



**Calhoun: The NPS Institutional Archive**  
**DSpace Repository**

---

Faculty and Researchers

Faculty and Researchers' Publications

---

1989

## A Design Database for Rapid Prototyping

Douglas, B.; Luqi

Naval Postgraduate School

---

B. Douglas and Luqi, "A Design Database for Rapid Prototyping", Technical Report NPS 52-89-022, Computer Science Department, Naval Postgraduate School, 1989.  
<https://hdl.handle.net/10945/65228>

---

*Downloaded from NPS Archive: Calhoun*



<http://www.nps.edu/library>

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

**Dudley Knox Library / Naval Postgraduate School**  
**411 Dyer Road / 1 University Circle**  
**Monterey, California USA 93943**

T39

NPS52-89-022

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



A DESIGN DATABASE FOR RAPID PROTOTYPING

BRYANT S. DOUGLAS  
LUQI

APRIL 1989

Approved for public release; distribution is unlimited.

Prepared for:

Naval Postgraduate School  
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL  
Monterey, California

Rear Admiral R. C. Austin  
Superintendent

H. Shull  
Provost

The work reported herein was supported by the National Science Foundation, the Office of Naval Research and the Naval Postgraduate School Research Council.

Reproduction of all or part of this report is authorized.

This report was prepared by:



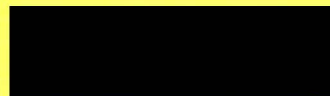
LUQI  
Assistant Professor  
of Computer Science

Reviewed by:

Released by:



ROBERT B. MCGHEE  
Chairman  
Department of Computer Science



KNEALE T. MARSHALL  
Dean of Information  
and Policy Science

# REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)  <b>NPS52-89-022</b>			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION  <b>Naval Postgraduate School</b>		6b. OFFICE SYMBOL (If applicable)  <b>52</b>	7a. NAME OF MONITORING ORGANIZATION <b>National Science Foundation &amp; ONR Sponsored Navy Direct Funding</b>		
6c. ADDRESS (City, State, and ZIP Code)  <b>Monterey, CA 93943</b>			7b. ADDRESS (City, State, and ZIP Code)  <b>Washington, D. C. 20550</b>		
8a. NAME OF FUNDING / SPONSORING ORGANIZATION  <b>Naval Postgraduate School</b>		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER  <b>NSF CCR-8710737 O&amp;MN, Direct Funding</b>		
8c. ADDRESS (City, State, and ZIP Code)  <b>Monterey, CA 93943</b>			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification)  <b>A DESIGN FOR RAPID PROTOTYPING (U)</b>					
12. PERSONAL AUTHOR(S) <b>DOUGLAS, Bryant S., LUQI</b>					
13a. TYPE OF REPORT <b>Progress</b>		13b. TIME COVERED FROM <b>Sept 88</b> TO <b>Mar 89</b>		14. DATE OF REPORT (Year, Month, Day) <b>1989 March</b>	
15. PAGE COUNT <b>30</b>					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) <b>Design database, project database, object-oriented design computer-aided prototyping, object-oriented database management system, specification, rapid prototyping</b>		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) <b>This paper presents a conceptual design and experimentation of a design database (DDB) for rapid prototyping. The experiments were conducted using an object-oriented database management system. Formal requirements and specifications are provided for the objects and interfaces of the design database. The design database contains the Prototype System Description Language (PSDL) prototype descriptions for each software development project using the Computer-Aided Prototyping System (CAPS). The DDB is responsible for managing and retrieving the versions, refinements, and alternatives of each prototype.</b>					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>		
22a. NAME OF RESPONSIBLE INDIVIDUAL <b>LUQI</b>			22b. TELEPHONE (Include Area Code) <b>408-646-2735</b>		22c. OFFICE SYMBOL <b>52Ld</b>





## 1. INTRODUCTION

The development of hard real-time and embedded software systems is an extremely complex and expensive process. A software development methodology which will reduce the development costs, increase the productivity rates, and reduce the maintenance costs of these systems is long overdue. The prevailing ideas of today are computer-aided rapid prototyping, software reusability, and the use of an executable high-level specification language. The goal of the Computer-Aided Prototyping System (CAPS) is to integrate all of these ideas and more, into one software development tool. [Ref. 1]

### 1.1. Hard Real-Time and Embedded Software Systems

The development of hard real-time and embedded software systems creates additional problems in the software development process. They all generally share a set of common characteristics:

- Large. Thousands/millions of lines of code.
- Long-lived. 10 to 15 years.
- Continuous Change. Due to changing requirements.
- Physical constraints. In target hardware address space/speed.
- High reliability. Also fault-tolerant. [Ref. 2]

Each of these characteristics makes embedded systems difficult to develop. The effects of using Ada on embedded software development costs may not be felt immediately, but the long term savings of a prototyping language for embedded systems will be realized [Ref. 2,3,9]

### 1.2. The Computer-Aided Prototyping System

The Computer-Aided Prototyping System is one attempt to improve the productivity and reliability of software through the use of computer-aided rapid prototyping via specification and reusable components [Ref. 1]. This approach to rapid prototyping uses a specification language called Prototype System Development Language (PSDL) integrated with a set of software tools [Ref. 1]. The major components of CAPS [Ref. 1,22] are a user interface [Ref. 11] consisting of a syntax directed editor [Ref. 12] and graphical editor [Ref. 10], a design management system consisting of a software base management system [Ref. 13] and design database [Ref. 14], and an execution support system consisting of a translator [Ref. 15,16], static and dynamic scheduler [Ref. 17,18,19,20], and debugger [Ref. 21].

### 1.3. The Design Database

Vast amounts of evolving data are created in the design of hard real-time software systems. Conventional database management systems (DBMS) were designed for business applications and as such are insufficient to handle the needs of computer-aided design (CAD) applications. The data must be managed so that it can be stored and retrieved according to the needs of design engineers. In CAPS, the Design Database (DDB) must manage the storage and retrieval of the Prototype System

Description Language (PSDL) program. The DDB must be a specialized DBMS which will store PSDL specifications in a hierarchical format.

## 2. CONCEPTUAL DESIGN FOR THE DESIGN DATABASE

The purpose of the Design Database (DDB) in the Computer-Aided Prototyping System (CAPS) is to manage the project database so that it can be stored and retrieved according to the needs of design engineers. The requirements analysis was conducted using the specification language SPEC [Ref. 4]. SPEC is a language for giving black-box specifications in the early stages of software design [Ref. 4]. The goal of requirements analysis is to establish the purpose of the proposed software system and to establish constraints and boundary conditions on the rest of the software development process [Ref. 5].

The result of the requirements analysis should include the following:

- A model of the system's environment.
- A description of the goals of the system and the functions it must perform.
- Performance constraints on the system.
- Constraints on the implementation of the system.
- Resource constraints for the development project.
- A specification of the external interfaces of the system. [Ref. 5]

### 2.1. Environment Model

The environment model must supply the concepts needed for describing the world in which the proposed system will operate. These concepts consist of the types of objects in that world, the attributes of those objects, the relations between those objects, and the laws governing those objects and relations. [Ref. 5]

The environment model for the Design Database is shown below. The model was formulated in terms of reusable model components. A reusable component is a definition of a general type or relation [Ref. 5]. The model is expressed in a simple notation that is explained as it appears. Explanatory comments are preceded by a "-" symbol.

type CAPS

  a\_kind\_of(software\_system,CAPS)

  -- The Computer-Aided Prototyping System (CAPS) is a software\_system.

type psdl

  a\_kind\_of(psdl,language)

  -- PSDL is a language for expressing specifications.

  created\_by(every psdl,a user\_interface)

  -- All specifications are created in the user interface.

type design\_database

  a\_kind\_of(design\_database,software\_system)

  -- The design\_database is going to be a software\_system.

```

part_of(a design_database, every CAPS)
-- The design_database is part of CAPS
unique(design_database)
-- There will be only one instance of the design_database for each project.
proposed(a design_database)
-- The system to be built is a design_database.
controls(a design_database, node)
-- The design_database controls the design data by
-- managing collections of data called nodes.

```

```

type design_engineer
a_kind_of(design_engineer, user)
-- The design engineer is a user of the system.
uses(every design_engineer, a user_interface)
-- The model includes only the design engineers that will interact
-- with the design_database via the user_interface.

```

```

type user_interface
a_kind_of(user_interface, software_system)
-- The user_interface is a software_system.
part_of(a user_interface, every CAPS)
-- The user_interface is a part of CAPS.
created_by(a user_interface, every node)
-- The user_interface is the only source for data.

```

```

type data
-- any kind of data that is used by a software system
uses(a software_system, every data)

```

```

relation reads(software_system, data)
-- true if the data is an input for the software system
reads(any software_system, any data) => uses (software_system, data)

```

```

relation writes(software_system, data)
-- true if the data is an output for the software system
writes(any software_system, any data) => uses (software_system, data)

```

```

relation updates (software_system, data)
-- true if the data is both an input and an output for the system
updates(any software_system, any data)
<=> reads(software_system, data) & writes(software_system, data)

```

```

type node
a_kind_of(node, data)
-- A node is the basic structure for maintaining the design database.
needed_for(node, every specification)
-- Every specification created in the system will be stored in a node.

```



```

created_by(every node,a user_interface)
-- All nodes are created via the user interface.
-- The attributes of a node are listed below.
specification(node):psdl
implementation(node):psdl
control_constraints(node):psdl
graphic_record(node):graphic_record
-- A node consists of a specification, a graphic record,
-- an implementation, and control constraints.
text(node):psdl_file

```

```

type graphic_record
a_kind_of(graphic_record,data)
part_of(node,graphic_record)
-- A graphic record is one input into in a node.
created_by(a user_interface,every graphic_record)

```

```

type implementation
a_kind_of(implementation,data)
part_of(node,implementation)
-- An implementation is one input into a node.
created_by(a user_interface,every implementation)

```

```

type control_constraints
a_kind_of(control_constraints,data)
part_of(node,control_constraints)
-- Control constraints are part of the data in a node.
created_by(a user_interface,every control_constraints)

```

```

type psdl_file
a_kind_of(psdl_file, data)
-- A file containing the PSDL program will be the ultimate output.

```

## 2.2. High Level Goals and Constraints

The requirements for the DDB are formalized by writing a description of the goals of the system and the functions it must perform in terms of the model. A major part of the requirements analysis is turning the informal problem statement into a precise, testable, and feasible set of requirements. The high level goals for the Design Database are shown below.

**G1:** The purpose of the system is to store the levels of a PSDL design in a hierarchical format.

**G1.1:** The system must allow design engineers to retrieve the levels for review or editing.

**G1.2:** The system must be able to create and insert new levels into the structure.

**G1.2.1:** The system must interface with the user interface for inputs to the database.

**G2:** The system must be able to generate the entire PSDL program.

There are three types of constraints for a software system: implementation, performance, and resource. The constraints for the Design Database are given below.

**Implementation constraints:**

**C1:** The design database must be implemented with a DBMS which is compatible with the Sun workstation and Unix operating system.

**Performance constraints:**

**C2:** The responses of the design database must be fast enough not to irritate the design engineers using the system.

**Resource constraints:**

**C3:** The Design Database will be developed by thesis students at the Naval Postgraduate School.

### 2.3. Conceptual Model of the Design Database

The DDB is a hierarchical storage structure for the PSDL program. This structure is a tree consisting of atomic and composite nodes. Figure 1 illustrates this structure. Both types of nodes contain a node name, a PSDL specification, an implementation, and parent <-> child relationship information. An atomic node's implementation is ADA code. A composite node's implementation is graph. A graphic implementation may contain timing constraints in the form of control constraints. Each level of the tree is created by the decomposition of the parent node. The decomposition process is complete when all leaf nodes are atomic. Figures 2 and 3 present graphical representations of atomic and composite nodes.

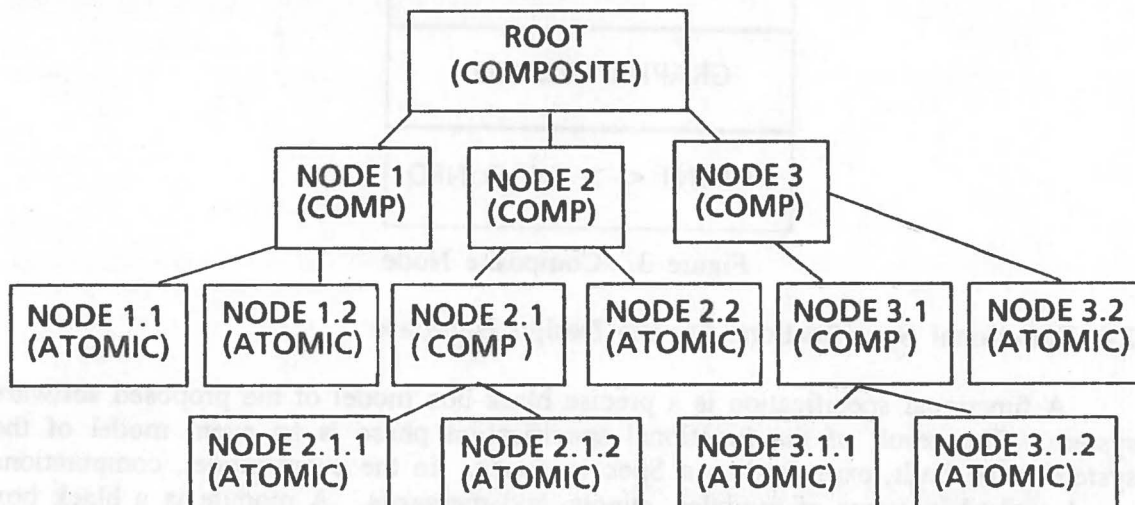


Figure 1. Conceptual DDB Structure

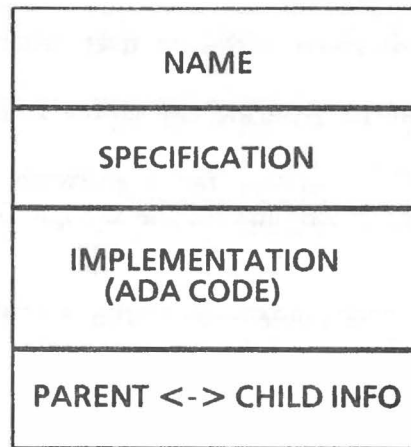


Figure 2. Atomic Node

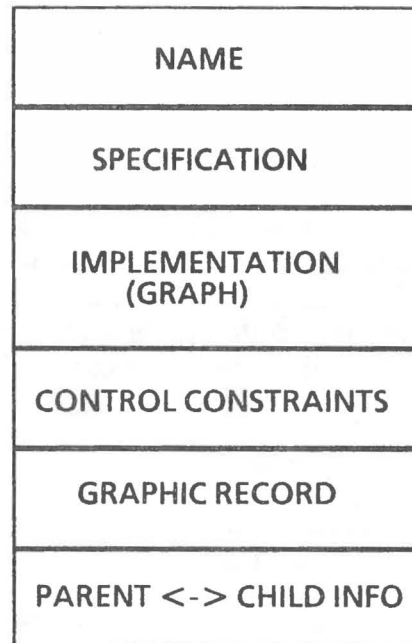


Figure 3. Composite Node

#### 2.4. Functional Specifications for the Design Database

A functional specification is a precise black-box model of the proposed software system. The result of the functional specification phase is an event model of the system to be built, expressed in a Spec language. In the event model, computations are described in terms of modules, events, and messages. A module is a black box that interacts with other modules only by sending and receiving messages. An event

occurs when a message is received by a module at a particular instant of time. A message is a data packet that is sent from one module to another. [Ref. 4]

The Spec language provides a means for specifying the behavior of three different types of modules: functions, state machines, and abstract data types. Function modules are immutable, and calculate functions on data types. A machine is a module with an internal state. An abstract data type consists of a set of instances and a set of primitive operations involving instances. [Ref. 4]

The functional specifications begin with a skeleton of a specification with places for each of the missing details to be filled in later. The initial specifications are shown below.

#### **MACHINE design\_database**

-- The design database is a machine because it is time sensitive.

INHERIT user\_interface\_interface

-- The system will interface with the user interface portion of the

-- CAPS system therefore, inheritance relations exist.

STATE

INVARIANT true

INITIALLY true

END

#### **MACHINE user\_interface**

-- The user\_interface is an external system that sends messages to

-- the design database.

STATE

INVARIANT true

INITIALLY true

END

The next step is to make a list of the messages in each interface by consulting the requirements. The user interface is the only interface for the design database. The following messages are produced corresponding to the goals of the system:

#### **user\_interface\_interface**

create\_root\_node       G1, G1.2

create\_child\_node       G1, G1.2

delete\_node            G1, G1.2

lookup\_node            G1.1

get\_parent             G1.1

get\_child              G1.1

traverse\_tree          G2

These messages have covered all of the goals in the goal tree with the exception of G1.2.1. This goal is an assumption about the environment. The result of the user\_interface interface messages are shown below.



```

MACHINE user_interface_interface
  -- The specification begins by identifying the interface messages.
  STATE (tree:map{node,set{node}})
  INVARIANT existing_nodes(node)
  INITIALLY node = {}

  MESSAGE lookup(node_name) -- G1.1
  -- Find the node requested.
  WHEN ? -- node found
    REPLY node -- return node contents
  OTHERWISE REPLY EXCEPTION node does not exist

  MESSAGE get_parent(node) -- G1.1
  -- Find the parent of the node requested.
  WHEN ? -- parent found
    REPLY node -- return parent node
  OTHERWISE REPLY EXCEPTION node is the root

  MESSAGE get_child(node) -- G1.1
  -- Find the child or children of the node requested.
  REPLY set{node} -- return child node(s)

  MESSAGE create_root_node(node) -- G1.1, G1.2
  -- Create a new node and insert into the top of the hierarchy
  WHEN? -- node created
    REPLY done
  TRANSITION? -- add root

  MESSAGE create_child_node(node) -- G1.1, G1.2
  -- Create a new node and insert into the hierarchy correctly
  REPLY done
  TRANSITION? -- add node

  MESSAGE delete_node(node) -- G1.1, G1.2
  -- Find the node and delete it and all children from the structure.
  WHEN? -- node found
    REPLY done
  TRANSITION? -- remove node and children
  OTHERWISE REPLY EXCEPTION node does not exist

  MESSAGE traverse_tree(node) -- G2
  -- Find the requested root node and display all children if they all
  -- consist of PSDL specifications.
  WHEN? -- node found
    REPLY set{node}
  OTHERWISE REPLY EXCEPTION psdl program does not exist
END

```

The goal of functional specification is to construct a model of the proposed system as it is visible to the users [Ref. 5]. The concepts that the users will be expected to know and the details of the interfaces are defined. The functional specification does not include any information on how the system behavior is to be realized. The result is a set of definitions for the system concepts and interfaces. The major functions of the DDB are:

- Store the levels of a PSDL program in a hierarchical format by specification.
- Retrieve the levels of a PSDL program in a hierarchical format by specification for review or editing.
- Create and insert new levels of a PSDL program in a hierarchical format by specification.
- Generate the entire PSDL program.

A brief example to illustrate the expected patterns of use of the methods provided by the DDB follows. To construct a prototype, the PSDL specification of the root operator is entered. At this point, the DDB would create a root node. Assuming the root node is composite, the node would be decomposed into children operators. The DDB would create child nodes for each decomposition. The decomposition process would continue until all leaf nodes are atomic. This process will be time consuming and the prototype will be complex. For this reason, the functions of retrieving nodes, parent-child relationships, and deleting nodes will be required. The tree will be traversed and the entire PSDL program produced once all leaf nodes are atomic.

## 2.5. Architectural Design

The next step in the design process is to develop an architectural design. An architectural design is a model of the proposed system capturing the aspects of its behavior and structure relevant to the development team. The behavior of a system consists of its interactions with other systems, while the structure of a system consists of its component parts and their interconnections. The functional specification is a subset of the architectural design. The functional specification is the least detailed view of the system. [Ref. 5]

The goal of the architectural design is to break up the proposed system into a set of small modules. A module is defined as a self contained unit of code.

A module is both a self-contained abstraction and a unit of work. Modules have several different views: black-box specification, parts lists, glass-box specifications, and programs. Black-box specifications are expressed in terms of the event model at the architectural design and functional specification stage. The parts lists contains the set of lower level modules used directly in the implementation. Glass-box specifications are represented by pseudo-code. Programs are produced in the implementation phase. [Ref. 5] The black-box specifications will be shown below. The parts lists and pseudo-code are not contained. Programs will be discussed in section 3.

## Black-box specifications

### Type Node

Model (specification implementation graphic\_record  
control\_constraints: string)

- The following messages are used to replace the current value of
- a node's property with a new value.

MESSAGE add\_graphic\_record(node)

TRANSITION? -- update node graphic record info

MESSAGE add\_implementation(node)

TRANSITION? -- update node implementation info

MESSAGE add\_specification(node)

TRANSITION? -- update specification info

MESSAGE add\_control\_constraints(node)

TRANSITION? -- update control constraints

END

The result is a lower level set of definitions for the system concepts and interfaces. These messages reveal operations on the abstract data type node. The lower level messages in the user\_interface interface are:

add\_graphic\_record  
add\_implementation  
add\_specification  
add\_control\_constraints

The addition of these messages creates an additional function requirement:

- Create and maintain version control.

A new node could be created when a node's properties are significantly changed by these low level messages. The ability to define a significant change will be required. The current implementation of the DDB does not address this function. As evidenced by the functions required of the DDB, a conventional DBMS will not suffice. Therefore, an object-oriented DBMS will be used to design and implement the DDB. The object-oriented DBMS that will be used for experimenting the concepts of the CAPS Design Database is Vbase by Ontologic Incorporation.

## 3. EXPERIMENT ON DDB USING AN OBJECT-ORIENTED DBMS

There is an increasing interest in developing object-oriented database management systems to manage the large amount of data involved in computer-aided design (CAD) applications [Ref. 6]. In the past three years, several object-oriented database management systems have emerged. The impact of these systems on the software development process are just beginning to be felt. The object-oriented approach represents a true paradigm shift [Ref. 7].

An object-oriented database management system (OODBMS) must provide persistence, concurrency, recovery, transaction management, authorization, and security [Ref. 6]. An OODBMS is an object-oriented system and as such it must also provide the following capabilities:

- Objects
- Active data
- Abstraction
- Extensibility [Ref. 6]

An OODBMS should provide application-oriented capabilities such as:

- Version and configuration control for CAD applications.
- Dynamic creation of classes.
- Recursive classes, multiple inheritance, and extensive tool interface capabilities.
- Support for multimedia objects, distributed environments, and graphics. [Ref. 6]

An object-oriented DBMS is one that supports persistency, values, an extensible set of data structures, an extensible set of operations, and abstractions [Ref. 6].

### 3.1. Vbase

The Vbase object-oriented database management system is a product of Ontologic, Inc. Vbase is an object database system, providing an integrated software development platform which combines the latest advances in compiler design and database management techniques. The Vbase Integrated Object System is a database and language platform for rapidly and inexpensively building sophisticated commercial and engineering applications. [Ref. 8]

The Vbase system environment consists of the following components:

- Vbase Database. Persistent objects are stored in the Vbase Database.
- Object Language. Type Definition Language (TDL) is the Vbase data definition language. It is used to define object types in the database. "C" Object Processor (COP) is the Vbase data manipulation language. It is an object extension of Kernighan and Ritchie standard C. It is used to implement the operations of the object types defined using TDL.
- System Type Library. The system type library contains many object types which provide powerful building blocks for the application developer.
- Integrated Tool Set (ITS). A single tool combining the functionality of a source browser, database browser, and a source debugger.
- Object SQL. The Vbase implementation of the SQL standard query language. [Ref.8]

The steps involved in a typical Vbase database design are listed below:

- Identify the objects.
- Identify their properties as much as possible.
- Identify the frequent operations performed on the objects.
- Define the objects using TDL.
- Compile and debug the TDL definition of the objects.
- Develop COP routines to implement the operations.
- Compile and debug the COP programs.
- Develop C or COP user applications. [Ref. 6]

Vbase is a powerful tool for implementing and maintaining large software applications. Its integration of compiler technology with database functionality in a



strongly-typed system provides both sophisticated modeling and efficient language and database performance.

### 3.2. Type Definition for Node Objects in the Design Database

As stated earlier, the first steps in a Vbase design are to identify the objects, their properties, and the frequent operations performed on the objects. The next step is to define the objects in TDL. The only object in the Design Database is a Node. The properties of a node were defined earlier as well. The frequent operations are those necessary to assist in the accomplishment of the required functions of the design database. The TDL code below illustrates the definition of a Node.

```
define Type Node
  supertypes = {Entity}
  properties = {
    name: String;
    specification: String;
    implementation: optional String;
    controlconstraints: optional String;
    graphicrecord: optional String;
    subNodes: distributed Set[Node] inverse $Node$isChildOf;
    isChildOf: optional Node inverse $Node$subNodes;
  }
```

The main points to notice about the above definition are the supertypes, optional, and inverse keywords. The supertypes declaration is used to show the parent class of a type. This is used for inheritance purposes. The supertype Entity is the root of all types in the Vbase system type library [Ref. 8]. Entity specifies basic behavior for all object types in the system.

The keyword optional specifies that a property need not necessarily have a value. This indicates that a PSDL specification may or may not have an implementation, control constraints, or graphic record. The implementation property is optional due to the process through which PSDL specifications are created. The control constraints and graphic record are true optional properties in that they may or may not ever exist depending on whether implementation is graphic or ADA.

The inverse property sets up a system-maintained relationship between the property defined and its specified inverse. This property is used to maintain the parent-child relationship between Nodes. The inverse property also illustrates the "\$" notation. The "\$" notation is used to provide a mechanism for referring to names relative to their scope. The symbol "\$" acts as a pathname separator.

The next step is to define the frequent operations on the object. The clause "operations = {...};" defines the set of operations that type Node will implement. The operations for a Node are buildDisplay, listsubNodes, listsubNodesInternal, and a refinement of the delete operation. BuildDisplay is used for output of a Node contents. ListsubNodes and listsubNodesInternal are used to retrieve the children of a particular Node. The refines delete operation means the current definition is refining an operation already defined in the supertype. The actual refinement is achieved in the COP method

which implements the operation and will be described later. The operations for a Node are defined as:

```

operations = {
  buildDisplay (n:Node,)
    returns(Node)
    method (NodeBuildDisplay);
  listSubnodes (n:Node)
    returns (Set[Node])
    method (NodeListSubNodes);
  listSubnodesInternal (n:Node, s:Set [Node])
    returns (Set[Node])
    method (NodeListSubNodesInt);
  refines delete (n:Node)
    raises (CannotDelete)
    triggers (NodeDeleteTrigger);
};

```

The definition of a Node includes two procedure definitions:

```
define Procedure Create ... end Create;
```

and

```
define Procedure Lookup ... end Lookup;
```

Procedures differ from typed operations in that they are not tied to a type. For example, the operation `$Node$buildDisplay` can only be called on instances of type `Node`, while the procedure `$Node$Lookup` can be called with any arguments which satisfy the argument type specification of the procedure. The procedure `Create` has an argument specification of the form:

```

define Procedure Create
(t:Type,
  keywords
  name: String,
  specification: String,
  optional implementation: String,
  optional controlconstraints: String,
  optional graphicrecord: String,
  optional isChildOf: Node,
  optional where: Entity,
  optional hownear: Clustering
)
returns (Node)
raises (NodeAlreadyExists)
triggers (NodeCreateTrigger)
supertypes = {$Entity$Create};
end Create;

```

This specification gives more examples of the power of Vbase. Specifically, the keywords "keywords", "raises", "triggers", and "hownear". The keyword "keywords"

specifies that the remaining arguments are passed by keywords, rather than by position in the argument list.

The statement "raises (NodeAlreadyExists)" specifies that the procedure may raise the exception NodeAlreadyExists. This exception is to alert the caller that the Node already exists, rather than creating a new copy of the Node.

The Create procedure also has a triggers clause, "triggers = (NodeCreateTrigger)". The Vbase system automatically generates a Create Procedure for every type defined. An explicit definition is required to specify a trigger to the system-defined Create. A trigger is a method associated with the invocation of a procedure or operation. Whenever the Create procedure is invoked, the trigger, NodeCreateTrigger, will be executed first. The trigger checks whether a Node already exists before creating a new one. An operation can have more than one trigger associated with it. The triggers are invoked in the order they are listed, and the last trigger must invoke the base method. The base method is the method which is specified as implementing the operation or procedure. [Ref. 8]

The optional keywords where and hownear can be used in the Create operation to optimize disk storage of the object created to improve database performance. The type Clustering is an enumerated type with three instances: \$area, \$segment, and \$chunk. Each is a unit of storage on the disk. Segment is the atomic unit of transfer from disk to the main memory cache. To specify that an object created is to be stored in the same segment as some other object, the value of the hownear argument is "\$segment" and the value of the where argument is the other object. \$Area and \$Chunk are not currently supported. [Ref. 8]

The second procedure defined is called Lookup. This procedure is used to look up a given Node, identified by its Node name, in NodeCatalog which is a global Node catalog. The specification for procedure Lookup is as follows:

```
define Procedure Lookup (s:String)
  returns (Node)
  raises (NodeNotInCatalog)
  method (NodeLookup)
  supertypes = {Entity};
end Lookup;
```

There is one additional TDL definition in the DDB, NodesExceptions.tdl. NodesExceptions contains the definitions of the exceptions used in the application.

### 3.3. Operations on Node Objects

The next step is to implement the frequent operations using COP. COP is an object-based superset of the language C. One method implemented for the object Node was "NodebuildDisplay". This operation is used to output the contents of a Node to a file. The COP code below implements the method:

```
#include <stdio.h> /* include standard C routines */
#include <string.h>
```

```

#define MAXLINE 81 /* maximum line length is 81 characters */
#define MAXSTRING 4000 /* maximum string length is 4000 */
char opname[MAXLINE]; /* declaration of local variables */
FILE *out;
char spectext[MAXSTRING],
    imptext[MAXSTRING],
    cctext[MAXSTRING];
import $Type;
import $Class;
enter module $Node;
method
obj $Node
NodeBuildDisplay(aNode)
obj $Node aNode;

{
    out = fopen("ddb.out", "a");
    /* convert object code to C code */

    AM_stringToC(aNode.name,opname,sizeof(opname));
    fprintf(out,"%s\n",opname);
    AM_stringToC(aNode.specification, spectext, sizeof(spectext));
    fprintf (out,"%s\n", spectext);

    /* determine if optional property has a value */
    /* before executing an operation on it. */

    if (hasvalue(aNode.implementation))
    {
        AM_stringToC(aNode.implementation, imptext, sizeof(imptext));
        fprintf(out, "%s\n", imptext);
    }
    if (hasvalue(aNode.controlconstraints))
    {
        AM_stringToC(aNode.controlconstraints, cctext, sizeof(cctext));
        fprintf(out, "%s\n", cctext);
    }
    fclose(out);
    return(aNode);
}

```

This example helps to show the ability to interweave the standard C language with COP. Object code and variables are prefaced by the "\$" symbol. This is used to distinguish object variables from standard C variables.

The declarative statements "import" and "enter module" are used for name visibility. Making a name visible provides the COP compiler with a reference to what the name defines. Database names are defined in the Vbase Kernel Database and in



TDL code. Names defined in the Vbase Kernel Database are globally defined in a default set. Names not included in the default set must be made visible explicitly using the "import" and "enter module" statements. An "import" declaration imports the definitions of a set of database names so they are visible within the current COP compilation unit. An "enter module" declaration establishes visibility for all names defined in a module. [Ref. 8]

The functions "hasvalue" and "AM\_stringToC" are system supplied. The function "hasvalue" is used to test whether an optional property has a value before executing any operations on it. This is necessary because of the strong-typing of Vbase. The function "AM\_stringToC" is used to convert from an object string to a standard C string for use by systems external to Vbase.

Two other operations defined where "NodeListSubNodes" and "NodeListSubNodesInternal". These operations are used to assist in the traversal of the tree structure. The COP code to implement these operations is shown below:

```
method
obj $Set[obj $Node]
NodeListSubNodes(aNode)
obj $Node aNode;
{
    obj $Set[obj $Node] theSubNodes;
    theSubNodes = $Set[obj $Node]{};
    $Node$ListSubNodesInternal(aNode, theSubNodes);
    return(theSubNodes);
}

method
obj $Set[obj $Node]
NodeListSubNodesInt(aNode, theSubNodes)
obj $Node aNode;
obj $Set[obj $Node] theSubNodes;
{
    obj $Node currentNode;
    iterate(currentNode = aNode.SubNodes)
    {
        $Set$Insert(theSubNodes, currentNode);
        $Node$ListSubNodesInternal(currentNode, theSubNodes);
    }
    return(theSubNodes);
}
```

This code helps to demonstrate other powerful capabilities of Vbase. One is the ability of one method to invoke another method. This is shown in the method "NodeListSubNodes". The other capability is the system supplied iterator operation. This operation is used to control aggregate types. The system defined iterator can be modified to return the aggregate in any order the user decides.

### 3.4. Application Programs

The final step in a Vbase design is to develop C or COP user applications. The user applications developed correspond to the functional specifications and architectural design. The user applications developed in response to the functional specifications are:

- CreateRootNode. Used to create a Node which is the root of a tree.
- CreateChildNode. Used to create a Node which is a child of a Node.
- GetParent. Used to retrieve the name of a Node's parent Node.
- GetChildren. Used to retrieve the name(s) of a Node's child Node(s).
- DeleteNode. Used to delete a Node and it's children from the tree.
- TraverseTree. Used to traverse the entire tree structure to generate the PSDL program.

The user applications developed in response to the architectural design are:

- StoreProperty. Used to update, insert, or change the value of a Node's property.
- GetProperty. Used to retrieve the contents of a Node's property.

The applications were all implemented using COP. The actual code for some of these applications are quite long, therefore the code for TraverseTree will be shown here for demonstration purposes. This application takes as input the name of the root Node of a tree. It then iterates through the entire tree writing the properties of a Node to the output file "ddb.out".

```
#include <stdio.h> /* include standard C routines */
#include <string.h>
#define LINELENGTH 80
#define MAXLINE 81 /* maximum linelength is 81 characters */
FILE *in, *out; /* local variable declarations */
char rootname[MAXLINE],
    tempname[MAXLINE];
import $Node;
main(argc, argv)
int argc;
char **argv;
{
    /* local object variables */
    char *dbname;
    char *getenv();
    obj $Node theNode, currentNode;
    obj $Set[obj $Node] theNodes;
    obj $String theRoot;
    if (argc > 1)
    {
        dbname = argv[1];
    }
    else if (dbname = getenv("DBNAME"))
    {
    }
    else
```

```

    {
        printf("Must specify database name, either as a command line
argument,\nor via the Unix environment variable DBNAME\n");
        exit(1);
    }
    {
        AM_databaseOpen(dbname, 0);
        in = fopen("ddb.in", "r");
        out = fopen("ddb.out", "w");
        fclose(out);
        fgets(tempname, LINELENGTH, in);
        strncpy(rootname, tempname, (strlen(tempname) - 1));
        theRoot = rootname;
        theNode = $Node$Lookup(theRoot);

        (void) $Node$BuildDisplay(theNode);

        theNodes = $Node$listSubnodes(theNode);
        iterate(currentNode = theNodes)
        {
            (void) $Node$BuildDisplay(currentNode);
        }
    }
    protect
        AM_databaseClose(dbname);
}

```

The above code illustrates two additional keywords: "void" and "protect". "Void" is a standard C keyword, indicating that the method does not return a value. The method simply outputs the information passed to it.

"Protect" ensures execution of a statement when an exception is raised. "Protect AM\_databaseClose(dbname)" ensures the database involved is closed in the event an exception is raised.

#### 4. CONCLUSIONS

The development of hard, real-time software systems continues to be an expensive process for the DOD. The Computer-Aided Prototyping System (CAPS) is one tool under development which will help to decrease the costs of these systems. CAPS is an attempt to integrate several of the prevailing software development methodologies into one tool. With a central theme of rapid prototyping, CAPS shows great promise for the future of software development in the DOD.

This paper concentrated on the development of the Design Database (DDB) for CAPS. It is a key element of the system as project management has become an issue of increasing importance in software development. A robust Design Database which can efficiently and effectively, store and retrieve the Prototype System Description Language (PSDL) program will significantly contribute to the overall success of CAPS.

The goal of this paper has been to develop a conceptual level design and initial implementation of the Design Database for CAPS. The basic design was developed using the object-oriented approach and the initial implementation was accomplished with an object-oriented DBMS (Vbase). Object-oriented technology offers several enhancements to current DBMS technology, and with its maturity it will become as important to CAD applications as relational database technology has become to business applications.

1. Luqi and Ketabchi, M., *A Computer Aided Prototyping System*, Tech. Rep. NPS 52-87-011, Naval Postgraduate School, Monterey, CA, 1987 and in *IEEE Software*, pp. 66-72, March 1988.
2. Booch, G., *Software Engineering with Ada*, Benjamin Cummings Publishing Co., Inc., Menlo Park, CA, 1983.
3. Chitwood, G., "Ada Meets the Challenge of Real-Time Simulation," *Defense Computing*, v. 1, no. 4, pp.32-38, July/August 1988.
4. Berzins, V., and Luqi, *An Introduction to the Specification Language SPEC*, Tech. Rep. NPS 52-88-031, Naval Postgraduate School, Monterey, CA, 1988 and to appear in *IEEE Software*, 1989.
5. Berzins, V. and Luqi, *Software Engineering with Abstractions: An Integrated Approach to Software Development using Ada*, Addison-Wesley, 1989.
6. Ketabchi, M., *Object Oriented Database Management Systems for Complex Data and Process Intensive Applications*, September 1988.
7. McKenna, J., "Teaching OOP," *OOPSLA '88 Conference Proceedings*, The Association for Computing Machinery, New York, NY, 1988.
8. *Vbase Integrated Object Database User's Manual*, Ontologic Inc., Billerica, MA, 1987.
9. Luqi, Berzins, V., Yeh, R., *A Prototyping Language for Real-Time Software*, *IEEE TSE*, October 1988.
10. Thorstenson, R., *A Graphical Editor for the Computer Aided Prototyping System (CAPS)*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.
11. Raum, H., *The Design and Implementation of an Expert User Interface for the Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.

12. Porter, S., *Design of a Syntax Directed Editor for PSDL*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.
13. Galik, D., *A Conceptual Design of a Software Base Management System for the Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.
14. Douglas, B., *A Conceptual Level Design of a Design Database for the Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989.
15. Altizer, C., *Implementation of a Language Translator for a Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.
16. Moffitt II, C., *A Language Translator for a Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.
17. O'Hern, J., *A Conceptual Level Design for a Static Scheduler for Hard Real-Time Systems*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.
18. Janson, D., *A Static Scheduler for the Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.
19. Marlowe, L., *A Scheduler for Critical Timing Constraints*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.
20. Eaton, S., *A Dynamic Scheduler for a Computer Aided Prototyping System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1988.
21. Wood, M., *Run Time Support for Rapid Prototyping*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988.
22. Luqi, *Rapid Prototyping for Large Software System Design*, Ph.D. Dissertation, University of Minnesota, Duluth, Minnesota, May 1986.

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2  
Cameron Station  
Alexandria, Virginia 22304-6145
2. Library, Code 0142 2  
Naval Postgraduate School  
Monterey, California 93943-5002
3. Office of Naval Research 1  
Office of the Chief of Naval Research  
Attn. CDR Michael Gehl, Code 1224  
800 N. Quincy Street  
Arlington, Virginia 22217-5000
4. Space and Naval Warfare Systems Command 1  
Attn. Dr. Knudsen, Code PD 50  
Washington, D.C. 20363-5100
5. Ada Joint Program Office 1  
OUSDRE(R&AT)  
Pentagon  
Washington, D.C. 20301
6. Naval Sea Systems Command 1  
Attn. CAPT Joel Crandall  
National Center #2, Suite 7N06  
Washington, D.C. 22202
7. Office of the Secretary of Defense 1  
Attn. CDR Barber  
STARS Program Office  
Washington, D.C. 20301
8. Office of the Secretary of Defense 1  
Attn. Mr. Joel Trimble  
STARS Program Office  
Washington, D.C. 20301
9. Commanding Officer 1  
Naval Research Laboratory  
Code 5150  
Attn. Dr. Elizabeth Wald  
Washington, D.C. 20375-5000



10. Navy Ocean System Center 1  
Attn. Linwood Sutton, Code 423  
San Diego, California 92152-500
11. National Science Foundation 1  
Attn. Dr. William Wulf  
Washington, D.C. 20550
12. National Science Foundation 1  
Division of Computer and Computation Research  
Attn. Dr. Peter Freeman  
Washington, D.C. 20550
13. National Science Foundation 1  
Director, PYI Program  
Attn. Dr. C. Tan  
Washington, D.C. 20550
14. Office of Naval Research 1  
Computer Science Division, Code 1133  
Attn. Dr. Van Tilborg  
800 N. Quincy Street  
Arlington, Virginia 22217-5000
15. Office of Naval Research 1  
Applied Mathematics and Computer Science, Code 1211  
Attn: Dr. James Smith  
800 N. Quincy Street  
Arlington, Virginia 22217-5000
16. New Jersey Institute of Technology 1  
Computer Science Department  
Attn. Dr. Peter Ng  
Newark, New Jersey 07102
17. Southern Methodist University 1  
Computer Science Department  
Attn. Dr. Murat Tanik  
Dallas, Texas 75275
18. Editor-in-Chief, IEEE Software 1  
Attn. Dr. Ted Lewis  
Oregon State University  
Computer Science Department  
Corvallis, Oregon 97331
19. University of Texas at Austin 1  
Computer Science Department  
Attn. Dr. Al Mok  
Austin, Texas 78712

20. University of Maryland  
College of Business Management  
Tydings Hall, Room 0137  
Attn. Dr. Alan Hevner  
College Park, Maryland 20742 1
21. University of California at Berkeley  
Department of Electrical Engineering and Computer Science  
Computer Science Division  
Attn. Dr. C.V. Ramamoorthy  
Berkeley, California 94720 1
22. University of California at Los Angeles  
School of Engineering and Applied Science  
Computer Science Department  
Attn. Dr. Daniel Berry  
Los Angeles, California 90024 1
23. University of Maryland  
Computer Science Department  
Attn. Dr. Y. H. Chu  
College Park, Maryland 20742 1
24. University of Maryland  
Computer Science Department  
Attn. Dr. N. Roussapoulos  
College Park, Maryland 20742 1
25. Kestrel Institute  
Attn. Dr. C. Green  
1801 Page Mill Road  
Palo Alto, California 94304 1
26. Massachusetts Institute of Technology  
Department of Electrical Engineering and Computer Science  
545 Tech Square  
Attn. Dr. B. Liskov  
Cambridge, Massachusetts 02139 1
27. Massachusetts Institute of Technology  
Department of Electrical Engineering and Computer Science  
545 Tech Square  
Attn. Dr. J. Guttag  
Cambridge, Massachusetts 02139 1
28. University of Minnesota  
Computer Science Department  
136 Lind Hall  
207 Church Street SE  
Attn. Dr. J. Ben Rosen  
Minneapolis, Minnesota 55455 1

29. International Software Systems Inc. 1  
12710 Research Boulevard, Suite 301  
Attn. Dr. R. T. Yeh  
Austin, Texas 78759
30. Software Group, MCC 1  
9430 Research Boulevard  
Attn. Dr. L. Belady  
Austin, Texas 78759
31. Carnegie Mellon University 1  
Software Engineering Institute  
Department of Computer Science  
Attn. Dr. Lui Sha  
Pittsburgh, Pennsylvania 15260
32. IBM T. J. Watson Research Center 1  
Attn. Dr. A. Stoyenko  
P.O. Box 704  
Yorktown Heights, New York 10598
33. The Ohio State University 1  
Department of Computer and Information Science  
Attn. Dr. Ming Liu  
2036 Neil Ave Mall  
Columbus, Ohio 43210-1277
34. University of Illinois 1  
Department of Computer Science  
Attn. Dr. Jane W. S. Liu  
Urbana Champaign, Illinois 61801
35. University of Massachusetts 1  
Department of Computer and Information Science  
Attn. Dr. John A. Stankovic  
Amherst, Massachusetts 01003
36. University of Pittsburgh 1  
Department of Computer Science  
Attn. Dr. Alfs Berztiss  
Pittsburgh, Pennsylvania 15260
37. Defense Advanced Research Projects Agency (DARPA) 1  
Integrated Strategic Technology Office (ISTO)  
Attn. Dr. Jacob Schwartz  
1400 Wilson Boulevard  
Arlington, Virginia 22209-2308

38. Defense Advanced Research Projects Agency (DARPA) 1  
Integrated Strategic Technology Office (ISTO)  
Attn. Dr. Squires  
1400 Wilson Boulevard  
Arlington, Virginia 22209-2308
39. Defense Advanced Research Projects Agency (DARPA) 1  
Integrated Strategic Technology Office (ISTO)  
Attn. MAJ Mark Pullen, USAF  
1400 Wilson Boulevard  
Arlington, Virginia 22209-2308
40. Defense Advanced Research Projects Agency (DARPA) 1  
Director, Naval Technology Office  
1400 Wilson Boulevard  
Arlington, Virginia 2209-2308
41. Defense Advanced Research Projects Agency (DARPA) 1  
Director, Strategic Technology Office  
1400 Wilson Boulevard  
Arlington, Virginia 2209-2308
42. Defense Advanced Research Projects Agency (DARPA) 1  
Director, Prototype Projects Office  
1400 Wilson Boulevard  
Arlington, Virginia 2209-2308
43. Defense Advanced Research Projects Agency (DARPA) 1  
Director, Tactical Technology Office  
1400 Wilson Boulevard  
Arlington, Virginia 2209-2308
44. MCC AI Laboratory 1  
Attn. Dr. Michael Gray  
3500 West Balcones Center Drive  
Austin, Texas 78759
45. COL C. Cox, USAF 1  
JCS (J-8)  
Nuclear Force Analysis Division  
Pentagon  
Washington, D.C. 20318-8000
46. LTCOL Kirk Lewis, USA 1  
JCS (J-8)  
Nuclear Force Analysis Division  
Pentagon  
Washington, D.C. 20318-8000

- |     |   |   |
|-----|---|---|
| 47. | University of California at San Diego<br>Department of Computer Science<br>Attn. Dr. William Howden<br>La Jolla, California 92093             | 1 |
| 48. | University of California at Irvine<br>Department of Computer and Information Science<br>Attn. Dr. Nancy Levenson<br>Irvine, California 92717  | 1 |
| 49. | University of California at Irvine<br>Department of Computer and Information Science<br>Attn. Dr. L. Osterweil<br>Irvine, California 92717    | 1 |
| 50. | University of Colorado at Boulder<br>Department of Computer Science<br>Attn. Dr. Lloyd Fosdick<br>Boulder, Colorado 80309-0430                | 1 |
| 51. | Santa Clara University<br>Department of Electrical Engineering and Computer Science<br>Attn. Dr. M. Ketabchi<br>Santa Clara, California 95053 | 1 |
| 52. | Oregon Graduate Center<br>Portland (Beaverton)<br>Attn. Dr. R. Kieburz<br>Portland, Oregon 97005  | 1 |
| 53. | Dr. Wolfgang Halang<br>Bayer AG<br>Ingenieurbereich Progressleittechnik<br>D-4047<br>Dormagen, West Germany                                   | 1 |
| 54. | Dr. Bernd Kraemer<br>GMD Postfach 1240<br>Schloss Birlinghoven<br>D-5205<br>Sankt Augustin 1, West Germany                                    | 1 |
| 55. | Dr. Aimram Yuhudai<br>Tel Aviv University<br>School of Mathematical Sciences<br>Department of Computer Science<br>Tel Aviv, Israel 69978      | 1 |

56. Dr. Robert M. Balzer 1  
USC-Information Sciences Institute  
4676 Admiralty Way  
Suite 1001  
Marina del Ray, California 90292-6695
57. U.S. Air Force Systems Command 1  
Rome Air Development Center  
RADC/COE  
Attn. Mr. Samuel A. DiNitto, Jr.  
Griffis Air Force Base, New York 13441-5700
58. U.S. Air Force Systems Command 1  
Rome Air Development Center  
RADC/COE  
Attn. Mr. William E. Rzepka  
Griffis Air Force Base, New York 13441-5700
59. LuQi 50  
Code 52Lq  
Computer Science Department  
Naval Postgraduate School  
Monterey, CA 93943-5100
60. Research Administration 1  
Code: 012  
Naval Postgraduate School  
Monterey, CA. 93943