first column stores the name of the tool and the second column stores the address of the tool in the server. To communicate with a tool in Adabra server, the address of the tool is to be known. The methods in this module are

Create(list of tools, communicaiton channel address)-

For each tool in the list of tools, send a message on the communication channel and get the address of the tool. Create an entry in the table with the tool name and address obtained.

Delete()

Delete all entries in the ToolAddress table.

RetrieveToolAddress(Toolname)

Given the toolname, get the address of the tool from the ToolAddressTable.

Data Members

CommunicationChannel

4.2.2. The Callbacks Module

Methods

Create(ToolAddressTable, Communication Channel)

Delete()

Start_Callback(toolname)- Send a message to the Server to start the tool

Stop_Callback(toolname) - Get the address of the tool from the tool address table, and send it a message to stop

Get_ConfigurationCallback(toolname) - Send a message to get the various fields in the configuration.

Help_Callback() -

Exit_Callback() - Exit from

Data Members

ToolAddressTable

4.2.3. The Communication Channel Module

The methods in this module can be obtained from Propagator Design Document.

5. PPN Propagator

REQUIREMENTS DOCUMENT

This document lists the requirements for the PPN Propagator module in the rapid prototyping environment [SBIRQ92]. Section 1 lists the requirements of the PPN Propagator module. Section 2 describes the requirements for each sub module of the Propagator. Finally, section 3 gives an overall requirements chart for the Propagator.

5.1. Propagator Module

The requirements for the PPN Propagator can be divided into two main categories: requirements for the Propagator as a client in the environment and requirements in terms of its functionalities.

5.1.1. Requirements as a client in the rapid prototyping environment

The following are a list of requirements that must be supported for the Propagator to be a client in the environment.

Independent of tool integration-

The Propagator must be independent of the type of tool integration provided in the environment. By tool integration we mean a mechanism that provides for inter-tool communication and simultaneous access of all tools integrated in it. There are at least two different tool integration mechanisms based on whether the integration is interpreter-based (e.g. Vista's HyperWeb) or encapsulator-based (e.g. HP's Softbench). The Propagator module can only assume the presence of a server in the environment. The server is responsible for inter-tool communication by providing handles for each tool-tool communication.

Ease of portability across environments-

The Propagator module should be easily portable across environments. Portability across environments involves writing an interface module for the Propagator in the new environment. The interface module should have the actions for communication with external modules in the environment. By ease of portability we mean ease of writing

the interface module. A simple specification for the interface module should be provided. *Note:-* We need to formalize this as it will go as guidelines for porting across environments.

Communication-

The PPN and nodes in the Propagator should be able to communicate with other clients in the environment. Individual PPNs and their nodes must be addressable from outside the Propagator module. Communication between PPNs and external modules in the Propagator is carried out either directly or through the Propagator.

Multiple Propagators-

Facilitate the existence of more than one Propagator in the prototyping environment. In other words, there is a need to provide for a mechanism that will break up large Propagator modules into smaller ones and integrate them with the environment. This may be necessary to enhance the performance of the computation process. Thus, it is possible that there are more than one Propagators running simultaneously in the environment. Multiple Propagators can either run with one server or in a distributed environment which has multiple servers. *Note:-* Need to see if the design of the Propagator will be affected in a distributed environment.

5.1.2. Requirements in terms of the functionality

Interact with User Interface-

The PPN Propagator should be able to interact with the user-interface to obtain values for the parameters. The user must also be able to change the constraints for parameters via the user interface. *Note:-* Currently we assume that the user interface is an external module.

External Communication-

The Propagator should create communication channels for communication with other modules in the environment. Refer to the communication requirements at the module level.

Internal Communication-

Within the Propagator, PPNs should be able to communicate with each other.

Start Computation-

The PPN Propagator must create the run time PPN instances and start computation after obtaining relevant information from the external user interface module.

Execution Control-

The Propagator must facilitate control of PPN computation. For example, setting breakpoints, changing constraints and parameter values at nodes. Constraints and parameter values can be changed during run time.

PPN Querying-

The Propagator should be able to query PPNs for PPN specific information.

5.2. Requirements for each sub-module in the Propagator

The architecture of the Propagator module was discussed in the Propagator's design document[DESIGN92]. The various modules in the Propagator are communication channels, PPNs Channel Table and the Master Table.

5.2.1. Communication Channel Requirements

This module is responsible for external communication. It communicates with the external modules through *channel messages*. A *channel message* is a packet of information exchanged between the Propagator and other clients in the environment.

Number of Communication Channels -

Each node in a PPN computation can potentially communicate with a client in the environment. It is not practical to create a communication channel for every node in a PPN. Hence, it is reasonable to have a single communication channel for all the nodes in the PPN. Thus, a communication channel could be established for each PPN participating in the computation process. However, when there are multiple Propagators in the environment, the number of communication channels to be created in the environment may be very large. This calls for a design which provides an option for the number of communication channels created. A user can assign a separate channel to a specific PPN. PPNs without a separate channel then communicate through a master channel.

In summary, the following set of communication channels should be provided:

- A master channel for general communication with the environment.
- Dedicated channels for specific PPNs in the environment.

Special Messages -

The Propagator module receives requests for some specific actions within the Propagator. A list of messages that the Propagator receives are listed below:

Start Computation - Trigger PPN computation.

Propagate a value - Provide a value for some node in a PPN.

Stop Computation - Stop PPN computation.

Query a node in a PPN - Query a node in a PPN about node specific information stored in it.

Change, modify or delete a constraint at a node - Add, modify and delete constraints at nodes.

Set breakpoints- Provide breakpoints to control PPN execution.

Resume Propagation- Restart computation at the node where propagation stopped when breakpoint was set.

5.2.2. Initializer Requirements

The Initializer module of the Propagator must be responsible for creating communication channels, PPN instances, the Channel Table and the Master Table. The requirements for the Initializer module are listed below:

Create Communication Channels -

Create the necessary communication channels (Master and PPN channels).

Create PPNs -

Create the run time instances for PPNs and nodes. The initializer also enters the instances for the PPNs in the

Create Master Table -

Create the Master Table, a mapping of the PPNs created and their names.

5.2.3. The Master Table

The Master Table maintains the run time instances for each PPN and node in the Propagator. At any time the actual run time instance for a given node or PPN can be obtained from it. Based on these functions of the Master Table, its requirements can be listed as:

Create Master Table -

Create a Master table.

Update Master Table -

Add an entry into the Master Table.

Retrieve from Master Table -

Return the run time instance when given a PPN name and a node name.

5.2.4. The Scheduler

5.2.4.1. Required Functionalities

- Schedule a PPN for execution.- The scheduler picks a PPN for execution as long as they are available for execution.
- Enforce Parameter Propagation between PPNs - Based on the results from a PPN that was executed, Scheduler triggers Parameter Propagation.
- Enforce Conditions on PPN Interconnections - The Scheduler initiates Parameter Propagation based on the conditions on the Inter-PPN interactions.
- Should be able to implement several scheduling algorithms.

5.2.4.2. Scheduler Input

Here is a list of input needed by the Scheduler to schedule a PPN for execution.

- **Interconnection Table** - The Scheduler module interacts with the Interconnection table to determine the dependencies between PPNs. The scheduling algorithm which schedules a PPN for execution is based on PPN interconnections.
- **Master Table** - The addresses of the PPNs are required to enforce parameter propagation based on the condition interconnections.

5.2.4.3. User Interface Requirements

- **Optional PPN dedicated channels**- The user interface should provide the user with a list of all PPNs and request him/her to click on those PPNs for which a communication channel is desired.
- **Setting Breakpoints**- The user interface should obtain breakpoints for PPN execution from the user.
- **Primary Inputs**- The inputs for primary nodes in the Propagator have to be obtained.
- **Constraint Insertion/Deletion**- Constraints for parameters at a node in the network can be added and deleted with the help of the interface.

5.2.5. PPN Integrator Requirements

- **Interconnections Table**-The interconnections of PPNs is obtained from the user and entered into the table. This information is essential by the Propagator for creating PPNs.
- **PPN initialization Table**- Each PPN table is useful for creating nodes in the PPNs and setting their initial variables.

5.2.6. PPN Compiler Requirements

- **Channel variable**- Associated to each PPN object is a channel variable that gives the address of the channel assigned to it for external communications.

- ***Breakpoint variable***- When this variable is set propagation will stop, else propagation continues. This variable is part of the code derived by the PPN compiler.

5.2.7. Review of Requirements

In this section a review of the requirements for the Propagator is made possible through the chart shown in Figure 1. We do not show the requirements of the PPN Scheduler in this figure. This chart takes a layered approach to the requirements. At the top most layer we have the module level requirements. In the second layer we have the functionality requirements. Finally, in the bottom layer are the sub module level requirements which satisfy the functional requirements of the Propagator.

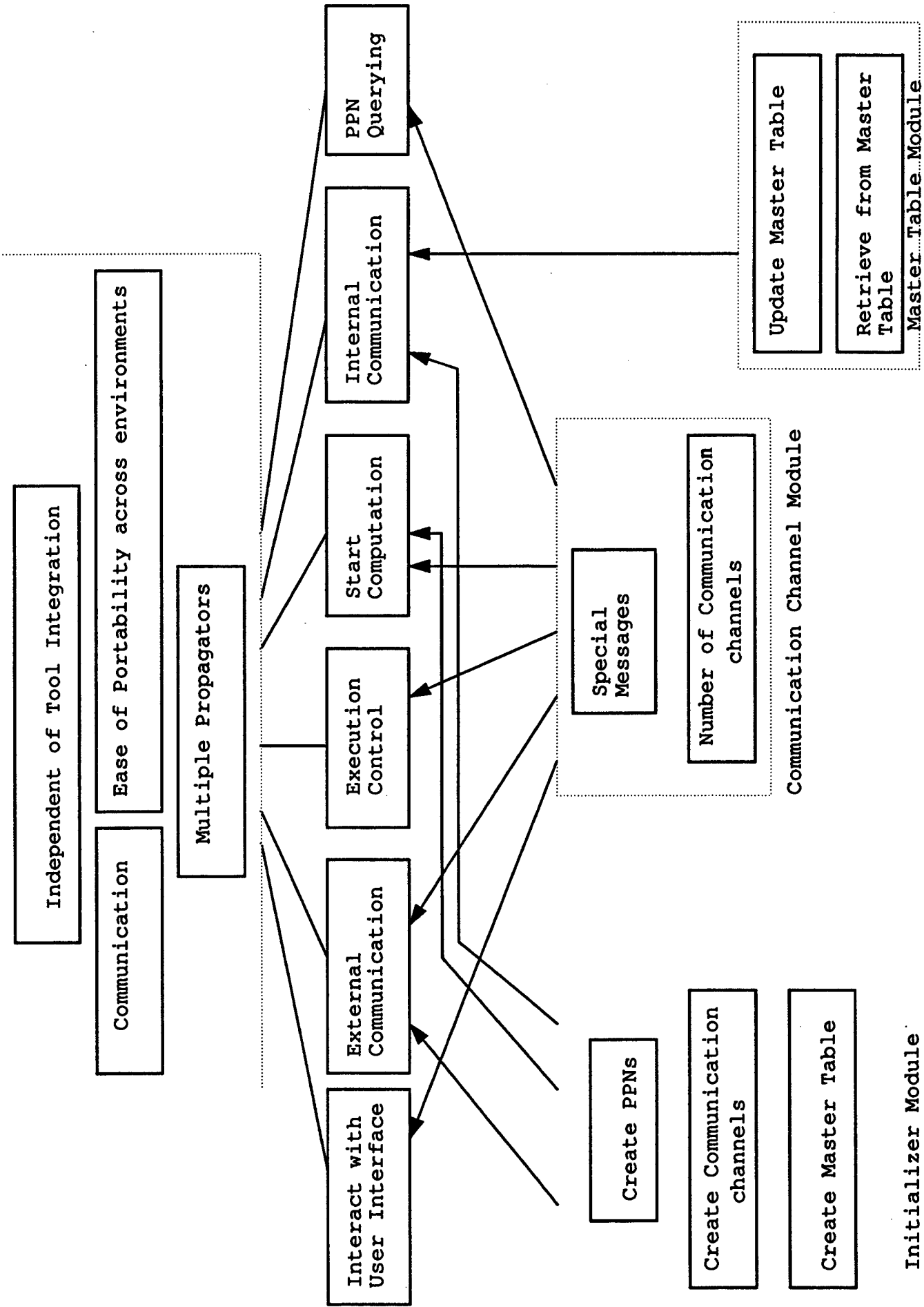


FIGURE 19. Architecture of Adabra Tool Interface

6. Propagator Design

Abstract This document describes the design of the PPN Propagator module in the rapid prototyping environment. Section 1 sketches the architecture of the PPN Propagator and describes the functions of each sub-module in the architecture.

6.1. Architecture of the Propagator

The Propagator consists of the following basic modules: Communication Channel, Initializer, PPNs, Channel Table and Master Table. The Communication Channel and the Channel Table are the modules concerned with external communication and processing of special messages received by the PPN propagator. The Master Table stores the addresses of the PPNs in the propagator. Inter-PPN communication is made possible by accessing the addresses of PPNs from the Master Table and sending messages to the appropriate PPN. The initializer module creates the channel table, master table, PPN run-time instances and the necessary communication channels. As discussed in the requirements document[PREQ92], there are two types of channels in the Propagator: the Master Channel and optional PPN Channels that are dedicated to a specific PPN. This is to facilitate direct communication between the PPN and external tools in the environment. PPNs which do not have a dedicated PPN channel communicate with the environment via the Master Channel.

Figure 14 shows the architecture of the Propagator module. We view each sub-module as an object. From the architecture we can derive four different classes of objects- Communication Channels, PPNs, Operators, Master Table and Initializer. The control and data flow between these objects is based on the messages received by the propagator module and the action it takes for each message received. The Initializer module creates

the other modules in the Propagator. (Shown by solid arrows from the Initializer to all the other modules.) The Channel Table and the Communication Channels request the .

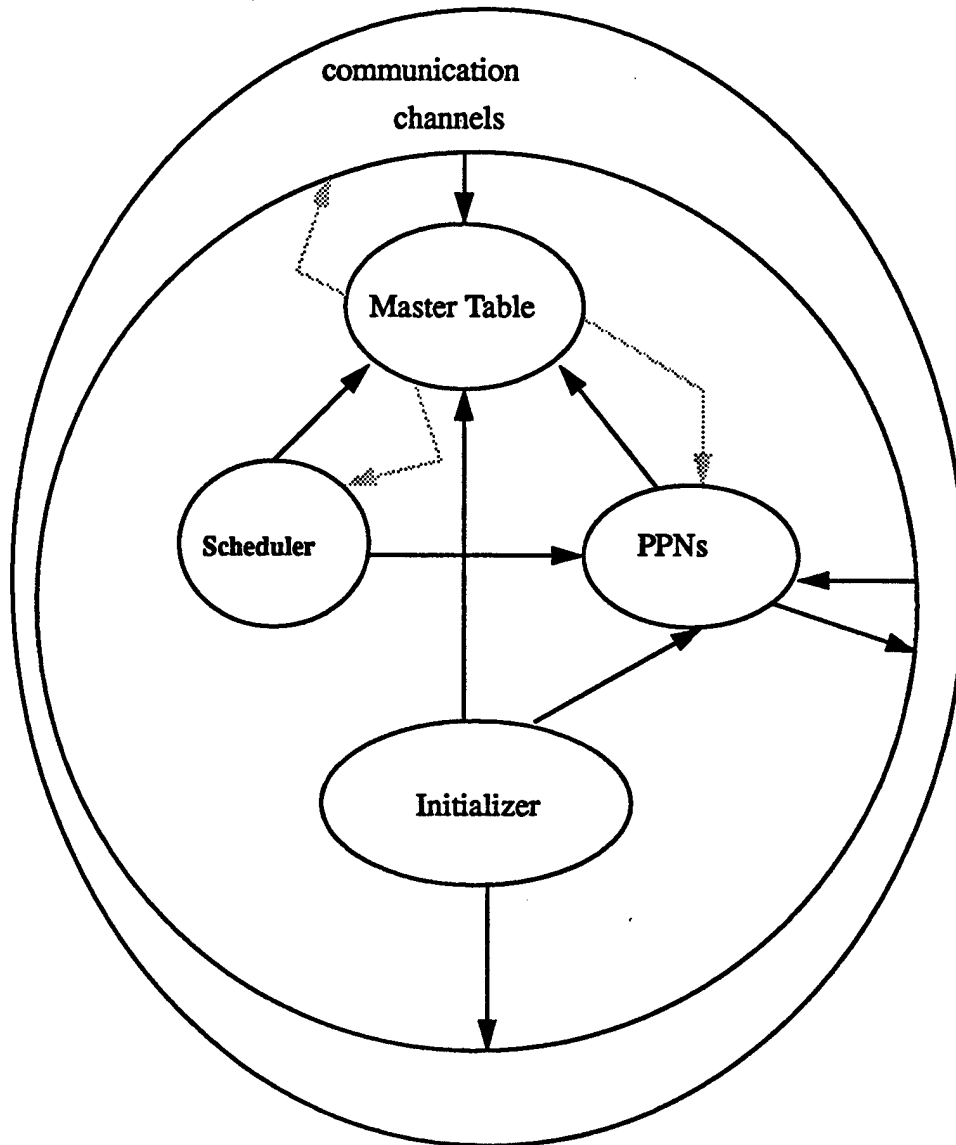


FIGURE 14. Architecture of the Propagator

Master Table for the address of the PPN it wishes to communicate with. (Again shown by a solid arrow.)The Master Table responds to these requests with the address of the appropriate PPN. (Response of the Master Table is indicated by the shaded arrows.)

When the Propagator module starts up, it first creates the Initializer sub-module, which in turn creates the Channel table, Master Table, PPN's and Communication Channel sub-modules. This is depicted by the arrows from the Initializer to other sub-modules in Figure 13. Initially, only the MasterChannel is created. Figure 14 shows the action taken by the Master Channel when it receives a message for creating more channels from the user.

When the Master Channel updates the channel table, with the addresses of the channels just created, the channel table sends the address of the PPN dedicated channels to the respective PPNs as shown in Figure 15. Each PPN stores the address of the channel dedicated to it on receiving the SetChannelInfo message. It should be noted here that the channel table is a persistent module. This is because we wish to provide the user with the feature of creating new channels and assigning them to PPNs interactively (i.e., at any point in the parameter propagation.) The user can change the channels associated with PPNs at any point of time during execution.

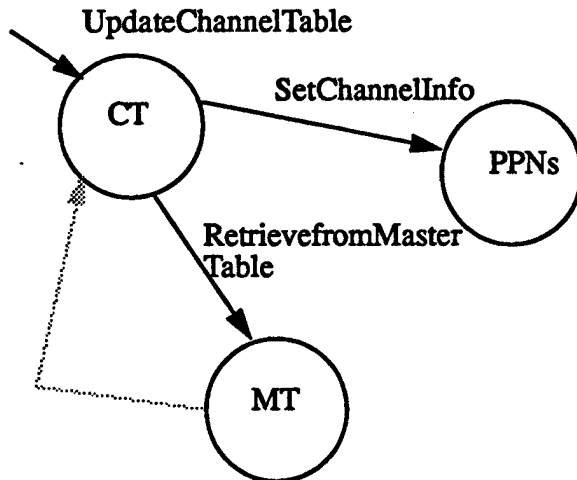


FIGURE 15. Action taken by the ChannelTable on receiving *UpdateChannel* message

If the messages on the communication channels (master or PPN dedicated channels) is to propagate a value to a PPN or a node, set a value for a node, change or modify constraints at a node, or to set break points, then the channel object retrieves the actual PPN or node instance from the Master Table and sends an appropriate message to the PPNs

by placing the message in the PPNs internal message queue. This is depicted in Figure 16.

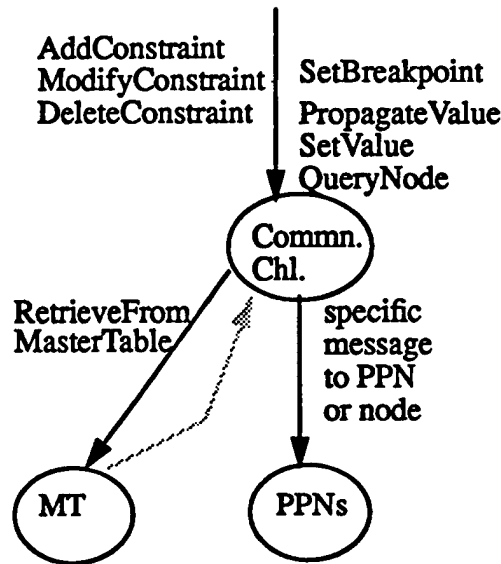


FIGURE 16. Actions taken by a Communication channel on receiving special messages

A node in a PPN can talk to external clients via the communication channels dedicated to it. It does so, by placing the message to the external client on the channel dedicated to it.

Besides the basic modules discussed above, the propagator has a top level module not shown in Figure 14. This module is responsible for creating the initializer object and sending it a start message. This module executes in a loop, processing messages pending on each channel, choosing a PPN for execution and initiating its execution. A scheduling algorithm to choose a PPN for execution is needed. Scheduling a PPN for execution depends on the availability of input for execution and satisfying certain execution criteria given by the user. The Scheduler module shown in Figure 14 is responsible for scheduling PPNs for executions. This module is created by the top level module in the Propagator.

The input to the propagator module are two tables, called the PPNTTable and InterConnectionsTable. The PPNTTable gives the PPNs, and the number of instances of each PPN Class that need to be created by the Propagator. The InterConnectionsTable gives the interconnections between the PPNs specified in the PPNTTable. The InterConnectionsTables may contain references to PPNs in another Propagator.

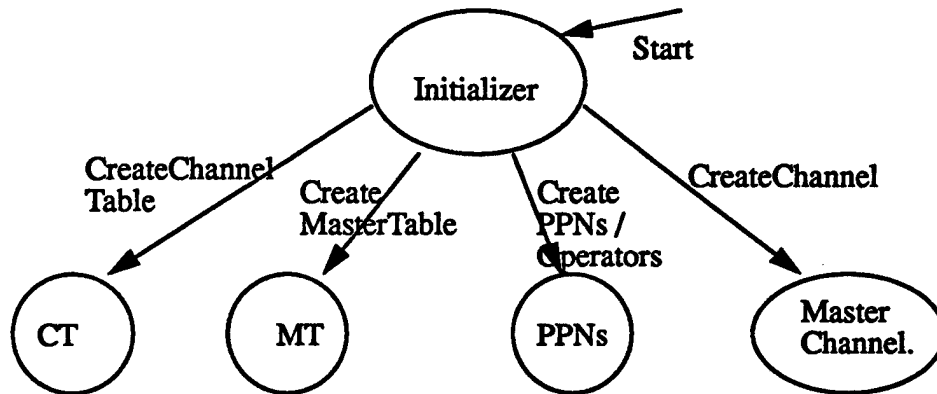


FIGURE 17. Actions taken by the Initializer on receiving a *start* message

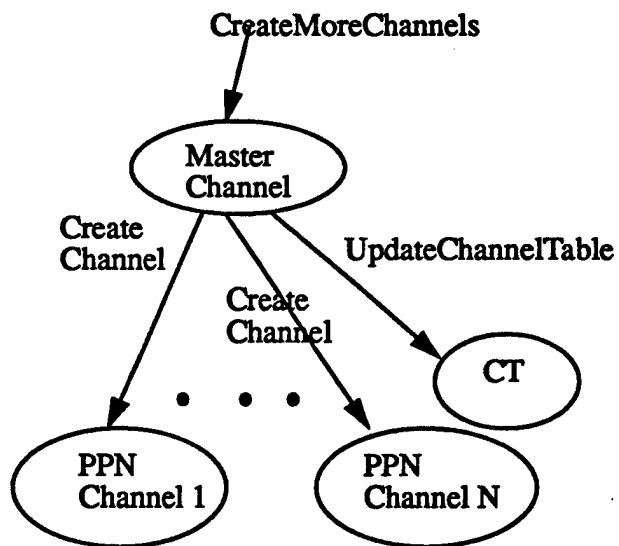


FIGURE 18. Actions taken by the MasteChannel on receiving *CreateMoreChannels* message

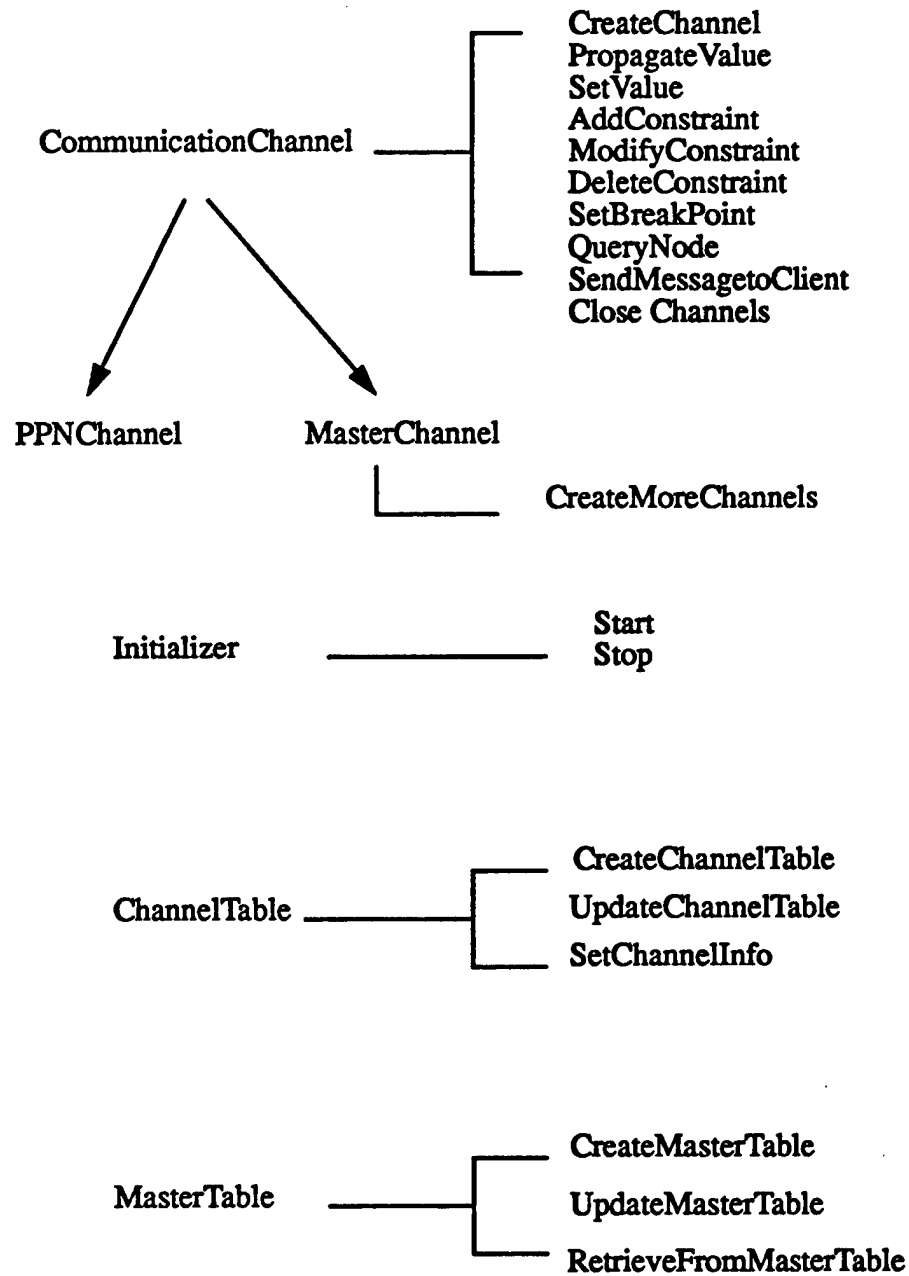


FIGURE 19. Propagator classes and their methods

6.2. Design of Propagator Sub-Modules

6.2.1. Communication Channel Module

This sub-module is responsible for external communication. It communicates with other external modules in the environment through *channel messages*. A *channel message* is a packet of information exchanged between the propagator and other clients in the environment.

Each node in a PPN computation can potentially communicate with a client in the environment. However, it is not practical to create a communication channel for every node in a PPN. Therefore, it is reasonable to have a single communication channel for all the nodes in a PPN. All communications to the nodes in a PPN are buffered in this channel in the order of their arrival. Thus, for each PPN participating in the computation process we can establish a communication channel. The user is to decide whether a particular PPN should have a channel dedicated to it or not, depending on the users assessment of the communication load between the PPN and other external modules. This information is provided through the user interface. PPNs without a dedicated channel, communicate with external modules via a Master Channel. A Master Channel is always created and is an essential module of the Propagator. Special messages to be processed by the propagator are placed on the master channel by external modules.

We shall now describe a communication channel object class in C++, whose methods perform the functions of communication channels. We define a `CommunicationChannel` class, a `MasterChannel` class and a `PPNChannel` class. `MasterChannel` and `PPNChannel` are sub classes of `CommunicationChannel`. The distinction between a Master Channel and PPN dedicated channels is that, a PPN dedicated channel does not receive all the messages received by the Master Channel. For example, only the Master Channel receives a message from the user interface outside the propagator to create PPN dedicated channels. Also, only the Master Channel sends a message to the framework to register the PPN dedicated channels just created. The methods in `CommunicationChannel` class are `CreateChannel`, `PropagateValue`, `SetValue`, `AddConstraint`, `ModifyConstraint`, `DeleteConstraint`, `SetBreakPoint`, `QueryNode` and `TalktoAnotherClient`.

The `CreateChannel` function creates a channel for communication (A channel could be a socket, a file, a data structure etc.). `PropagateValue`, `AddConstraint`, `DeleteConstraint`, `ModifyConstraint`, `SetBreakPoint`, `ResumePropagation` and `QueryNode` messages on the channel have the following action- retrieve PPN or node instance from the Master Table and send a corresponding message to the instance.

A PPN node can communicate with other tools in the environment during computation. The node which desires to communicate with the external client places an appropriate message on the communication channel dedicated to it. This is done with the help of the Channel Table. The `TalkToAnotherClient` method of the communication channel class,

message	actions
SetValueofNode(<i>PPN</i> , <i>node,args</i>)	Retrieve address of <i>PPN</i> from the Master Table and send it a message to set the value of <i>node</i> .
PropagateValueToNode (<i>PPN,node,args</i>)	Retrieve the address of <i>PPN</i> from the Master Table and send it a message to propagate <i>args</i> to <i>node</i> .
QueryNode(<i>PPN,node</i>)	Retrieve the address of <i>PPN</i> from the Master Table and send it a message to Query <i>node</i> .
SetorModifyConstraint (<i>PPN</i>) DeleteConstraint	Retrieve the address of <i>PPN</i> from the Master Table and set the constraint field of <i>node</i> appropriately.
SendMessageToClient (<i>client-class,type,message</i>)	WriteChannel(<i>client-class, type, message</i>)
SetBreakPoint(<i>PPN,node</i>) or ResumePropagation (<i>PPN.node</i>)	Retrieve the address of <i>PPN</i> from the Master Table and set the status field of <i>node</i> in <i>PPN</i> to STOP or CONTINUE
CreateMoreChannels (<i>PPN_list</i>)	For each <i>PPN</i> in <i>PPN_list</i> check if <i>PPN</i> has a dedicated channel. If <i>PPN</i> already has a dedicated channel, close the existing channel and create a new <i>PPN</i> Channel.

FIGURE 20. Methods of Communication Channel

retrieves the channel address from the channel table and places the message to be sent to the external client on it.

Besides the messages described above, a MasterChannel can also receive CreateMoreChannels method, and RegisterPPNChannels method. A CreateMoreChannels message received by the master channel results in sending a CreatePPNChannels message to the initializer. The initializer creates more channels and intimates the server by placing the message RegisterPPNChannels on the master channel. These methods belong only to the MasterChannel class and are not available to the PPNChannel class as shown in Figure 19.

When the Master Channel receives a message for creating more channels, it creates an instance of PPNChannel class. Each channel created is registered in the Channel Table. After creating the required number of channels the initializer writes a message to inform the external modules about the PPN dedicated channels just created.

Note that the list of messages received by the Propagator module, is by no means exhaustive. New messages to the Propagator module can be added by writing methods for them in the corresponding class object. For example, messages to modify PPN execution, like ExhaustBacktrackNode, ExhaustBacktrack, ExhaustBacktrackandAccumulate force PPN execution so that all feasible solutions are obtained.

6.2.2. Initializer Module

The Initializer sub module of the Propagator is responsible for creating the other sub modules in the propagator. The Start method of this class creates the Channel Table, Master Table and Master Channel Objects. The sequence of actions in this method are summarized in Figure 21.

Message	Sequence of Actions
Start	Create an instance of MasterChannel Create an instance of ChannelTable Create an instance of MasterTable Create PPNs and their interconnections
Stop	Delete MasterTable, ChannelTable and Channels

FIGURE 21. Methods of the Initializer

6.2.3. Channel Table Module

The Channel Table object stores the address of the communication channel for each PPN which has a dedicated channel for its communication. The Channel Table class consists of three methods: **CreateChannelTable**, **UpdateChannelTable**, and **SetChannelInfo**. The **CreateChannelTable** method is called with the master channel as an argument. Thus, until any PPN dedicated channels are created, all PPNs use the master channel for communication with external modules. The **UpdateChannelTable** method enters an entry into the channel table. The **SetChannelInfo** method sends the addresses of the special channels created to the PPNs. These PPNs then directly communicate with external clients by placing messages on these clients. The table in Figure 22 summarizes the methods of the Channel Table class.

Message	Sequence of Actions
CreateChannelTable	Create Channel Table. (maintain CT as a record of PPN name and channel_id.)
UpdateChannelTable	Add an entry into the table.
RetrieveFromChannelTable	Search table and return channel_id.
SetChannelInfo	For each PPN in table, obtain PPN instance from Master Table and set its socket field to channel_id.

FIGURE 22. Methods of ChannelTable

6.2.4. MasterTable Module

A Master Table Class is defined for the Master Table. The Master Table object stores the run time instances for the PPNs and nodes in the propagator. The methods in this class, are **CreateMasterTable**, **UpdateMasterTable** and **RetrieveFromMasterTable**. The **CreateMasterTable** method creates the Master Table object. The **UpdateMasterTable**, method adds an entry into the master table. The **RetrieveMasterTable** method takes a PPN name as input argument and returns its run time instance. These methods are summarized in Figure 23.

Message	Sequence of Actions
CreateMasterTable	Create Master Table.(maintained as a record of PPN_name and instance address.
UpdateMasterTable	Add an entry into the MasterTable.
RetrieveFromMasterTable (PPN)	Search through table for <i>PPN</i> and return its instance address.

FIGURE 23. Methods of MasterTable

6.2.5. Scheduler Module

The function of the Scheduler module is to determine the PPN to be executed at a given point of time. The Scheduler looks up the PPN InterConnection Table to determine the condition on the interconnections across PPNs. Based on these conditions a PPN is scheduled for execution.

Issues to be considered while scheduling PPNs for execution-

- n Time of completion of a PPN computation is not finite.
- n The InterConnections across PPNs have conditions associated with them. Based on these conditions a PPN gets scheduled for execution.
- n If PPN execution is preemptive, nodes that have completed their processing will have to be stored in the PPN, so that the next time the PPN gets scheduled for execution, processing begins at the node where execution was halted.
- n Input to the Scheduler- a list of PPN instances in the Propagator and the interconnections between them.
- n Output of the Scheduler - a PPN name for execution.

The interaction between the Scheduler and other relevant modules in the Propagator are shown below.

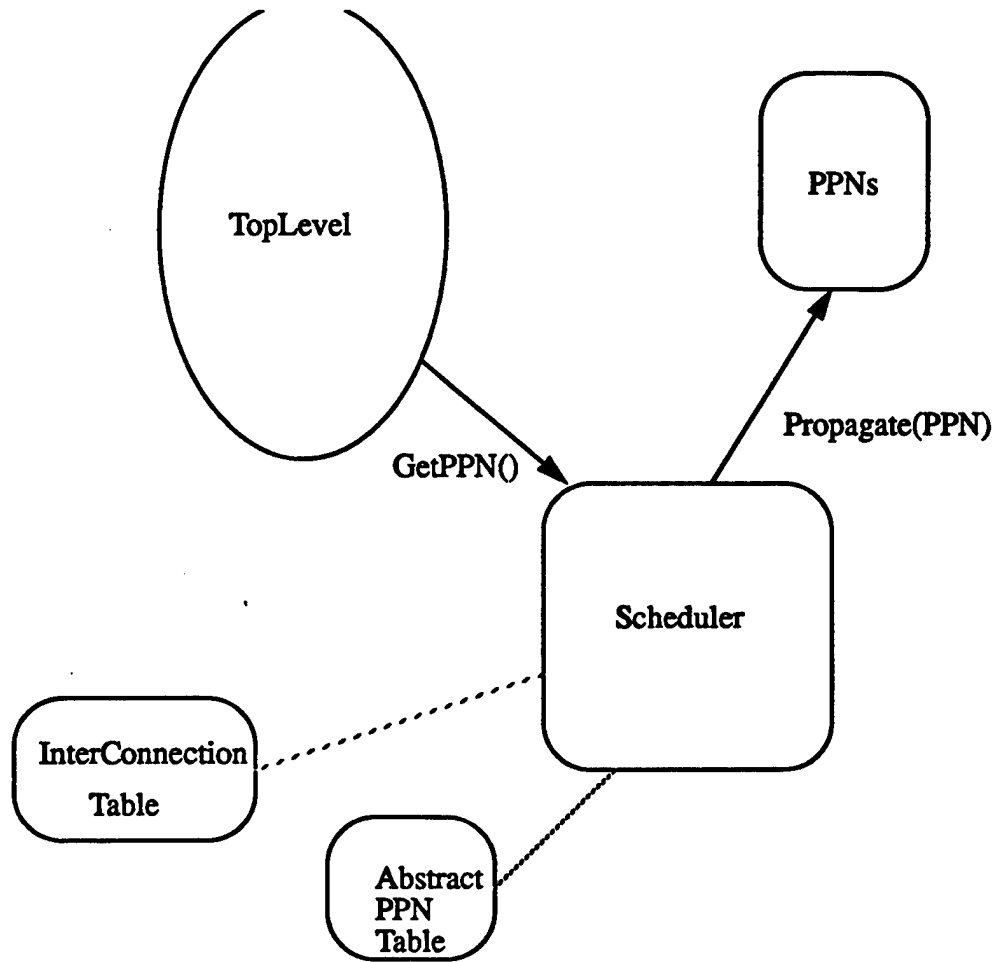


FIGURE 24. Interaction between Scheduler and other Modules in the Propagator

Figure 25 shows a control flow diagram for the toplevel module in the Propagator that involves the Scheduler module. The toplevel module does some initialization before interacting with the Scheduler.

Here, we show a main loop where the Propagator first reads all its input channels, and passes control to the Scheduler module. The Scheduler returns a PPN for execution. The Propagator executes this PPN, reads the input channels and passes control to the Scheduler again.

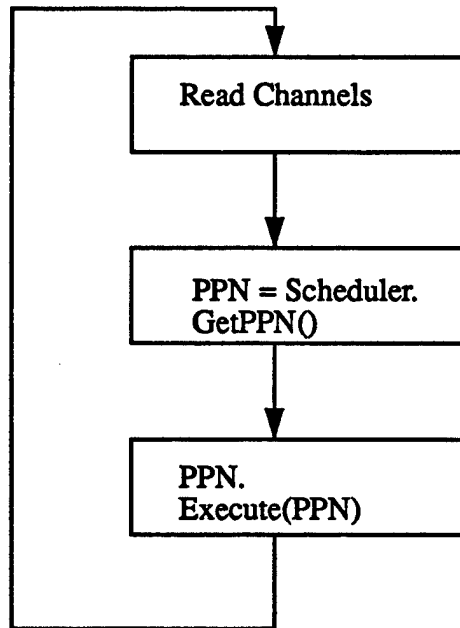


FIGURE 25. Control Flow for the Toplevel Module of the Propagator

An assumption made here is that PPN gives control to the toplevel at regular intervals, thus preempting its execution. By adopting such a preemptive scheduling, we have achieved the following

- Input channel is read more often, and hence queries can be processed immediately.
- All PPNs get a chance to execute as much as possible.

6.2.5.1. Scheduler Class Definition

Member Functions

1. **Constructor Function** - This is the constructor for the Scheduler object. It does some initialization.
2. **GetPPN** - Get the next PPN from the Schedule (by calling the Scheduler function).
3. **Scheduler Function** - Stores the algorithm for determining a Schedule. We will adopt a breadth first schedule. First a Level Graph is created by looking up the interconnections from the InterConnectionTable. A Schedule List is prepared, based on the levelgraph, such that AbstractPPNs with a lower level number, occur earlier in

the list. Each time the Scheduler function is invoked an AbstractPPN from the list is returned in a round-robin fashion.

Data Members

InterConnection Table, AbstractPPNTable, LevelGraph

Note that the Scheduler treats Operators and PPNs identically. The Scheduler picks up either a PPN or an Operator from the Schedule and hands it over to the TopLevel Module. The toplevel module, executes the AbstractPPN.

6.2.6. Multiple Propagators

Since PPN Networks are typically very large, it is useful to break down the network into smaller networks, such that each sub network is part of a different Propagator. Each Propagator executes as a separate process in the framework. Communication between PPNs is via the Master Channels in each Propagator.

Below is a list of decisions made to incorporate Multiple Propagators in our design.

Abstract PPN's Do Not Have Private Channels

Due to the Complexity of PPN Channels when there are multiple Propagators in the Framework, all PPNs now communicate through the MasterChannel. Moreover, in the earlier scheme where Abstract PPNs have private channels, we did not foresee any performance gain.

Abstract PPN's not concerned with the destination node during propagation.

It is task of the channel associated with the PPN, (here MasterChannel) to determine the address of the destination node for propagation.

MasterChannel determines the propagator to which the destination node belongs to and does the following:

- n** If address of destination node not in the current propagator, MasterChannel queries its client object to determine the physical address of the other propagator, and then communicates with the destination node.
- n** Otherwise, the propagate message is added to the Message queue of the PPN in the current propagator, to which the destination node belongs to.

Impact on the Server/Client Registry

1) Each Propagator is associated with a logical name (The name as appears in the inter-connection table.) Every propagator, registers interest in this logical name.

- 2) When a Propagator needs to talk to another Propagator whose logical name it knows, it sends a message to its client object to get the physical address of the other propagator.
- 3) A method in the client object :find-propagator-by-id, determines the physical address of the given propagator id.

6.2.6.1. PPN's and Operator's

In our experience with PPN's a need for a special type of a PPN arose. This special PPN is a lightweight object, whose main objective is propagation, based on input from various nodes. Our PPN's do not allow multiple inputs to a single node in a PPN. Therefore we introduce special type of PPN called an Operator. An Operator can receive inputs from multiple nodes and perform an operation on it and fan out the results.

Operator's and PPN's are Abstract PPN's. They are different in that

- a) An Operator does not compute and propagate until all its inputs are available.
- b) An Operator has multiple inputs and multiple outputs.
- c) An Operator only receives propagate messages.
- d) Unlike a PPN, an Operator, does not propagate when a new value arrives, it propagates only when all its input values arrive and belong to the same wavefront. Values from earlier wavefronts will be discarded.

User distinguishes between Operator and PPN in the AbstractPPN Table and the InterConnectionTable.

- a) The class name for an Operator will be a derived class of an Operator Class and will be given in the PPNTTable.
- b) In the InterConnectionTable, the syntax of PPN node name is - <propagator_name>/<ppn_instance_name>/<parameter> and Operator name is - <propagator_name>/<operator_instance_name>. *The Propagator distinguishes between Operator's and PPN's during instance creation by looking up the PPNTTable.*

6.2.7. PPN Redo

The propagator is designed for interactive use. Through the redo feature, users can easily experiment with alternative PPN input values and constraints. Three forms of redo are available: instant redo, selective redo, and input-determined redo.

Instant redo re-executes the PPN last executed, or Redo PPN. This is accomplished by resetting all PPNs in the propagator schedule, starting from the PPN last executed, and then reconstructing their input queues. The input queues are rebuilt by sequencing through interconnection table entries looking for input nodes ("to-nodes") that go to a

PPN, and re-propagating the values in the history list of the corresponding output nodes ("from-nodes"). The Redo() function of the propagator performs these duties.

Selective redo is a minor variation of instant redo. The only difference is that instead of using the last PPN to start the redo, the currently selected PPN is used. The RedoCallback routine (found in PropUI/callbacks.c), first checks for a selected PPN. If found, then the selected PPN becomes the Redo PPN, otherwise, the PPN last executed is used.

Input-determined redo is the last redo form. If the user changes a PPN input, by altering a value in the user interface and pressing the SetInput button, then the Redo PPN is determined by the propagator. Regardless of which PPN executed last, or which PPN is executed, the first PPN in the propagator schedule which uses the changed input will become the Redo PPN.

The propagator method GetNextRedoPPN() performs the checking of changed inputs. This function is passed the Redo PPN, either the last executed PPN or a selected PPN, and looks for PPNs appearing earlier in the schedule having changed inputs. If none of the previous PPNs have changed inputs, then the given Redo PPN is used. Otherwise, the first changed PPN becomes the Redo PPN. Once the Redo PPN is found, it is passed to the Redo() function of the propagator, which, as described earlier, resets and rebuilds the input queues of Redo PPN and its successors.

The following figure illustrates this process.

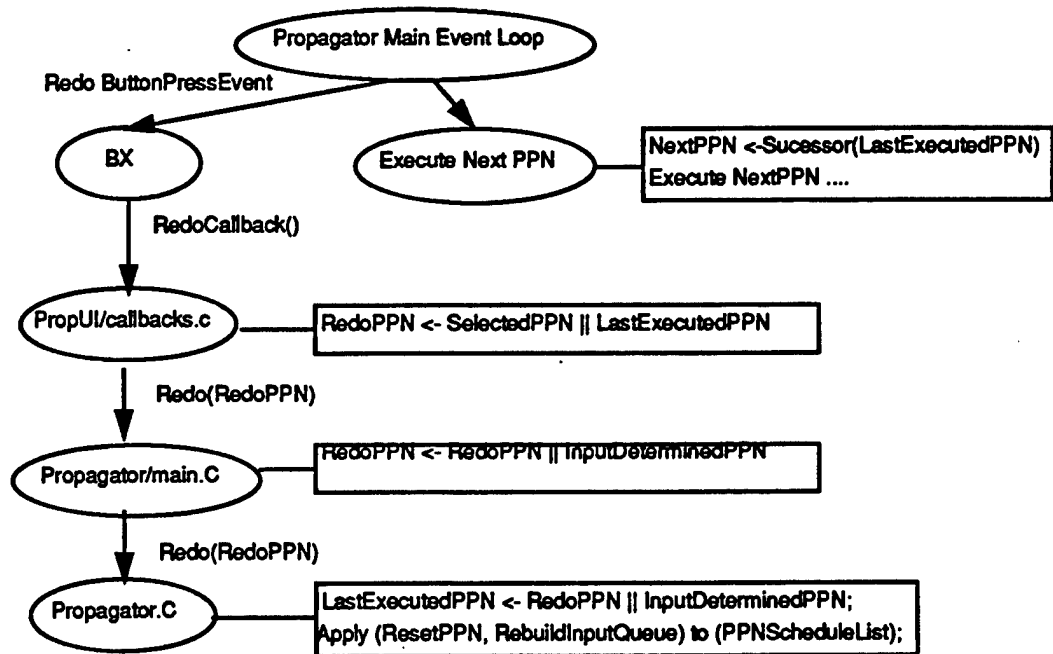


FIGURE 26. Redo execution

6.2.8 PPN Constraints

Using the propagator user interface, a constraint computation may be set by pressing the Set button under the Constraint portion of the control panel. If a PPN is not selected, you will get an error dialog indicating so. You must select a PPN which has a constraint parameter defined for it. PPNs without constraints are indicated by the symbol "NULL" in the constraint field of the PPN header or status line. PPN status lines are shown in the bottom panel of the propagator interface, along with the values of the PPN's nodes.

After the SetConstraint button is pressed, a text entry dialog box will appear, prompting you to enter the constraint expression. Constraint expressions must be a valid XLISP expression or function call. Pressing the OK button causes the constraint expression to be checked for syntax, and, if no error occurred, the constraint expression is saved in a string attribute of a PCTE node. For syntax checking to work properly be sure that the PCTE Workbench Server control variable `*socket-breakenable*` is set to `nil`. See Figure 19.

An environment variable "PROP_CONSTRAINT_DIR" should be set in the user's shell to indicate the directory (PCTE parent node) to use to store nodes containing constraint expressions. If this environment variable is not set, then the user's home object, identified by the environment variable "PCTE_HOME" is used. When constraints are saved, a node of type `hyperlisp` is created from the designated parent node. The linkname will be *constraint-parameter-name.lisp*. Thus, for the PPN `MCMDieAttach`, the link name will be `DieAttach.lisp`. The PCTE Workbench name of the node will be set to `DieAttach Constraint Computation` for easy reference. Stored constraints are loaded into their respective PPNs and nodes during propagator initialization.

Once set, a constraint can be changed by selected the same PPN and pressing the Set Constraint button. The dialog box will show the previous value, which can be edited. To delete the constraint, erase the text in the dialog box and press OK. Alternatively, you can find the node containing the constraint expression and delete it (if you do this, however, you will have to restart the propagator to remove the constraint from its internal memory).

When constraints are evaluated by the PCTE Workbench Server, the constraint value is stored in the global XLISP variable `*PPN-constraint-value*`. This permits the writing of constraint expressions such as `(< *PPN-constraint-value* 10)`. If your constraint computation does not fit on one line, define an XLISP function for the computation, and store it in the constraint parameter's node. For example, suppose you have a function `P-LT` which returns `T` if the constraint value is less than some number, `nil` otherwise. After testing the function thoroughly, use the propagator interface to enter the constraint as `(P-LT 10)`. This action will cause a node to be created as discussed above. Now, using Epoch, or any other PCTE Workbench integrated editor, copy your function into the new node. Now, when the constraint parameter expression is loaded, the contents of the node, containing your function, will also be loaded into the Server. To change the function, just edit the node.

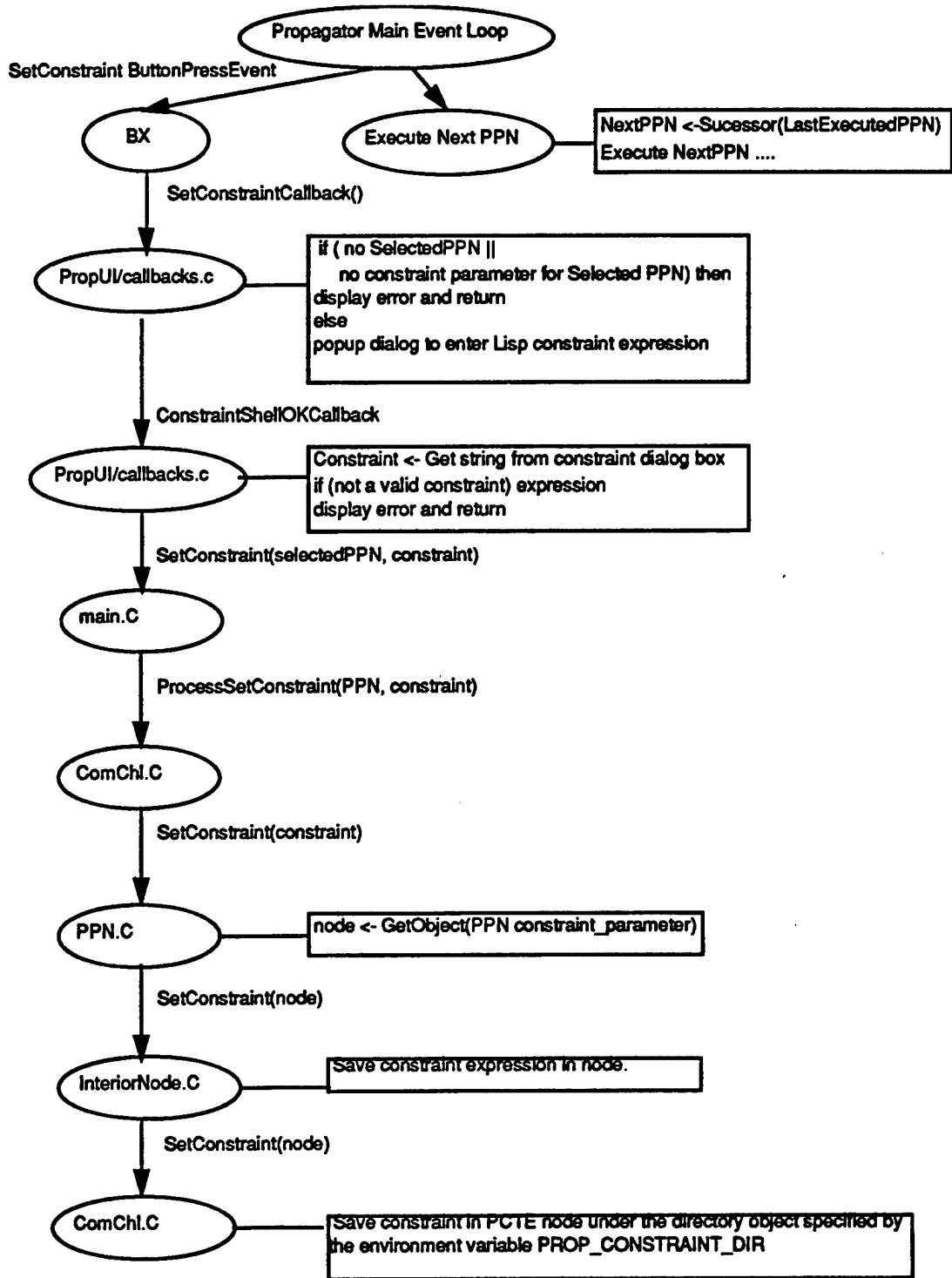


FIGURE 27. Methods of MasterTable

7. PPN Design

7.1. Introduction

This document specifies the requirements for the PPNs in the Adabra rapid prototyping environment. Section 1 presents an introduction to the PPN model. Section 2 contains definitions for some of the terms used in this document. Section 3 contains some notes about the rest of the document. Section 4 lists the requirements of the PPN. Section 5 lists the requirements of the nodes in a PPN. Section 6 lists the requirements of the operators. The Appendix at the end of the document contains a list of all requirements in this document.

7.2. Introduction to the PPN model

The PPN model is a general-purpose decision and analysis model. An example of a generic PPN model is shown in Figure 1. It consists of nodes and arcs; the nodes represent a computation process associated with a parameter and the arcs describe the relationship between parameters. The nodes and the arcs of the PPN depict the interdependencies between various parameters. Parameters whose values are known form the lowest layer of the graph as they initiate the problem solving process. Associated with each PPN is a constraint (not depicted in the figure).

In a very real sense, PPN's embody technology interdependence that is characteristic of any decision making process. PPN's are so named because they formalize the notion of object interdependence and causality relationships in terms of propagating the values of parameters to other parameters in a manner defined by a network. The edges of the network embody the relationships between parameters. The nodes of the network represent a computation of the parameter (and is often referred to as the parameter itself). The computational nodes in the PPN model are responsible for constraint checking, and can initiate backtracking. When a constraint is violated, a node for recomputation is chosen and propagation resumes from there. A chief feature of the computation model is its capacity to provide a "new" value to other parameters along the network's edges. A new value is a result of computations at the nodes of the network.

Computation at a node can be within the model, or outside. For example, the value of X in Figure 28 could be equivalent to a summation of U, V and W. Such a computation can be performed within the model by a procedural call. Complex calculations can be performed external to the PPN by a different tool. The tool can be invoked with the values of the input parameters (here U, V and W). On completion of the computation, the tool returns a value for X, or a value that aids in computing a value for X.

Parameter propagation in a PPN is entirely autonomous in that it does not require any external stimuli. It does so only when the value of the parameter changes. A change in the value of the parameter autonomously propagates the change to other parameters. The heart

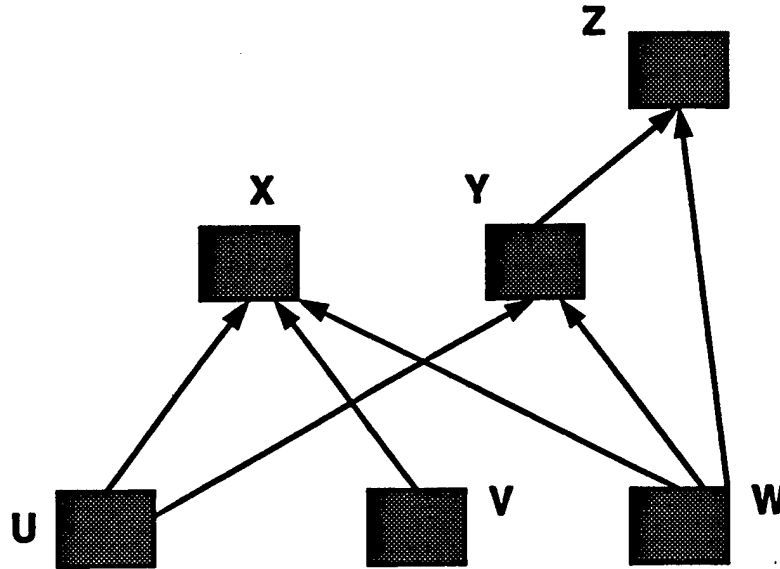


FIGURE 28. A PPN depicting interdependence between parameters

of the PPN implementation consists of a Propagator engine. Parameter propagation in a PPN can be controlled by the user with the help of break-points in the execution. This feature is useful for the partial execution or propagation if desired by the user.

While PPN's parameter propagation is an excellent way to implement parameter dependency through autonomous updates, it is not sufficient for computing a solution. The PPN's powerful backtrack feature does the following: a new value is selected in one of the previous nodes and the value is propagated as before. The new value of the parameter results in a re-computation of subsequent nodes and so on. In effect, the PPN searches through the problem space for a solution. We can visualize this computation as a conic propagation wave front alternately surging and retreating through the PPN's nodes, and rejecting all stable values that do not satisfy the constraint. Finally, when the PPN does stop computation, a solution is found.

Thus, the hallmark of the model is autonomous propagation of parameter values and recomputation of values when constraints are violated.

7.3. Definitions

Wavefronts- A wavefront is defined to be a set of values for all parameters arising from the progression of computation through the network of PPNs for a specified set of input values.

Wavefront Numbers- Each wavefront has a unique number. The syntax of a wavefront number is:

<wavefront number> ::= <field>| <field>\$<wavefront number>

<field> ::= # | <parameter index>.<p-index>

<parameter index> ::= <number>

<p-index> ::= <number>

Field- Each field of the wavefront number is delimited by a '\$' symbol.

Input Wavefronts- An input wavefront for a PPN with N input nodes is defined as a non-empty set of at most N plists in the input queue, such that each plist is for a different input node.

pvalue- A pvalue is a pair consisting of a wavefront number and a value.

plist- A plist is a linked list of pvalues. All pvalues in a plist have an unique wavefront number. All pvalues in a plist have an unique value.

p-index- The position of the pvalue in the plist. The positions are numbered starting from one.

Input Node- A node in a PPN that receives values from operators or nodes in other PPNs, or the user, and does not have another node in the same PPN on its fan-in.

Output Node- Any node in a PPN which has a fan-out to a node or operator not in the same PPN.

Interior Node- Any node in a PPN that is neither an input node nor an output node.

Level Graph- A level graph of a PPN (or network of PPNs) is a directed acyclic graph with an ordered set of levels. A level is a set of nodes (or PPNs) that are independent. Two nodes are independent if there exists no path between the nodes in the graph. Nodes in level i have ancestor nodes in levels j (j < i) and descendants in levels k (k > i). Nodes at level 1 do not have any ancestors.

Active and Inactive Constraints- Constraints in a PPN can be in one of two states active or inactive. Active constraints are tested in PPN execution and may cause backtracking of the PPN execution. Inactive constraints are ignored.

7.4. Notes about the Requirements

The requirements specified in this document completely specify the expected behavior of the modules/objects of interest. Any requirement not included is excluded. **Bold** and *Italic* fonts have been used to further emphasize certain aspects of the requirements.

All requirements, over all sections, have been numbered sequentially. Requirements have been grouped into sub-sections numbered using roman numerals (I, II,...). Each requirement also has a short descriptive title. Each requirement has a complete description following the title.

Where appropriate Requirement Notes have been used to specify the requirements imposed on other modules/objects in the Adabra rapid prototyping environment by the assumptions, made in this document, about their behavior.

The following is the list of requirements that must be satisfied for the PPN.

I Input Messages to PPNs

R:1. PPN Input Messages-

Each PPN class must provide methods to handle the following messages:

1. **Create**
Create an instance of the PPN class and all its nodes and initialize all instance data structures.
2. **Delete**
Delete the specified instance.
3. **GetValueList (*parameter*)**
Return a plist for the *parameter*.
4. **GetFanOutList (*parameter*)**
Return a list of the fan-out of the *parameter*.
5. **GetFanInList (*parameter*)**
Return a list of the fan-in of the *parameter*.
6. **GetInterconnectionTable**
Return the interconnection table for this PPN.
7. **GetPropagatedWavefront**
Return a list containing the pvalues of the input nodes from the current propagated (internal) wavefront.
8. **GetExecutionState**
Return all the information pertaining to the current execution state of the PPN instance.
9. **GetConstraint**
Return the constraint if any associated with this PPN.
10. **GetBreakPoints**

Return a list containing all breakpoints, including wavefront and backtrack breakpoints, for this PPN instance.

11. **GetBacktrackNodes**

Return a list containing the backtrack nodes for this PPN instance.

12. **Propagate (*item*)**

Add *item* to the tail of the input queue.

13. **SetConstraint (*value*)**

Set the constraint constant to *value* specified, and activate constraint if deactivated. If no constraint is defined or if *value* is not of the same type (or subtype) of the constraint parameter then do not perform the operation and return an error message, else return an acknowledgment.

14. **ActivateConstraint**

Activate the constraint.

15. **DeactivateConstraint**

Deactivate constraint. Note the constraint is never actually deleted, just deactivated.

16. **SetValueList (*parameter, plist value list*)**

Set the value of *parameter* to be *value list*. *parameter* may not be an input parameter. The length of *value list* must be greater than zero. The Node computation is not performed until an UnSetValueList message is received. The value list can not be over written by PPN (Node) computation.

Requirement Note:

SetValueList may not be used when the PPN is suspended at a breakpoint.

17. **SetFanOutList (*source parameter, target parameter*)**

Set the fan-out of a output node. The *source parameter* is a node in the PPN instance, *target parameter* is a input node in another PPN instance.

18. **SetBreakPoint (*parameter*)**

Set a breakpoint for forward execution of the PPN instance at *parameter*. If *parameter* specifies an input node then return an error message, else return an acknowledgment. If a breakpoint has already been set at *parameter* then return an acknowledgment.

19. **SetBacktrackBreakPoint**

Set a breakpoint for backtracking. Execution is suspended whenever the PPN instance backtracks.

20. **SetWavefrontBreakPoint**

Set a breakpoint before the next (internal) wavefront is propagated. Execution is suspended before the next legal cross-product element is propagated.

21. **UnSetValueList (*parameter*)**
Allow PPN execution to over write the value for *parameter*. This does not delete the current value list of the parameter.
22. **DeleteBreakPoint (optional *parameter*)**
Delete the breakpoint at *parameter*. If no *parameter* is specified then delete all breakpoints in the PPN instance.
23. **DeleteBacktrackBreakPoint**
Delete the backtrack breakpoint.
24. **DeleteWavefrontBreakPoint**
Delete the wavefront breakpoint.
25. **Continue**
Continue execution of the PPN instance from where it was suspended.
26. **Execute**
Begin execution.
27. **Stop**
Stop executing this PPN instance after completing the current internal wavefront and export any values at the output nodes. Also reset all data structures that would reset upon normal completion of execution.

II Output Messages from PPNs

R.2. PPN Output Messages-

PPN instances must generate the following messages:

1. **Export**
Inform the propagator that there are values to be exported, by calling the export function of the propagator module for each output node that has values to be propagated. This message should be generated when the PPN instance has completed execution. Multiple nodes in a PPN instance may serve as output nodes.
2. **Tool Messages**
These are messages generated by the nodes in a PPN instance for communication with other tools. The PPN will just forward these messages to the communication module of the propagator.
3. **Error Messages**

These error messages should not be generated if the PPN generates an export message. The PPN may generate the following error messages:

i. Backtracking failed

This message is generated when the PPN backtracks to the input level and all legal combinations caused the active constraint to be violated. If any legal combination did not cause constraint violation then an export message should be generated instead.

ii. Type check failed for input *parameter*.

This message is generated when any input to any parameter to PPN is of the wrong type. The error message must specify the parameter and the offending input.

iii. Trigger not set.

This message is generated when the trigger to a PPN is not set.

iv. Incomplete input wavefront.

This message is generated if no complete legal combination of input values can be generated from the current set of input plists.

III Data Inputs to PPNs

R.3. PPN Data Inputs-

Input nodes in a PPN, will receive data input from the user and other PPNs.

R.4. PPN Input Queue-

PPNs must have an input queue where all the data inputs to the PPN should be placed.

R.5. Input Nodes-

All input nodes should process plists, by computing legal combinations over all inputs and then propagating each combination separately.

R.6. User specified values for the Input Nodes of a PPN-

The user may choose to specify a value for any input node of a PPN.

Requirement Note: Builder Requirement

It is desirable that the User Interface Operators be generated automatically by the builder module.

R.7. Type Checking of Inputs-

All type checking must be performed on each member of the plist. If any member of the plist is of the wrong type the entire plist should be rejected and an error reported.

IV Data Outputs from PPNs

R.8. Outputs from a PPN-

Each output node of a PPN must form a plist of its values and then export the plist.

R.9. Output Nodes in a PPN-

Any node, except input nodes may serve as an output node. If the PPN has a constraint specified (active or inactive) then the constraint node and all nodes that do NOT have a common ancestor with the constraint node in the PPN graph, may serve as output nodes.

R:10. Exporting Multiple Values from Output Nodes-

Only output nodes at the top most level of the of the PPN level graph (nodes without any descendants in the PPN) may export multiple values. All other output nodes must export the last propagated value.

V Breakpoints in a PPN

R:11. Breakpoints in PPN execution-

Users can specify breakpoints in PPN instance execution. Breakpoints are set at nodes in a PPN and cause execution to be suspended when execution reaches that node, i.e. when the node is scheduled for execution. The suspension takes effect before the node computation is performed. When a breakpoint is reached the execution state of the PPN must be updated.

R:12. Deletion of Breakpoints-

Breakpoints are in effect until they are deleted.

R:13. Breakpoints and Backtracking-

Users may set a breakpoint for backtracking by using the message SetBacktrackBreakPoint. If a backtrack breakpoint is set then PPN execution should be suspended as soon as a PPN backtracks. This breakpoint is in effect until deleted by sending a DeleteBacktrackBreakPoint message. PPN execution state must be updated before the PPN suspends at this breakpoint.

R:14. Breakpoints and Wavefronts

Users may set a breakpoint between wavefronts by using the message SetWavefrontBreakPoint. When a PPN completes execution of one internal wavefront, PPN execution should be suspended before the next internal wavefront is propagated, but after the next legal input combination is computed. This breakpoint is in effect until deleted by sending a DeleteWavefrontBreakPoint message. PPN execution state must be updated before the PPN suspends at this breakpoint.

R:15. Resuming after a Breakpoint-

Users can resume execution after a breakpoint by sending a Continue message. PPN execution state must be updated upon receiving a Continue message.

R:16. No Breakpoints at Input Nodes-

Breakpoints may not be set at input nodes.

VI Backtracking in PPNs

R:17. Backtracking-

A PPN must backtrack if an active constraint is violated. Backtracking involves restarting the PPN execution from the next backtrack node in the list of backtrack nodes. This node may now propagate a new value from its value list. If the value list is empty then backtracking proceeds to an ancestor of this node and this node must reset its history list.

R:18. Backtrack Nodes-

The list of backtrack nodes should be specified when a PPN is defined. This list can not be changed after PPNs are compiled into the propagator. All input nodes are backtrack nodes. There may be at most one backtrack node at each level in the level graph for a PPN, with the exception of the input nodes.

R:19. Backtracking History-

If backtracking reaches a node which generates multiple values then the previously propagated value must be deleted from the value list of the node and added to the history list of the node.

R:20. Termination of Backtracking-

Backtracking should be terminated when backtracking reaches the input level and all legal input combinations for the current input wavefront have already been propagated, or when the PPN gets a stop message.

VII PPN Execution

R:21. Triggers-

Each PPN instance should have a special input parameter called the trigger. The trigger does not have a type. The trigger should be reset at the end of each input wavefront. Any output node (irrespective of type) of any PPN instance maybe connected to the trigger input of a PPN instance.

R:22. PPN Execution Condition-

Each PPN upon being scheduled for execution should test if all conditions for execution are satisfied before beginning execution. The conditions to be tested are:

- i. Is the trigger set for the current input wavefront, and
- ii. Do all input nodes have legal values, and
- iii. Is the input queue non-empty.

R:23. Completion of Execution-

A PPN instance should complete execution on an input wavefront if:

- i. backtracking is terminated, or
- ii. all parameter propagation has been completed for the current input wavefront, or
- iii. the PPN instance receives a Stop message

Upon completion of execution delete the current input wavefront and reset appropriate internal data structures.

VIII Wavefronts in PPN Execution

R:24. Wavefronts and Multiple Values for a Parameter-

If the parameter is an output node then the base wavefront number should be computed for the multiple values as a concatenation of the wavefront numbers of the current cross-product combination of input values. The computed base wavefront number should not have identical fields. For each of the values generated by the output node assign a wavefront number as follows:

$\langle \text{base wavefront number} \rangle \$ \langle \text{output parameter index} \rangle . \langle \text{p-index} \rangle$

Where $\langle \text{output parameter index} \rangle$ is the unique number assigned to this output node in the network and $\langle \text{p-index} \rangle$ is the location of the value in the plist counting from one.

For instance if the current legal combination has the wavefront numbers {1.1, 1.1\$2.1, 3.1} then the base wavefront number would be 1.1\$2.1\$3.1 and if the output plist had three values then their wavefront numbers would be 1.1\$2.1\$3.1\$29.1, 1.1\$2.1\$3.1\$29.2 and 1.1\$2.1\$3.1\$29.3, where 29 is the parameter index for this output parameter.

R:25. Legal cross-product combinations of Wavefronts-

An input pvalue may be legally combined with another input pvalue if:

- i. both pvalues have the same wavefront number, for example 1.2\$3.5 and 1.2\$3.5 or
- ii. if the wavefront numbers are different and there are no common parameter specifications, for example 1.2\$3.5 and 23.2\$34.5, or
- iii. if the wavefront numbers are different and some parameter specifications are common, and the p-indices are identical, for example 1.2\$3.5 and 1.2\$33.1.

IX Scheduling of PPN Nodes

R:26. Node Scheduling-

Within the PPN, nodes must be executed as per a schedule, created by a specified scheduling algorithm. The scheduling must be non-preemptive. The scheduling algorithm may be different from the one used by the propagator module.

X PPN Definition

R:27. Node Specification-

Nodes in a PPN will be specified using a table called the node table.

R:28. Node fan-in and fan-out-

A node instance, except input nodes, in a PPN may have multiple fan-in and fan-out. Input nodes must have exactly one fan-in and multiple fan-out.

R:29. Interconnection Specification-

Interconnections between nodes in a PPN will be specified using an interconnection table. All interconnections in a PPN are unconditional.

XI PPN Constraints

R:30. One Constraint per PPN-

Each PPN can have at most one constraint.

R:31. Constraint Syntax-

A constraint is of the form:

$\langle \text{constraint} \rangle ::= \langle \text{func} \rangle (\langle \text{param} \rangle) \langle \text{relational operator} \rangle \langle \text{constant} \rangle$

$\langle \text{relational operator} \rangle ::= < | > | = | < = | > = | != | \text{subset} | \text{superset}$

$\langle \text{func} \rangle ::=$ a well defined function of the parameter

The constraint is associated with one node in the PPN. $\langle \text{func}(\text{param}) \rangle$ is a function of the parameter at the node.

R:32. Constraint Constant Type-

$\langle \text{constant} \rangle$ must be of the same type as the parameter. $\langle \text{constant} \rangle$ could be a range if the type supports ranges. $\langle \text{constant} \rangle$ can also be a discrete set, in which case each element of the set must be of the same type as the parameter.

7.5. PPN Nodes

The following is the list of requirements that must be satisfied by the nodes in a PPN.

I Node Computation

R:33. Types of Node Computation-

There are three types of node computation. They are:

1. Algebraic Computation

Algebraic computation represents evaluation of simple algebraic expressions. An algebraic expression can be any legal 'C' expression. Expressions may also use function calls from a set of pre-defined math functions.

2. Procedural Computation

Procedural computation is a call to a well-defined procedure written in 'C'. It is required that procedures can be defined as a part of the node specification. It is desirable that procedures could be specified in procedural languages other than 'C'. It is desirable that procedures need not be defined as a part of the node definition, but are available as a separate archive.

3. Constructive Computation

Constructive computations involve external tools, database queries and any other tool interaction. Nodes must communicate with external tools only via the communication module of the propagator. Nodes may not depend on the details of the communication apparatus or the location of the tool. Nodes must address tools by their logical names. Nodes must not have hard-coded paths to the location of the tools.

Nodes must be capable of supporting all three kinds of computation specified above.

R:34. Node Values are Persistent

All nodes including input nodes should maintain their values until they are over written by the next node computation, with the exception specified by R39.

R:35. Multiple Values generated by Node Computation-

Node computation may generate multiple for any set of input pvalues.

R:36. Best Value first Propagation-

The PPN designer may choose to provide a method for each node, except input nodes, that can order the pvalues of the parameter for that node in a increasing order of “goodness”. The node will then propagate the best (first) value from the list and then (if necessary) the next best pvalue and so on. If the node is an output node then the pvalues should be placed in a plist in a increasing order of “goodness” and then exported. Wavefront numbers should be assigned to the pvalues after they are ordered.

R:37. Default value for Input Nodes-

The PPN designer may choose to provide input nodes of a PPN class with a default plist. If no default value is specified and the input node does not have a value then the PPN should not execute and signal an error.

R:38. User Specified values for Nodes-

If the value of a node is set by a SetValueList message then it cannot be over written by node computation until it is unset by a UnSetValueList message.

7.5.1. Operators

The following is the list of requirements to be satisfied by operators.

I Input Messages to Operators

R:39. Operator Input Messages-

Each Operator class must provide methods to handle the following messages:

1. **Create (integer *num_fan_in*)**
 Create an instance of the class with number of inputs set to *num_fan_in*.
 num_fan_in > 0.
2. **Delete**
 Delete the specified instance.
3. **GetValueList**
 Return the value list of the operator instance.
4. **SetFanOutList (target parameter)**
 Add *target parameter* to the fan-out list of the operator instance.
5. **Execute**

Begin execution.

6. Propagate (*source parameter, value*)
Place *value* at the tail of the queue for *parameter*.

II Output Messages from Operators

R:40. Operator Output Messages-

1. Export

Operator instances should generate export messages. This message is used to export the values to the destination nodes. The same plist should be exported on all the fan-out.

2. Error Messages

- i. Type Checking Failed

This message is returned when type checking failed on an input plist. The offending plist is deleted from the input queue.

- ii. Incomplete Inputs

This message is returned when an operator is scheduled for execution, and requires all inputs but a legal input combination cannot be constructed.

III Data Input to Operators

R:41. Operator Data Inputs-

An operator may receive input from the user, PPNs, or other Operators.

R:42. Operator Input Queues-

Each operator input must be modeled as a separate queue.

R:43. Operator Input Queue Processing-

When a set of inputs are delivered to the computation procedure they should be deleted from the input queue. Inputs to the computation procedure must be at the head of the input queues.

R:44. Operator Input Processing-

The operator designer may choose to configure an operator class as:

- i. requiring all inputs before computation can be started or
- ii. requiring at least one input before computation can be started.

R:45. Inputs to Operators-

Each input to an operator should be a plist. Operators should deliver legal combinations in the cross-product of the input plists to the computation procedure.

R:46. Type Checking of Operator Inputs-

All operator inputs must be of the same specified type. Each element of the input plist must of the same specified type. Any input which is not of the specified type should be rejected and an error signalled.

R:47. Operator Inputs are NOT Persistent

Operators should not store their last input values after operator computation for those values is completed.

IV Data Outputs from Operators

R:48. Outputs from Operators-

The operator instance must accumulate the outputs from all items in the cross-product and place it in a plist, and then export the plist.

R:49. Operator Outputs are Persistent

Operators should maintain the latest output plist.

V Breakpoints in Operators

R:50. No Breakpoints in Operators-

Users may not specify breakpoints for operator instance execution.

VI Operator Definition

R:51. Operator Computation Definition-

Each operator class encapsulates a well-defined operation. The operation must be specified as a well-defined 'C' procedure accepting a variable number of inputs of a particular data type and producing an output of the same data type.

R:52. Operator Fan-out-

Each operator may have multiple fan-out.

R:53. Type of Operator Inputs and Outputs-

All operator inputs and outputs are of the same type and are specified by the operator designer. These types cannot be changed by the user at run time.

VII Operator Execution

R:54. Operator Execution Condition-

Each operator upon being scheduled for execution should execute if a legal combination can be produced of the input plists. If the operator requires all inputs for execution then a legal combination must contain a value for all inputs, else if the operator requires at least one input then a legal combination must have at least one input.

VIII Wavefronts in Operator Execution

R:55. Legal cross-product combinations of Wavefronts-

See similar requirement for PPNs.

IX User Input Operators

R:56. Subtype of Operators

User Input Operators are a subtype of the class of operators.

R:57. User Input to PPNs via User Input Operators

User input to input nodes in a PPN may be propagated via User Input Operators.

R:58. User Input Operator Computation

User Input Operators are used to compute the wavefront number for user inputs to PPN input nodes. User Input Operator computation is as follows:

for each pvalue in the input plist

```
{  
  count <- count + 1;  
  pvalue.wavefront-number <- <parameter index>.<count>  
}
```

R:59. Input to User Input Operators

Input to User Input Operators provided by the user.

R:60. Fan-out of User Input Operators

User Input Operators may have multiple fan-out. All fan-out must be connected to input nodes of PPN instances.