The 3DIS: An Extensible Object-Oriented Framework for Information Management

Technical Report CRI-85-21
October 11, 1985

Hamideh Afsarmanesh Tehrani

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Science)

July 1985

Acknowledgements

I would like to express my appreciation to my advisor, Professor Dennis McLeod for his continuous guidance and support during this research. Other members of my dissertation committee, Professor Amihai Motro, and Professor Alice Parker deserve special thanks for their invaluable technical help and comments on my final draft. I am grateful to Professor Richard Hull for his interest in this research, valuable advice, and careful reading of the early drafts of this dissertation.

I would also like to thank all of my friends at USC for their moral support and encouragement. In particular, I am indebted to David Knapp for his candid involvement and friendship in the course of the development of the VLSI database, Dennis Heimbigner for his comments and help to shape many of my early ideas, and Alice Parker for her sincere friendship and personal concern.

I am grateful to my parents, Mehri and Esmaiel Afsarmanesh, with all my heart for their love and encouragement. Finally, I wish to thank my husband, Farhad, for more than I can ever acknowledge here, for his love, optimism, unending moral support, and confidence in me.

This research was supported, in part, by the Joint Services Electronics Program through the Air Force Office of Scientific Research under contract #F49620-81-C-0070. The research and development of the VLSI database

Table of Contents

List of Figures

project was also supported, in part, by the National Science Foundation under grant #MCS-8203485.

Abstract

This dissertation presents an extensible information modeling framework, the 3 Dimensional Information Space (3DIS), and a simple user interface, the Information Sub-Language (ISL), to support its user-database interactions. 3DIS is mainly intended for applications that have highly dynamic and complex structures, and whose end-users are non-database-experts, who, nevertheless, become the designers, manipulators, and evolvers of their databases. A unique characteristic of the 3DIS is that it models all data and the descriptive information about data (meta-data) uniformly. All identifiable information, of various kinds and levels of abstraction, is viewed and modeled by the same constructs as objects. Consequently, the data definition and data manipulation languages merge into a single database language. Basic relationships among objects are defined through the three fundamental abstraction mechanisms of classification/instantiation, aggregation/decomposition, and generalization/specialization. Significantly, the model is extensible to accommodate other kinds of abstractions; in particular, it has been extended to support the definition of recursive entities and concepts such as sets, lists, and binary trees.

An integral part of the 3DIS model is its simple and multi-purpose geometric representation. This geometric framework graphically organizes both structural and non-structural database information in a 3-D representation

space. It reflects a semi-formal definition for the 3DIS modeling constructs in terms of the geometric components that represent them. This representation also provides an appropriate foundation for information browsing and serves as an environment for a simple graphics-based user interface. Database browsing is supported by a "display window" to the geometric framework, through which users may focus on and investigate an object or an "information neighborhood" of their interest. ISL provides users with a simple set of basic primitives for object definition, manipulation, and retrieval. Since all concepts and entities are modeled as objects, these primitives can be used to view and handle information at any level of abstraction. We have developed an experimental ISL prototype that is a single-user, browsing-oriented user interface to 3DIS databases.

The class of application environments for which the 3DIS information model and its user interface are suitable include personal information management systems, office automation systems, and information systems to support engineering design, specifically, VLSI and software engineering. In particular, a VLSI design environment for a project (ADAM) at the USC Electrical Engineering Department was studied as an application for the 3DIS information model and its user interface. Necessary extensions were made to the 3DIS to support the modeling needs of this environment, e.g. to allow the recursive definition of VLSI cells.

Chapter One

Introduction

In modern society, computerized database systems are essential and inseparable components of many public and private organizations. Database systems are utilized at all levels of management, research, and production to provide convenient access to consistent information. Applications in which the use of database systems are critical include the large commercial applications such as banking, accommodation and ticket reservation, personnel, and inventory systems. However, relatively smaller and more special purpose applications are recently becoming considerably more important and appealing. Examples of these specialized applications come from personal information management systems, office automation systems, software engineering, and information systems to support engineering design (specifically CAD/CAM and VLSI).

The work described in this dissertation has two main objectives. The first is to introduce a suitable information modeling framework, the 3DIS (3 Dimensional Information Space), to address several modeling needs of a large class of recent special purpose application environments. The second goal is to devise a simple user interface, the ISL (Information Sub-Language), to support

the interaction needs of the potential end-users of 3DIS databases. ISL is intended to allow a wider range of database manipulation methods than are conventionally provided on databases; for example, to support database browsing as well as a query language.

Several characteristics and directions suggested by the 3DIS are unique in database modeling and distinct from those of most other recent database models. A fundamental concept in this work is to model data and the descriptive information about data (schema or meta-data) uniformly. This makes it equally easy for database users to model, view, access, and query the structural information, and the data content of databases. It also simplifies user-database communications by unifying the data definition and data manipulation languages into a single database language. A single database language permits the same database language constructs to be used in querying and updating both the schema and the data-content of databases. A second concept underlying the 3DIS is to model various kinds of data (including formatted and unformatted data) uniformly, so that they can be manipulated by a single set of basic operations. By and large, the 3DIS provides a homogeneous environment where many classes of database information can coexist and their manipulation is simplified.

The 3DIS database model proposes a specific generalization hierarchy that contains a small number of predefined (structural) objects to organize the common structural information of application environments. The hierarchy is a

unifying super-structure that fits on top and connects to the structural graphs of 3DIS databases. The resulting hierarchies support investigation of the structural information of databases without requiring any prior knowledge about them. It should be emphasized that the predefined structural objects do not introduce any specific database-dependent descriptions or classifications, but rather describe the generic kinds of structural information that are common to many specialized application environments, as well as a wide variety of commercial databases.

A fundamental part of the 3DIS model is its geometric representation. The geometric representation (cube) of a 3DIS database is simple and serves the following purposes:

1. It organizes both structural and non-structural database information graphically and supports their uniform handling.
2. It provides an appropriate foundation for information browsing and serves as an environment for a simple graphics-based user interface.
3. It reflects a formal, mathematically founded definition for the 3DIS modeling constructs in terms of the geometric components representing them.
4. Through its geometric components (points, lines, and planes), it encapsulates database information at several levels of abstraction.

The variety of information encapsulation supported by the geometric representation is a unique feature of the 3DIS.

The ISL database language offers a browsing-oriented user interface to traverse the geometric representation of 3DIS databases. A simple set of "navigational" operations allow browsing through the geometric representation space. With certain "object specification" operations users can define, query, manipulate, and evolve database information at various levels of abstraction. The ISL uses the format of an ordered triplet for its query operations. The same format corresponds to the components in the geometric representation space of the 3DIS. Consequently, geometric components of the representation space are suggestive of answers to many simple retrieval queries.

The remainder of this chapter is organized as follows. Section 1.1 contains background to database modeling and user interfaces. Section 1.2 contains a statement of the specific problems addressed by this research. Section 1.3 presents an overview of the approach taken for providing the 3DIS conceptual database model and user interface; this includes a discussion of several properties and limitations of the 3DIS model, and the database modeling directions and characteristics unique to this research. Section 1.4 is a guide to the subsequent chapters of this dissertation.

1.1. Background

Database models provide the conceptual basis for describing and viewing application environments, while user interfaces provide the basic tools and techniques for use and manipulation of database systems. This section provides

an overview of these two areas and defines some of the terms that are later used in this dissertation.

1.1.1. Database Modeling

A continuum of database models, ranging from computer-oriented (less semantic) to human-oriented (more semantic) has been proposed during the past two decades. Every database model suggests a collection of modeling constructs and mechanisms to characterize and describe concepts within application environments.

The elementary forms of database systems consist of large repositories of data (files), and file management systems to control users' access to the files. File management systems support some record-level consistency of data but semantics of data, e.g. the description (meaning) of the content of each field of a file's record, remain outside of the database and entirely subject to the interpretation of users or application programs. There were several problems with not having this description computerized. For instance, unless the intention (meaning) of information was somehow documented heavily and all users had access to such centralized documents, database information could be erroneously misinterpreted by users or application programs. Since those early days, research in database modeling has tried to capture more of the knowledge about the meaning of data and include it in database systems as guides for their proper use.

A predominant research theme in databases during the past two decades has been the importance of separating the user-level, meaning-based views of a database from its machine-level representation. Classical business-oriented database systems define three levels of abstraction [Date 81, Ullman 82], as specified by the ANSI/SPARC committee in 1975 [ANSI 75]. A level of physical representation of data (internal database model) describes the storage structure of a database. Two additional levels of logical modeling, a global-logical view (conceptual database model) and a collection of specialized logical views (external database models) provide higher levels of abstraction through which end-users access a database. The focus of this dissertation is on conceptual database modeling, viz., models and techniques for defining a user-level, meaning-based specification of data, and user-interfaces that support user-database interactions.

A *conceptual database model* (also called data model or information model) is a generic mechanism for structuring a database to "simulate" a real-world or a hypothetical data-intensive application environment. It consists of a collection of constructs, a collection of fundamental operations, and a set of integrity rules defined on those constructs [Codd 81, Tsichritzis 82]. *Data modeling* is the process of constructing a representation of an application environment using a conceptual database model. A conceptual database model should fulfill the following properties: 1) simplicity: to make it easy to learn and use, 2) data-descriptiveness: to be capable of describing and modeling all

kinds of data, 3) semantic-expressiveness: to give a complete, user-understandable, and accurate meaning of data, and 4) evolvability: to make it easy to maintain databases.

A *conceptual schema* (schema, meta-data) is an instance of a conceptual database model, describing the structure and properties of a particular database for a specific application environment. Conceptual schemas facilitate data independence through isolation of logical external views of databases from their physical implementations. That is, changes to external views have no *direct* effect on internal schemas. At the same time, this isolation facilitates optimization of the internal schema for space and time efficiency and allows definition of application-oriented external views. Ideally, a conceptual schema is to be a "complete" description of an application environment.

1.1.2. User Interfaces

Along with the rapid growth in numbers and the scope of database models, user interfaces have grown in both sheer numbers and variety of their features to satisfy database users. *User interfaces* are the means of communication between users and databases and are usually presented as *database languages* (also called *query languages*).

A major trend in the development of user interfaces in the past two decades has been from *procedural* to *descriptive* or *non-procedural* languages. In the procedural database languages, queries are stated in terms of their

detailed performing functions, namely prescriptions to answer queries. Non-procedural languages, however, focus on describing the answers that queries must provide without stating how to obtain them.

A typical user interface contains the two generic parts, *data definition* and *data manipulation*, and hence two classes of languages to perform these functions. Data definition languages are generally used to design and maintain conceptual schemas, while data manipulation languages are used to access, query, and modify the data in databases.

User interfaces accommodate a wide range of database users from novices to experts. A typical user of a database is assumed to have a good knowledge of his application environment. *Novice* users are those who have no expertise in information management or programming in a database language. *Expert* users are those who are experienced in information management and in using some particular database languages. A *user-friendly interface* is the one that does not require programming and/or database expertise from its users and is easy to learn and simple to use. Ideally, a user interface must be user-friendly and at the same time support all users with different level of expertise without performance degradation.

1.2. Target Environment

Traditional business applications with large quantities of formatted data and relatively inflexible and unchanging structures constitute most existing database applications. However, recent studies indicate that these applications contain less than one fifth of the data used by today's individuals and public and private organizations [Brodie 84]. Recently, several special purpose applications are becoming considerably important and this motivates the development of database models to support them.

These applications are created mainly by recent advances in computer technology which have significantly reduced the cost of computing resources over the past several years. Availability of inexpensive computing resources has dramatically increased in the recent past, and will continue possibly at an even higher rate in the future. This increase in accessibility of computers has created a wide range and variety of computer users with new application requirements for database systems. Potential end-users of such applications are unsophisticated, novice database users, mostly from the following classes:

1. scientists, engineers and technical designers who use computers to manage their scientific, engineering, and design data, e.g. statistical data, documentation, programs, and graphics images;
2. users of home computers who wish to manage their household and personal data, e.g. address and telephone directories, and recipe files; and

3. secretaries and managers of corporations who need to manage their office information, e.g. client directories, calendars, and accounting files.

Current research in database modeling is beginning to study these specialized application environments and recognize their specific modeling needs. Several characteristics of these applications are different from and even contradictory to those of the traditional large commercial environments. Features that best describe this class of specialized databases are listed below[1]:

- their structural information (e.g. data-description, data-interrelation, and data-classification) is typically more complex than conventional commercial databases;
- the structure of the stored information changes rather frequently over the lifetime of the database; thus, the structural framework must be highly dynamic;
- unlike commercial databases, the amount of the structural information of the database is large compared to the size of its information content;
- large quantities of data of different modalities (including formatted data, documents, graphs, messages, and programs) must be accommodated; and

[1]The first three observations also appear in [Motro 85].

- the end-user is familiar with his/her application environment, but is not likely to have expertise in databases or programming. Yet, this same end-user often becomes the ultimate designer, retriever, maintainer, and evolver of the database.

Since the structural information is large, complex, and highly dynamic, it must be as conveniently accessible to end-users as databases' information contents.

1.3. Overview of the Approach

Two of the major trends in the database modeling research have been: 1) the evolution from record-oriented to object-oriented modeling [Zaniolo 85], and 2) the shift of emphasis from modeling strictly the statics of application environments to also including their dynamics. However, data and the descriptive information about data (schema or meta-data) have traditionally been modeled and treated differently; this is also true of most recent semantic database models.

When meta-data is stored distinctly from the data itself, different mechanisms are needed to define and manipulate data and meta-data. This dichotomous treatment of data and schema does not cause major problems as long as the schema is not accessed and manipulated frequently by database end-users. This is usually the case for databases whose schemas are relatively small, uncomplicated, and do not change very often. Examples include many of the large commercial applications that are designed and manipulated by database

experts and used by end-users who learn and memorize whatever they need to know about the schemas of their databases, and access the data accordingly.

Similarly, most database models tend to differentiate in the way they view, and treat certain kinds of data. For instance, in many of the so-called object-oriented semantic database models, certain kinds of data are represented as *objects* while certain others are treated as *attributes*, *relationships*, *functions*, *procedures*, etc. Forcing such distinction between certain kinds of data prevents the ability to model, view, and manipulate the same data in more than one way; this is called *semantic relativism* in the literature [McLeod 80].

Database models that support semantic relativism potentially allow the coexistence and evolution of differing views of the same information. For instance a **buying contract** between a buyer and a seller on a property could be interpreted as: 1) an **object** with attributes describing the buyer, the seller and the property, 2) an **attribute** of the buyer, the seller, and the property, or 3) a **relationship** between the buyer, the seller, and the property. Therefore, semantic relativism allows the information to be treated in the way most appropriate for the viewer, in this example, as an object, an attribute, or a relationship. A particular view can be specified by users at any time by stating the information and operational requirements. Furthermore, semantic relativism allows attributes, relationships, or functions to take part in other relationships as any other object can.

Forcing a distinction between views of data aggravates the need for

database design expertise. For example, to decide whether a piece of information is best modeled as an object, an attribute, or a relationship is sometimes a complicated and involved task. It also causes problems for subschema creation and integration, since the information viewed as an object in one subschema may be desired to be viewed as a relationship in another. Semantic relativism supports the design and evolution of schemas and subschemas tailored to specific user needs.

Another consequence of such heterogeneous treatment of information in database models is that they need a larger and more complex set of operations. For instance, in those database models, even the most primitive operations that apply to an object cannot be applied to its properties (usually modeled as attributes), its associations with other objects (usually modeled as relationships) or its transactions (usually modeled as procedures). This in turn, certainly complicates database manipulation for unsophisticated users.

As a step towards addressing the modeling needs of many recent specialized application environments the approach taken in this research is to unify the view and treatment of all kinds of information. Databases are considered collections of inter-related objects, where an object represents any identifiable piece of information, of arbitrary kind and level of abstraction. For example, a person **Mary**, an attribute **has-phone#**, a string of characters **821-4646**, a structural component (meta-data) **EMPLOYEE**, and a procedure **Hiring** are all modeled uniformly as objects in a homogeneous framework.

Therefore, what distinguishes different kinds of objects is not how they are modeled, rather the set of structural and non-structural (data) relationships defined on them. Consequently, the primitive object-definition, object-manipulation, and object-retrieval operations of such a framework provide users with simple tools to define, manipulate and query database information of any kind and at any level of abstraction.

In most object-oriented database models existence of an object requires the existence of a unique non-null value for its *key* attribute(s). For example, if **student-number** is the key for **STUDENT**, then a particular instance of **STUDENT** cannot exist unless it has a unique **student-number**. It is advantageous not to bind the existence of an object to the value of any particular (group of) property(ies) or relationship(s). Thus, properties and relationships defined on an object can change without affecting its existence. For example, a particular student can exist regardless of the value of its "key-fields", and the value of key-fields can change while the student object remains the same. In a 3DIS database an object can exist independent of any related property(ies).

Finally, there are two general principles that have greatly affected the approach taken in the design of the 3DIS. First, database modeling should be made easier for users to understand so that they can help in (and ideally, perform) the design and maintenance of their own databases. Second, the user-database communication should be made simple, so that users can learn how to

build, manipulate, and evolve their databases. We have taken a four-level approach to achieve our goals. The organization of this dissertation reflects these four levels as follows:

1. A simple object-oriented database model is described that subsumes the fundamental principles of semantic database models, specifically those defined in [Abrial 74, Chen 76, Smith 77a, Hammer 78, Codd 79, McLeod 80, King 82]. Three basic abstraction mechanisms, classification/instantiation, aggregation/decomposition, and generalization/specialization are incorporated in the model to organize and define basic relationships among objects. However, the model is extensible for other kinds of abstractions. In fact, it is extended to support the definition of recursive entities and concepts such as sets, lists, trees, etc., as well as a second useful kind of abstraction that defines the generic interrelations among objects and is used in several specialized applications. A set of primitive object-oriented operations called *object specification* operations are defined for the database model that serve both data definition and data manipulation purposes.

2. To organize the object-oriented framework discussed above, we introduce a predefined set of abstraction primitives that build a specific generalization hierarchy. This hierarchy fits on top of the

generalization DAG (Directed Acyclic Graph) of application environments and organizes databases accordingly. The resulting generalization graph provides a convenient extensible framework for access and retrieval of databases structural information at various levels of abstraction. The graph is traversed through its root object; consequently, investigating the structural information of databases does not require any prior knowledge about them.

3. A geometric representation in the form of a 3-dimensional space is described that arranges both structural and non-structural database information mentioned above. The geometric representation organizes the generalization graphs and also captures the classification and aggregation relationships defined among objects in a single simple 3-dimensional framework. The geometric components of the representation space play a meaningful role in representing certain abstractions of the data. The geometric representation provides a suitable framework for information browsing in which movements are in orthogonal directions and have unique meanings relative to their start position in the space. However, moving in each direction also has a specific meaning that is independent of the start position.

4. A simple object-oriented user interface is defined on top of the geometric representation that helps both novice and expert users to traverse the 3-dimensional framework, and facilitates database design, maintenance, and manipulation. This interface is both *browsing-oriented* and *menu-oriented.* It provides a graphical spatial display of database information. A simple set of *navigational* operations is defined in this level that consists of the two categories of *viewing* and *moving* primitives. Viewing operations provide "display windows" to "information neighborhoods" of interest, while moving operations allow information browsing and retrieval. *Query* operations are formulated using the generic format of triples. More complex queries can be formalized by applying a number of *set operations* on these triples. The menu-oriented part of the interface includes a set of simple object-oriented operations that support database editing.

This research does not consider all problems involved in database modeling. Rather, it addresses some specific unexplored problems in this area concerning the development of a database model and a user interface for a certain class of application environments. In particular, formulation and specification of generalized integrity constraints and many implementation-related issues are not addressed here.

Although we do not study modeling of generalized integrity constraints, many kinds of constraints can be included in the 3DIS database insertion, deletion, and modification transactions. Each transaction itself will then include verification and enforcement of its associated constraints. Moreover, a set of semantic constraints that guarantee the basic consistency of database specifications, are inherent in the geometric representation of the 3DIS, e.g. objects are unique within a database.

A number of implementation aspects of the 3DIS model and the ISL user interface are outside the scope of this research. For the purposes of this dissertation, we have developed an experimental ISL prototype that is a browsing-oriented user interface for 3DIS databases. We do not attempt to study the optimal performance of the 3DIS data model and the ISL user interface. Since our prototype is a single user system, we have not dealt with issues such as the synchronization of concurrent access requests.

1.4. Structure of the Dissertation

This chapter was an overview of the problem addressed by this dissertation and our approach. Chapter 2 contains a discussion of related work in database modeling and user interfaces. Chapters 3, 4, and 5 examine in detail the 3DIS database model and its user interface. Specifically, Chapter 3 discusses the first and second levels of our approach; it presents the database model, and the predefined abstractions that provide a top level structure on the

DAG organizations of application environments. Chapter 4 explains the third level of our approach; it describes a geometric representation for the database model described in Chapter 3 and examines its potentials and limitations as a diagrammatic representation for conceptual database models. Chapter 5 contains the forth level of our approach; it describes a simple object-oriented user interface for the 3DIS model. Chapter 6 studies a specific special purpose application environment from the VLSI CAD class of applications. An example VLSI design environment (ADAM[2]) and its 3DIS database [Afsarmanesh 85a] are described in this chapter. Finally, Chapter 7 summerizes the contributions of this dissertation and discusses areas and directions for future research.

[2]Advanced Design AutoMation, developed at the Computer Engineering Department of USC.

Chapter Two

Related Research

Many aspects of this research are influenced by the recent work in database modeling and user interfaces. All database models provide the basic concepts of database structures, database primitive operations, and database integrity rules. However, they differ in the specific details associated with these concepts and the techniques they choose to represent and apply them.

This chapter briefly describes the two major categories of conceptual database models, namely those frequently referred to as the *conventional* and *semantic* database models. Several of the modeling concepts suggested by semantic database models are described. Some of the current work in semantic database modeling that has had a larger impact on the design of the 3DIS and a few of their significant contributions are outlined. A detailed discussion of these database models and an evaluation of their modeling aspects can be found in [Tsichritzis 82, Brodie 84, King 84a, Schrefl 84, Hull 85]. A discussion of how modeling concepts of the 3DIS compare with some of its forerunners can be found in [Afsarmanesh84]. Finally, this chapter concludes with a brief overview of the related research in user interfaces and outlines a few concepts and ideas used in the development of the ISL.

2.1. Conventional Data Models

The conventional database models are usually exemplified by the hierarchical, network, and relational data models. Hierarchical database models represent data as records that are organized as nodes of tree-type hierarchies. Network database models also use the record structure to represent data, but they organize records in a network, using only one-to-many relationships. The relational data model, proposed by Codd in 1970 [Codd 70], has a well-defined set of structuring primitives based on the mathematical notion of a relation, where a relation is a set of n-tuples. Tuples are used to represent many-to-many relationships among data.

The three conventional data models provide general-purpose access and update operations (language facilities) for query and manipulation of their data structures. The hierarchical and network database languages provide record-at-a-time primitives. Users of these database languages must deal with certain implementation-dependent features of their databases. There are a number of database languages defined for the relational data model that provide set-at-a-time operations and are far more non-procedural than the hierarchical and network languages. The relational model supports the definition of integrity rules through relational languages. However, hierarchical and network data models have a rich set of built-in inherent integrity rules that reflect and maintain the structural integrity of their specifications.

Conventional database models are relatively machine-oriented, and mainly

intended for large commercial applications. These models accommodate large quantities of formatted, rigidly-structured data whose structures are very much stable with respect to the lifetime of the database. Examples of today's business applications where conventional database models serve best include banking and inventory databases. Conventional database systems and their underlying techniques provide efficient and flexible mechanisms for data storage, access, and control (namely, integrity, concurrency, recovery, and protection).

Modeling constructs of conventional database models are very much influenced by the record structure of physical database organizations [Kent 79, Hammer 81]. In fact, in many of these data models, the information is strictly modeled by records. Record structures are not suitable to model loosely-formatted information such as documents, graphs, messages, programs, etc., that arise in many application environments. Even as a tool for modeling fixed-sized formatted data, database expertise is needed to design suitable record formats that model all pieces of information in an application [Kent 79].

A major limitation of the record-based organization of modeling constructs is that such constructs may not directly correspond to identifiable entities or concepts in application environments. Therefore, the information that represents an entity is typically spread among many constructs and, in general, users are left responsible for putting those pieces together when they wish to investigate an entity that is part of their application environment. Generally, schemas defined in conventional data models are designed to support

efficient utilization of database resources for certain applications. This often implies that such schemas are difficult to use for certain other applications.

Another shortcoming of conventional database models is their failure to give proper attention to database dynamics. Application environments are evolving systems and therefore a faithful representation of such an environment must also capture changes that occur in its database states through transactions. Furthermore, capturing and modeling database dynamics decreases the maintenance cost throughout the lifetime of a database. Conventional data models lack the necessary concepts to model such evolution of data and schema.

Finally, there is very little guidance available to users of conventional database models. Design, maintenance, and use of conventional databases require high level skill and expertise. Therefore, these database models are not suitable for unsophisticated (novice) users who intend to build, use, and maintain their own databases.

2.2. Semantic Data Models

Semantic database models, a relatively recent trend in database modeling research, have succeeded in expanding data modeling beyond the capabilities of the conventional data models. They provide a rich set of semantically expressive constructs and mechanisms that reflect more of the meaning of both data and its logical structure, and make it easier for users to understand and use databases.

Semantic database models attempt to conceptualize data-intensive applications as they are observed by humans. Both the modeling constructs and the construct manipulators of semantic data models are defined so as to "naturally" correspond to the concepts, entities, and activities of application environments. Semantic databases are much closer to users' view of their application environments than conventional databases. A semantic schema is much easier to modify than a conventional one, as the nature of application environment activities evolve in time. The conceptual organization of data defined in semantic database models greatly facilitates database design, access, and manipulation. Furthermore, semantic databases are based on the more-generalized, higher-level views of application environments that allow data to be related and interpreted in various ways. This provides much more semantic relativism (described in section 1.3) than what conventional data models provide.

Semantic database models are applicable to large commercial applications as well as several other application environments where conventional database models are not satisfactory. Specialized application environments such as office information management systems [Gibbs 83], personal databases [Cattell 83, Lyngbaek 84], and databases to support engineering design [Katz 82, McLeod 83] fall into this category.

Many of the concepts used in semantic database models are derived from research in two major areas of computer science: data abstraction ideas in

programming languages, and knowledge representation concepts in artificial intelligence. In general, research in the three areas of artificial intelligence, programming languages and database modeling share many common goals. However, there are major differences between the scope of the problem, potential users, and the typical applications in the three areas, leading to different emphasis of research in each case. High-level programming language research is far less concerned with the detailed modeling of the static properties of application concepts than database modeling and artificial intelligence research are, while it is more concerned with efficiently simulating the application activities and providing a proper programming environment for its users. Database modeling research is far more concerned with the structuring of data towards better utilization of the organized information (e.g. providing a good user-oriented environment for design, query, and modification of the data) than artificial intelligence research is.

The ideas incorporated from programming languages into database modeling include: the type concept, data and procedural abstractions, abstract data types, the class concept, strong typing, and control structures [Brodie 84]. For instance, the concept of abstract data types, as in Simula [Birtwistle 73] and Smalltalk [Goldberg 83], are utilized in database modeling through the notions of database *types*, and database *objects*. In the context of programming languages, certain properties and operations are associated with each abstract data type that are applicable to its instances; this defines the notion of a class.

A similar association exists in database modeling where the operations defined on a given database type define the semantics of the members of that type (database objects).

Research in knowledge representation in the artificial intelligence area has introduced semantic networks to organize information. Early semantic networks were merely collections of nodes connected by directed arcs, where nodes represented objects, and directed arcs represented relationships among them, much as in the conventional network data model [Hendrix 77]. Later, semantic networks [Roussopoulos 75, Wong 77, Mylopoulos 80a] evolved to a point of meaning expressiveness that they now have a great deal in common with many semantic database models developed for database management systems.

Several of the ideas that have proven useful in semantic database modeling are either derived from semantic network research or evolved in both areas about the same time. These ideas include: differentiation between types and instances, the idea of partitioning an information network according to its relevance to different tasks (contexts), generalization and aggregation abstractions which define hierarchies of structures, the definition of the role concept for modeling constructs, and decreasing the sharp distinction (in representation and treatment) between data and the structural information (meta-data) [Tsichritzis 82].

2.2.1. Primitives of Semantic Database Models

Most contemporary semantic database models are defined in terms of their basic data modeling constructs (e.g. objects and inter-object associations) and support certain abstraction primitives to interrelate those constructs and organize the data (e.g. aggregation and generalization abstractions). This section describes a number of basic constructs and abstraction primitives that are common to most semantic database models.

2.2.1.1. Basic Constructs

Many semantic database models provide the following constructs to model data-intensive applications:

1. *objects*
2. *inter-object associations*
3. *object groupings*

Objects (entities) represent the identifiable concepts within an application environment, e.g. the person **John**[1]. Inter-object associations define relations among objects, e.g. John **works-at** USC. Objects that share a common set of descriptions and properties are categorized into groups (called types), e.g. John is an instance (member) of the type **PERSON**[2]. Although these modeling constructs are common to many semantic data models, the specific way in

---

[1]Boldface is used to denote examples

[2]Capitalized names represent types

which they are defined and used in every model is different. The rest of this section describes these three constructs in more detail.

1. Objects are mainly divided into: simple, compound, and behavioral objects, where the first two model the statics and the third captures the dynamics of application environments.

- *Simple* objects are atomic strings of characters, integers, or booleans, for example. They are displayable and serve as the *symbolic identifiers* for simple constants in databases. Examples of simple objects are the strings **1080 Marine Ave.** and **743-2745**, and the integer **539**.
- *Compound* objects describe (non-atomic) entities and concepts of application environments. Examples of compound objects are an employee $\mathbf{E_1}$ and a course **202** in a database, where $\mathbf{E_1}$ and **202** are *symbolic names* (colloquially, logical reference names) of objects. These objects are not displayable, expect in terms of their relationships with simple objects. For example, E_1's **phone number is 743-2745**, E_1's **address is 1080 Marine Ave.**, and E_1's **name is Susan Smith**. If a compound object is related to certain other compound objects then it may be displayed recursively in terms of the simple objects related to those compound objects.
- *Behavioral* objects as defined in [Brodie 81, King 84b] embody

database activities, modeling an object that is executable. Behavioral objects accomplish modeling of data definition, manipulation, and retrieval primitives, e.g. **insert E_1 as a member of the type EMPLOYEE.**

2. Inter-object associations (object properties) are mainly divided into two categories: attributes of objects and relationships among objects.

- A descriptive characteristic of an object (the association between a compound and a simple object) is modeled by *attributes*, e.g. John **has-address** 1080 Marine Ave., where **has-address** is an attribute of John. Most semantic database models define attributes as single-valued mappings on objects.
- A property of an object that relates it to another object (the association between two compound objects) is modeled through *relationships*, e.g. John **is-friend-of** Susan, where **is-friend-of** is a relationship defined on John. Relationships are often generically defined as multi-valued mappings in semantic database models, and represent one-to-one, one-to-many, or many-to-many relationships among objects.

3. Object groupings are generally supported through the structural component of *types*. A type denotes a collection of objects (instances or

members) that have a common description. **PERSON** and **EMPLOYEE** are examples of types in a database, whereas for instance, **name**, **address**, **phone number**, and **manager** are properties common to all members of the type **EMPLOYEE**.

2.2.1.2. Abstraction Primitives

Abstraction mechanisms are fundamental conceptual tools for database organization. Semantic database models utilize certain basic abstractions to organize and inter-relate the modeling constructs described previously. There are three principal abstraction primitives used in semantic database models [Smith 77a, Mylopoulos 80b, McLeod 80, Borgida 84, Abiteboul 84, Schrefl 84], mainly for modeling of commercial application environments:

1. classification/instantiation
2. aggregation/decomposition
3. generalization/specialization

The three abstraction mechanisms discussed here do not represent a "complete" set of abstractions to model all application environments. Nevertheless, they capture several fundamental organizational principles that offer a proper basic design methodology for semantic conceptual modeling.

1. The concept of *classification* refers to classifying objects which share common characteristics into a type. For example, **Susan Smith** and **John Davidson** can be classified into the type

PERSON. The concept of *instantiation* is the opposite of classification and is used to obtain objects (members) classified into a type. In the above example, the type **PERSON** has **Susan Smith** and **John Davidson** as its members (instances). The members of a type are unique objects. An object may be a member of more than one type. properties of that type. Classification/instantiation hierarchies connect every type to all of its members.

2. The concept of *aggregation* refers to treating a collection of objects as constituent of a single compound object. For example, **name**, **address**, and **phone number** can be aggregated to define a type **PERSON**. *Decomposition* is the opposite concept to aggregation and refers to decomposing a type into its constituents. In the above example, the type **PERSON** is decomposed into the **name**, **address**, and **phone number** properties. Generally, a multi-valued association can be defined among a type and each of its properties, although, the association may be confined to single-valued or functional relationships. Aggregation/decomposition hierarchies connect every type to all of its constituent properties.

3. The concept of *generalization* refers to building from one or more

types (sub-types) a more generic type (super-type). For example, the **COLLEAGUE** and **FRIEND** type in a personal database can be generalized to the type **PERSON**. A super-type is defined by excluding the individual differences (in properties and conditions) in the definition of its sub-types. In the above example, for instance, **COLLEAGUE** may have **name**, **address**, and **office** properties, and **FRIEND** can have has **name**, **address**, and **birth-date** properties. **PERSON**, then will have has **name** and **address** as its properties. *Specialization* is the opposite concept to generalization and is used to obtain a specific (sub-)type usually by introducing additional properties to the description of a more general (super-)type. In the example above, for instance, we may state that the type **PERSON** is specialized into the two types **COLLEAGUE** and **FRIEND**. A sub-type contains a subset (possibly all) of the members of its super-type. For example, if **COLLEAGUE** is a sub-type of **PERSON**, all members of **COLLEAGUE** are also members of **PERSON**, but the converse is not generally true. A sub-type inherits all of the definition (properties and conditions) of its super-type, and it may also have some additional definitions. For instance, if the super-type **PERSON** has **name** and **age** properties, the sub-type **COLLEAGUE** has **name** and **age** properties, and in addition it

may also have an **office** property, and a transaction of **Getting-Hired**. Generalization/specialization hierarchies connect every type to its sub-type component(s).

Semantic data models provide various mechanisms for defining a sub-type, and for the organization of their generalization/specialization hierarchies. A common way of defining a sub-type is to provide a predicate on the properties of a (super-)type; this predicate identifies which members of the super-type are members of the sub-type. For example, a sub-type **YOUNG-PERSONS** may be defined for the type **PERSON** via the predicate **age<40**. A sub-type may also be defined by enumerating those members of super-type space, that become members of the sub-type.

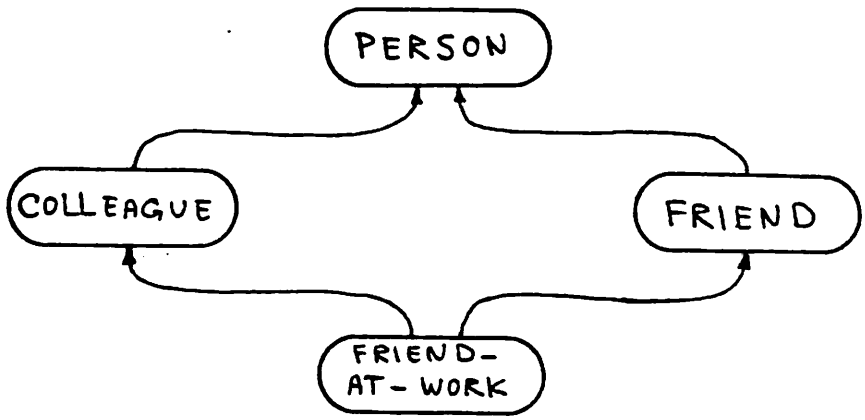


Figure 2-1: a generalization/specialization DAG

Many semantic data models, allow a type to be a specialization (sub-type) of only one other type, in which case the generalization/specialization hierarchies become *trees*. A few semantic data models allow a sub-type to have

more than one super-type. For the latter case the resulting generalization/specialization hierarchy becomes a *directed acyclic graph* (DAG). Generally, in a generalization/specialization hierarchy (e.g. as in Figure 2.1), a member of a super-type (e.g. **PERSON**) can also be a member of *any*, *all*, or *none* of its sub-type components (e.g. either a member of **FRIEND**, or a member of **COLLEAGUE**, or a member of both **FRIEND** and **COLLEAGUE**, or a member of neither one). However, some semantic data models restrict this general definition, for instance, by excluding the *none* case mentioned above. Figure 2.1 represents a generalization/specialization DAG for a personal database, where a member of the type **FRIEND-AT-WORK** is both a member of **FRIEND** and a member of **COLLEAGUE**.

2.2.2. Categories of Semantic Database Models

It is very difficult to characterize and compare the unique features of existing semantic database models [Brodie 84, King 84a, Afsarmanesh 84, Hull 85]. However, we have identified several fundamental trends in the evolution of database modeling towards richer and more expressive modeling constructs in the past two decades. These major directions include:

- from a record orientation to an object orientation
- from a collection of database modeling constructs to semantically organized database structures
- from a strictly static view of a database to also accommodating dynamics

- from separate concepts and mechanisms for handling data and meta-data (schema) to a uniform treatment of all information

Evolution from record-orientedness to object-orientedness modeling concepts resulted in introduction of semantic database models. Later, several semantic database models suggested mechanisms (abstraction primitives) to organize and inter-relate the modeling constructs of databases. Abstraction primitives capture the fundamental structural semantics of application environments and include it into the schema of databases. New modeling concepts were introduced to include the semantics of database dynamics into conceptual database models. A more recent trend, as seen in the 3DIS, is to reduce the sharp distinction between the view and treatment of the structural (meta-data) and non-structural (data) database contents in database modeling.

The rest of this section broadly classifies the recent work on semantic database modeling based on the unique fundamental principles of different data models and their historical order. Several semantic database models are representatives of each of the following categories and exemplify their identified features. Notice that there is some overlap among these categories.

1. **Binary Database Models**

   Binary data models [Abrial 74, Bracchi 76] suggest the concept of binary relations to relate database objects, and use the triplet notation of "object, relationship, object" to represent them.

Databases are modeled as collections of nodes and sets of binary links connecting them. In binary data models, there is a distinction between the representation and treatment of objects and relationships among them. However, data and description of data are viewed and treated similarly. Even though binary data models support the concept of object types, they do not directly support the generalization abstraction and mechanisms for creation of subtypes.

2. Functional Database Models

Functional data models provide the two basic constructs of entities and functional mappings among entities. In a functional database, entities represent database objects and functions represent object relationships and object types. Therefore, functions are allowed to be single-valued, e.g. to represent a person's *social security number*, or multivalued, e.g. to represent a person's *phone numbers*, and may relate an object to a scalar value, e.g. to represent a person's *address*, or an object to another object, e.g. to represent the relationship between a person and his *manager*. Object types are incorporated into functional data models and are represented by functions without arguments, e.g. *PERSON()* represents the type PERSON. A Functional DataBase Model [Kerschberg 75], FQL [Buneman 79, Buneman 84], and DAPLEX [Shipman 81] are

examples of functional data models. Generalization/aggregation relationships are supported by most functional data models. A uniform query facility based on functional composition is provided by the functional data models. This simplifies information retrieval, for example, *address(manager(P_1))* retrieves the address of the manager of the person P_1.

3. Entity/Association Database Models

Entity/association data models contain a wide range of database models that all define a database as a collection of entities and relationships connecting them. Primary data models in this category are limited in supporting the abstraction mechanisms. For example, the Entity/Relationship (E/R) model [Chen 76] does not support the generalization abstraction. Also the semantics defined on the aggregation abstraction restricts the E/R model to provide two levels of *entity* and *relationship* aggregations, where for instance, entities are the aggregations of their properties, and property values may not be entities (they must be scalar values). Later entity/association data models support the fundamental abstraction mechanisms to organize database schemas and many provide meaning-based user interfaces. E/R [Chen 76], ADD: Algebraic data definition [Roussopoulos 77], Hammer and McLeod's SDM: the

semantic data model [Hammer 78, Hammer 81], SAM: a semantic association model [Su 79], Relationship-Entity-Datum [Cattell 83], GALILEO [Albano 83], IFO: a formal semantic database model [Abiteboul 84], PDM: a personal data manager [Lyngbaek 84], and a loosly structured model [Motro 84a] are examples in this category. Many of these data models use a diagrammatic technique to represent database schemas in which several kinds of nodes and arcs correspond to various kinds of entities and inter-relationships in a database.

4. Extensions of the Relational Database Model

This class of semantic data models integrate concepts from the relational data model with the fundamental abstraction mechanisms that meaningfully organize database schemas. Aggregation/generalization model [Smith 77a, Smith 77b], also named SHM: semantic hierarchy model, develops the two kinds of aggregation abstraction and generalization abstraction. SHM suggests a single modeling concept to uniformly express objects in a way similar to how the relational model uses the single concept of relation to represent both entities and relationships. RM/T: relational model Tasmania [Codd 79] incorporates semantic relationships such as generalization and aggregation, and uses special

relations to represent them. For instance, there are three kinds of relations that represent RM/T entities as the aggregations of 1) single-valued attributes, 2) multi-valued attributes, 3) associations among objects. However, incorporation of semantics into the RM/T model sacrifices the benefits of having a uniform representation for all objects in the relational database model. Other work in this category similarly improves the meanings associated with relations and the relational tuples, to capture the semantics of application environments [Wiederhold 79, Stonebraker 84].

5. **Behavioral Database Models**

Behavioral data models are extensions to entity/association models which also include necessary concepts and constructs to model database dynamics. Dynamic properties of application environments are modeled as database transactions by "procedure" constructs. Behavioral models provide a framework that includes facilities and mechanisms for database specification, design, manipulation and evolution. TAXIS [Mylopoulos 80a, Mylopoulos 80b], SHM+ [Brodie 81], and the Event Model [King 84b] are examples of behavioral data models.

2.3. End-User Interfaces

User-database interactions (database languages) mainly consist of *query formulation* and *answer representation* tasks. A major part of the query formulation task is the set of instructions that describe the queries. The instructions must be intuitive to users and easy to remember and use. Answer representation must deal with the issues of representation complexity and the medium used for representation, among others. The complexity of representation relates to the format in which a reply is given to a user and how easy and acceptable it is for the user to follow and understand it. For example, a reply could be in the text format, or it may have a tabular or graphical representation. It is also desirable for users to have flexibility in selecting the medium on which they receive a reply. For example, replies could be printed on paper or displayed on the screen as text, graphics, or even animation.

Study of the application environments, specifically the human factor issues involving the audience for which the user-database interaction tool is targetted, are becoming more important in recent database languages [Lochovsky 81]. These languages try to provide an easy and interesting environment for user-database interactions. Some recent database languages support several versions of a language to address different requirements of application environments, e.g. support database browsing as well as a traditional query language to facilitate user-database communications. However, in general, many tradeoffs are involved in selecting among the desirable characteristics that a database language should provide, including efficiency of their implementation.

The rest of this section describes some major categories of query languages and presents examples of each category. the classification is partially due to Lochovsky and Tsichritzis in [Lochovsky 81], and is mainly based on the general format of these languages and the particular features and characteristics through which they interact with users. There is, of course, some overlap among these categories and a query language may fit into more than one category.

1. **Natural Languages**

   Natural language interfaces allow users to state their queries in a natural language (e.g. English) and the system attempts to translate it into database operations. Replies to requests are usually presented in terms of typed sentences. Examples of such languages include the TORUS [Mylopoulos 75], a Natural Language Interface to Complex Data [Hendrix 78], and RENDEZVOUS [Codd 78]. Users of these languages do not need to learn about language features since no set of instructions or commands are provided. However, users must have experience with the system before they can efficiently use it, because little technical guidance is provided as to how queries are to be formed. Users formulate their queries in terms of a dialogue with the system. As with regular sentences, simple requests are easy to specify, while complex ones are difficult.

Natural languages are the most uniform query languages across different applications and databases. However, many natural language interfaces are tailored to specific application environments, which makes them difficult and rather expensive to use in other environments.

2. **Query-by-example Languages**

Languages in this class allow a user to present a request by giving an example of the reply to that request. The example is constructed according to the format used for replies. Replies are presented in terms of the same templates used to form queries. Query by example languages help users to form their queries by providing the templates to be used for input and the attribute names that fill it. A user need only to indicate which attribute values he/she wishes to investigate and provide the conditions for selecting those values by giving some example values. Formulation of simple queries is quite easy and straightforward. However, more effort is needed for complex queries that involve several templates. Query-By-Example [Zloof 75, Zloof 80] and Form Operations [Luo 81] are examples of these query languages.

3. **Graphics-oriented Languages**

Users of a graphics-oriented query language formulate their requests

in terms of diagrams along with a little input text. Replies are usually presented in tabular forms. Examples of this class of query languages include the CUPID language of the INGRES system [McDonald 75], the GUIDE [Wong 82], the SNAP [Bryce 85], and the ISIS [Goldman 85]. Specific diagrammatic shapes are associated with types of objects and users state their requests by manipulating those shapes. The complex part of these languages is in the task of constructing diagrams to represent queries, even simple ones. It is very interesting to draw pictures that state queries. However, particularly for novice users, it is not always clear how to manipulate shapes to formulate their requests.

4. Menu-oriented Languages

Menu-oriented query languages allow users to specify requests by pointing at the data object or the function that they desire. Replies to requests are presented in the form that objects are stored in the system, e.g. text, tables, graphs, forms, etc. Available data objects and functions are presented to users in menus at all times. Menus can be defined in a hierarchical manner so that choosing a function or a data object may cause additional menus to become available. Functions and data objects are represented by either their names or

an image (icon) that symbolizes them. Formulating simple queries in menu-oriented languages is quite easy. However, it is more difficult to state complex queries that are not hierarchically structured the same way as the menus are. Officetalk system [Ellis 80] and the query language of the Relationship-Entity-Datum data model [Cattell 83] are examples of the menu-oriented query languages.

5. **Multi-feature Languages**

These languages allow users to interact with the system in the form of text, formatted data, voice, graphics, animation, etc. Users have the flexibility of formulating their queries by pointing to icons, choosing from menus, and typing input on the screen. Types of objects are represented by icons, and users can browse through the icons and select the desired ones. Information is structured in a hierarchical manner, so selecting an icon may result the expansion of its definition into either more icons, or the actual data itself. The spatial management of data [Herot 80] is an example of the multi-feature languages.

6. **Browsing-oriented Languages**

Browsing-oriented query languages support a set of navigational

operations that users iteratively employ to traverse database information by locating the appropriate portion of information they wish to investigate. Replies are usually presented in the form that information is stored, e.g. text, tables, graphics, etc. Browsing-oriented languages may support traversing of structural and non-structural information and some provide operations that support both. Database browsing has been recognized to facilitate the access and manipulation of database information for novice users. Several browsing languages based on either the conventional or the semantic database models have been designed. Examples of these query languages include the spatial management of data [Herot 80], the TIMBER [Stonebraker 82], the query language of the Relationship-Entity-Datum data model [Cattell 83], the language designed for browsing in a loosely structured database [Motro 84a], SNAP [Bryce 85], ISIS [Goldman 85], and BAROQUE [Motro 84b]. Most of these languages are also graphics-oriented. They include facilities to view data or database structures graphically, mainly as charts. The complex part in designing graphical displays for database browsing lies in displaying large database structures in a coherent fashion. It is important to avoid the display of confusing crowded networks of nodes and arcs. It is also desirable to take full advantage of the spatial relationships, e.g. localities to reflect some database semantics such as database abstractions.

Chapter Three

A Specification of the 3DIS Model

3DIS is a simple object-oriented database model that addresses several information management requirements of a large class of recent databases described in Section 1.2. The data model is based on a small number of simple constructs and primitive operations to manipulate them. 3DIS databases contain collections of inter-related objects where all information including the data, the descriptions and classifications of data (meta-data), abstractions, operations and constraints are treated uniformly as objects. Within this object-oriented framework, the 3DIS incorporates several predefined fundamental abstraction primitives, and is further extended to support a number of other abstractions that are commonly needed in application environments. The predefined abstractions include a specific generalization hierarchy that connects to the structural graph (DAG) of application environments to organize and simplify access to the database's structural information of databases. A set of primitive operations is defined on objects that allows uniform viewing, insertion, deletion, and modification of information objects in a 3DIS database. This chapter describes the 3DIS information model in terms of its modeling constructs, its predefined and extended abstraction mechanisms and its object manipulation primitives.

3.1. Modeling Constructs

Objects and *mappings* are the two basic modeling constructs in the 3DIS information model, where mappings are a subset of objects. Relationships among objects are modeled by "(domain-object, mapping-object, range-object)" triples. The structure of these triples is very much like the 3-element associative cell in the LEAP language [Feldman 69], and the binary relationships in the Binary data model [Abrial 74].

These triples provide a simple information representation scheme similar to a *relation* in the pure relational model [Codd 70]. A single-valued binary relationship stating: the name of Emp_1 is Smith, is represented by a *simple triple*: **(Emp_1, Has-Name, Smith)**[1]. The three elements of simple triples are atomic. A multi-valued binary relationship stating: the phone numbers of Emp_1 are 743-1234 and 743-5678, may be represented by the two simple triples:

(Emp_1, Has-Phone-No.s, 743-1234) and

(Emp_1, Has-Phone-No.s, 743-5678), or the single *compound triple* of:

(Emp_1, Has-Phone-No.s, {743-1234, 743-5678}). A compound triple contains a set as one of its three elements. Triples with more than one set element are not allowed, since in general, the semantics of the relationships among the elements of the two sets is ambiguous.

---

[1]Inverse relationships, e.g. (Smith, Is-name-of, Emp_1), are always automatically generated by the 3DIS system. More detail is covered under *mapping objects* in Section 3.1.1.

For n>2, n-ary relationships among objects are handled by defining a new object to represent the n-ary relationship itself, and then a set of binary relationships between that new object and all objects involved in the original n-ary relationship. For example, a contract is a relationship that can involve more than two objects, e.g., a buyer **Susan**, a seller **Mary**, a property **green-car**, and a date **3/25/85**. These facts are modeled by introducing a new object **Contract-223**. Then, the buyer, the seller, the property, and the date are related to this contract as follows:

(Contract-223, Has-Buyer, Susan),

(Contract-223, Has-Seller, Mary),

(Contract-223, Has-Property, green-car),

(Contract-223, Has-Date, 3/25/85).

Although some database models permit the direct representation of n-ary relationships, this break down of n-ary relations into their irreducible binary relations has not proven to be a limitation in capturing the semantics of n-ary relationships.

3.1.1. Objects

Every identifiable piece of information in an application environment corresponds to an *object* in its 3DIS database. Consequently, the generic notion of objects represents all modeling concepts described in Section 2.2.1.1. Simple, compound, and behavioral entities in an application environment, attributes of objects and relationships among objects, as well as object groupings and classifications are all modeled as objects. What distinguishes different kinds of objects in a 3DIS database is the set of structural and non-structural (data) relationships defined on them.

In Section 1.3, we discussed the problems of using user-defined key attribute(s) to identify objects. The 3DIS model deals with objects, of all kinds, directly. However, to represent objects, we need designators (i.e. object-identifiers) by which we can refer to them. Each object has a globally unique *object-id* that is generated by the system. Therefore, an object-id uniquely identifies an object within the entire database[2]. Objects may be referred to via their unique object-ids, or via their relationships with other objects. An example where an object is referred to via its relationships with other objects is when we refer to **John** as **Mary's spouse**.

The 3DIS model supports the three kinds of atomic, composite, and type

---

[2]All information comprising an object except its unique *object-id* is modeled by relationships between objects. Therefore, the 3DIS model captures the semantics of objects while retaining the simplicity of its modeling constructs and primitive operations

objects. Notice that this separation of object kinds is important only at object definition time and not for object manipulation, retrieval, and evolution. Object-ids of atomic objects are the (information contents of) atomic objects themselves. For composite and type objects, object-ids are chosen from a different domain than the one for atomic objects.

1. *Atomic* objects represent the symbolic constants in databases. Atomic objects cannot be decomposed into other objects. The contents of atomic objects are uninterpreted by 3DIS databases, in the sense that they are either displayable or executable, without any further interpretation. *Strings of characters, numbers, booleans, text, messages, audio,* and *video* objects, as well as *behavioral* (procedural) objects are examples of atomic objects.
   - Strings of characters represent short character strings, numbers represent integer and real numbers, and booleans represent the boolean true and false values. The contents of these objects are displayable.
   - Text objects and messages represent long character strings, e.g. the content of a book. The contents of text objects and messages are displayable. A text object may be referred to via its object-id or its relationship with other objects, e.g. the **content-of** the **book-named Ideas and Opinions**.

- Audio and video objects represent digitized voice and images. The contents of audio and video objects are displayable (representable) on appropriate devices.
- Behavioral objects (*procedures*) represent the routines that embody database activities, modeling objects that are executable. These objects represent the data definition, manipulation, and retrieval primitives, e.g. the data manipulation procedure that performs **Hiring-an-Employee**. There are three kinds of procedures: *integrity constraint evaluators*, *storage transactions*, and *retrieval transactions*. Integrity constraint evaluators are procedures that must be invoked whenever their related database objects are inserted, deleted, or modified, to verify the consistency of a database. Storage transaction procedures contain routines for frequently performed insertion, removal, and modification of objects. Retrieval transaction procedures contain routines that answer the frequently asked application-specific queries. The contents of these objects are displayable, however, behavioral objects are also executable.

2. *Composite* objects describe (non-atomic) entities and concepts of application environments. The information content of these objects

can be interpreted meaningfully by the 3DIS system through their decomposition into other objects. An example of a composite object is a student $\mathbf{Stud_1}$. The identifier $\mathbf{Stud_1}$ is the system generated object-id that serves as the symbolic name (colloquially, logical reference name) to this object. The composite object $\mathbf{Stud_1}$ may be decomposed into a set of objects that define its characteristics, e.g. its social security number, name, status, phone numbers, advisor, etc. Composite objects are not displayable, except in terms of their relationships with atomic objects; for example, $\mathbf{Stud_1}$'s **Social-Security-No is 999-99-0123**, $\mathbf{Stud_1}$'s **Status is graduate**, etc. If a composite object is related to certain other composite objects, e.g. a student has an advisor, then it may be displayed recursively in terms of the atomic objects related to those composite objects. For example, $\mathbf{Stud_1}$'s advisor's **Name is Susan Smith**.

- *Mapping* objects are a special kind of composite objects. Mappings can in general model arbitrary relations among two objects. They model both the descriptive characteristics of an object, e.g. a person's name via **Has-Name**, and the associations defined among objects, e.g. the association between a person and his manager via **Has-Manager**. They also model both single and multi-valued relationships, where a

multi-valued mapping is defined from a domain element to a set of elements in its range. Every mapping is defined in terms of, and may be decomposed into:

- o a domain type object[3]
- o a range type object
- o an inverse mapping object
- o The minimum number of the values it may return
- o The maximum number of the values it may return

Formally, a mapping M is M :D→**P**(R), where D and R are its domain and range sets and **P**(R) is the set of all subsets (powerset) of R. D and R are defined as type objects, and M is a mapping from the domain D (or a subtype of D) to the range R (or a subtype of R). For every mapping $\mathbf{m_1}$ there is a system generated *inverse mapping* denoted by $\mathbf{inv(m_1)}$[4]. If $\mathbf{m_1}$ is single-valued, then $\mathbf{inv(m_1)}$ is defined from $\mathbf{m_1}$'s range type object to its domain type object. If $\mathbf{m_1}$ is multi-valued, then for every value in its range the $\mathbf{inv(m_1)}$ returns its related domain elements. For example, if **Has-Friends** is

---

[3]The definition of type objects follows the definition of composite objects.

[4]The term "inverse" is used here in a different sense than in mathematics. If $m_1(d_1)=\{r_1,r_2\}$ and $m_1(d_2)=\{r_2\}$, then $inv(m_1)(r_1)=\{d_1\}$, and $inv(m_1)(r_2)=\{d_1,d_2\}$ which is different from the mathematical definition of inverse.

defined from **John** to his friends {**Mary, David, Alice**}, and from **Mary** to {**David, Sue**}, then **inv(Has-Friends)**, e.g. **Is-Friend-of**, is defined from **David** to his friends {**John, Mary**}. Mappings can be constrained on the number of values they return in order to represent certain kinds of mappings, e.g. one-to-one functions. In fact, limits can be imposed on both the minimum and maximum number of values. For example, the mapping **Has-wife** on the type **PERSON** can be defined with the restriction of minimum zero and maximum one return value.

3. *Type* objects specify the descriptive and classification information in a database; a type object is a structural specification of a group of atomic or composite objects. It denotes a collection of database objects, called its *members*, together with the shared common information about these members. A type object is defined in terms of[5]:

   - its members
   - a set of mappings common to its members, called *member-mappings*

[5]More detail is given under *basic mappings* in Section 3.2.

- the fundamental relationships between this type object and other type objects
- a set of operations defined on its members

Therefore, type objects encapsulate the stored information about object sets. A type object can be a *subtype* of other type objects (supertypes). Every member of a subtype is always a member of its supertype, but not vice versa. That is, members of a subtype are always a subset of the members of its supertype. Subtypes are defined by the enumeration of arbitrary members of their supertypes[6], and inherit some of their supertypes' definition, namely the common mappings and operations. Member-mappings of a subtype may not have the same object-ids as member-mappings of its supertype(s). A type object can be the subtype of more than one other type object. Therefore, the subtype/supertype relationships among type objects can be represented by a directed acyclic graph (DAG). Figure 3-1 represents the type **TEACHING-ASSISTANT** as a subtype of both **STUDENT** and **INSTRUCTOR**. Every type object in the subtype/supertype DAG is the root-node of a (possibly empty) subDAG consisting of its subtypes. In every subDAG, the nodes directly connected to its

[6]Enumeration may be accomplished through a behavioral object, i.e. a procedure defined on the supertype.

root-node are called its *immediate* subtypes. A type object can have more than one immediate subtype. In Figure 3.1, the type **PERSON** has two subtypes **STUDENT** and **INSTRUCTOR**. Every member of the type **TEACHING-ASSISTANT** is a member of both types **STUDENT** and **INSTRUCTOR**, while not every member of **STUDENT** (or **INSTRUCTOR**) is necessarily a member of **TEACHING-ASSISTANT**. Also, any member of **STUDENT** or **INSTRUCTOR** is also a member of **PERSON**. Furthermore, if the type **PERSON** has an associated set of member-mappings and operations common to its members, then these common properties are inherited down the hierarchy to **STUDENT**, **INSTRUCTOR**, and **TEACHING-ASSISTANT**. For example, if **PERSON** has the member-mapping **Name**, then all three such types inherit this mapping. If **STUDENT** has a member-mapping **Advisor** and if **INSTRUCTOR** has a member-mapping **Office**, then **TEACHING-ASSISTANT** inherits both **Advisor** and **Office** mappings.

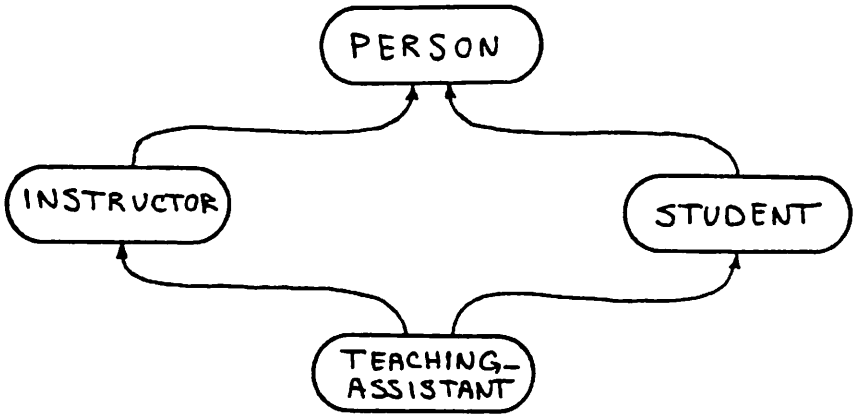


Figure 3-1: A DAG representing the subtype/supertype relationships among type objects

3.1.2. Update Propagation

1. When a type object $\mathbf{T_1}$ is deleted from a database:
   - If there exists a subtype of $\mathbf{T_1}$ that does not have any other immediate supertype, then its immediate supertype becomes the $\mathbf{T_1}$'s immediate supertype(s).
2. If a member $\mathbf{m_1}$ of a type object **T** is to be deleted:
   - $\mathbf{m_1}$ is also deleted from all subtypes of **T**.
   - All relationships defined between $\mathbf{m_1}$ and other objects, through the member-mappings of $\mathbf{T_1}$ or its subtypes, are deleted.
3. If a new member $\mathbf{m_1}$ is to be added to a type object **T**:
   - $\mathbf{m_1}$ will also become a member of all supertypes of **T**.

3.2. Abstraction Mechanisms

Basic associations among objects in 3DIS databases are established through a set of predefined abstraction primitives. These primitives provide the basic organization and interrelation tools appropriate for application environments. Modeling capabilities of 3DIS are also extended by a specific set of abstractions that represent certain concepts and entities common to a number of applications, including the ones described in Section 1.2. The abstraction mechanisms of the 3DIS model are described in this section.

3.2.1. Basic Data/Meta-data Relationships

The fundamental associations among data and meta-data are captured through a set of *basic relationships* defined on objects. These relationships are established via a set of seven specific built-in mappings, called *basic mappings* (and their inverses). Basic mappings support the definition of the three abstraction primitives described in Section 2.2.1.2, namely classification/instantiation, aggregation/decomposition, and generaliza tion/specialization.

- **Classification** is represented by *member/type* mappings that each relates an atomic or a composite object, e.g. Emp_1, to its generic type object(s), e.g. PERSON, EMPLOYEE, etc.
- **Aggregation** is represented by *member-mapping/type* mappings

that each relates a type object, e.g. EMPLOYEE, to a mapping of its members, e.g. Has-Name, Has-Spouse, Has-Address, etc.

- **Generalization** is represented by *subtype/supertype* mappings that each relates a type object, e.g. EMPLOYEE, to a more general type object, e.g. PERSON.

Basic mappings also support the definition of operations common to the members of a type (called procedures), namely their *constraint-evaluators*, *storage-transactions*, and *retrieval-transactions* as described in Section 3.1.1. The basic mappings are defined as follows[7]:

- *subtype/supertype* mappings

  Generalization:

  Has-supertype: type objects → **P**(type objects)

  Specialization:(Has-subtype)

- *member/type* mappings

  Instantiation:

  Has-member: type objects → **P**(atomic/composite objects)

  Classification:(Is-a-member-of)

[7] A parenthesised mapping name that appears below every mapping definition is its inverse mapping. **P**(X) is the set of all subsets (powerset) of X.

- *member-mapping/type* mappings

  Decomposition:

  Has-member-mapping: type objects → **P**(member-mappings[8])

  Aggregation:(Is-a-member-mapping-of)

- *procedure/type* mappings

  Has-constraint-evaluator: type objects → **P**(procedures)

  (Is-a-constraint-evaluator-of)

  Has-storage-transaction: type objects → **P**(procedures)

  (Is-a-storage-transaction-of)

  Has-retrieval-transaction: type objects → **P**(procedures)

  (Is-a-retrieval-transaction-of)

- *basic mappings defined on type objects*

  Has-basic-mappings: type objects → **P**(basic-mappings)

  (Is-a-basic-mapping-of)

---

[8]These mappings include all associations between the (atomic/composite) objects defined by users; see the following example.

The last mapping above relates all basic mappings to every type object. The main reason for including such a mapping is to support uniform representation of all database information. The following example defines the type **STUDENT** in terms of its basic relationships with other database objects:

```
(STUDENT, Has-supertype, PERSON)

(STUDENT, Has-member, {John,Mary,Susan})

(STUDENT, Has-member-mapping, {Has-name,Has-advisor,Has-status})

(STUDENT, Has-constraint-evaluator, average-above-3)

(STUDENT, Has-storage-transaction,
                          {admit,graduate,select-advisor})

(STUDENT, Has-retrieval-transaction,
                         {status-is-graduate,highest-GPA})

(STUDENT, Has-basic-mapping, {Has-supertype,
                              Has-member,
                              Has-member-mapping,
                              Has-constraint-evaluator,
                              Has-storage-transaction,
                              Has-retrieval-transaction,
                              Has-basic-mapping})
```

3.2.2. Underlying Database Structure

A set of predefined abstraction primitives in 3DIS organize the underlying structure of all 3DIS databases. This set includes several predefined type objects that are interrelated via the generalization (subtype/supertype) relationships. *Root* is a predefined type object that is the root-node of this subtype/supertype hierarchy. Figure 3-2 represents the predefined type objects and their subtype/supertype interrelationships, and Figure 3-3 illustrates the subtype/supertype hierarchy itself.

```
(ROOT, Has-subtype, {TYPES, MAPS, EVENTS})

(TYPES, Has-subtypes, {STRINGS, NUMBERS, BOOLEANS})

(MAPS, Has-subtypes, {META-MAPPINGS})

(EVENTS, Has-subtypes, {CONSTRAINT-EVALUATORS,
                STORAGE-TRANSACTIONS, RETRIEVAL-TRANSACTIONS})
```

Figure 3-2: predefined type objects and their subtype/supertype interrelationships

In Figure 3-3 ovals represent type objects, solid arrows show subtype/supertype relationships, circles represent mapping objects, and dashed arrows show the member/type relationships.

The subtype/supertype hierarchy of these predefined type objects connects on top of the structural organization of a 3DIS database that itself may be a DAG. The resulting combined hierarchy provides a convenient and extensible framework for access and retrieval of the structural information of

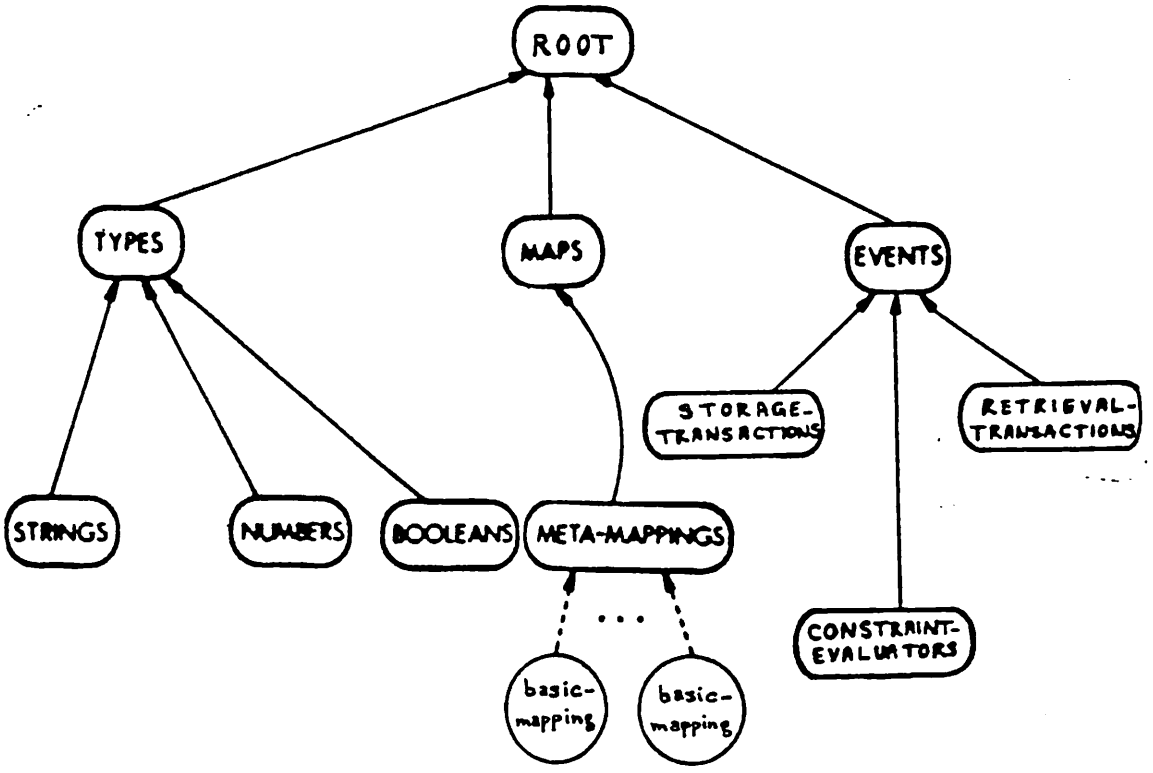


Figure 3-3: The subtype/supertype hierarchy of predefined type objects

databases at various levels of abstraction. Figure 3-4 illustrates an example of a combined subtype/supertype hierarchy, and shows the member/type relationships among database objects. In this figure cross-hatched ovals represent user-defined type objects, cross-hatched circles represent user-defined atomic/composite objects, and dashed arrows show the member/type relationships.

The predefined type objects are:

- *ROOT*

  The root of the subtype/supertype hierarchy in all 3DIS database

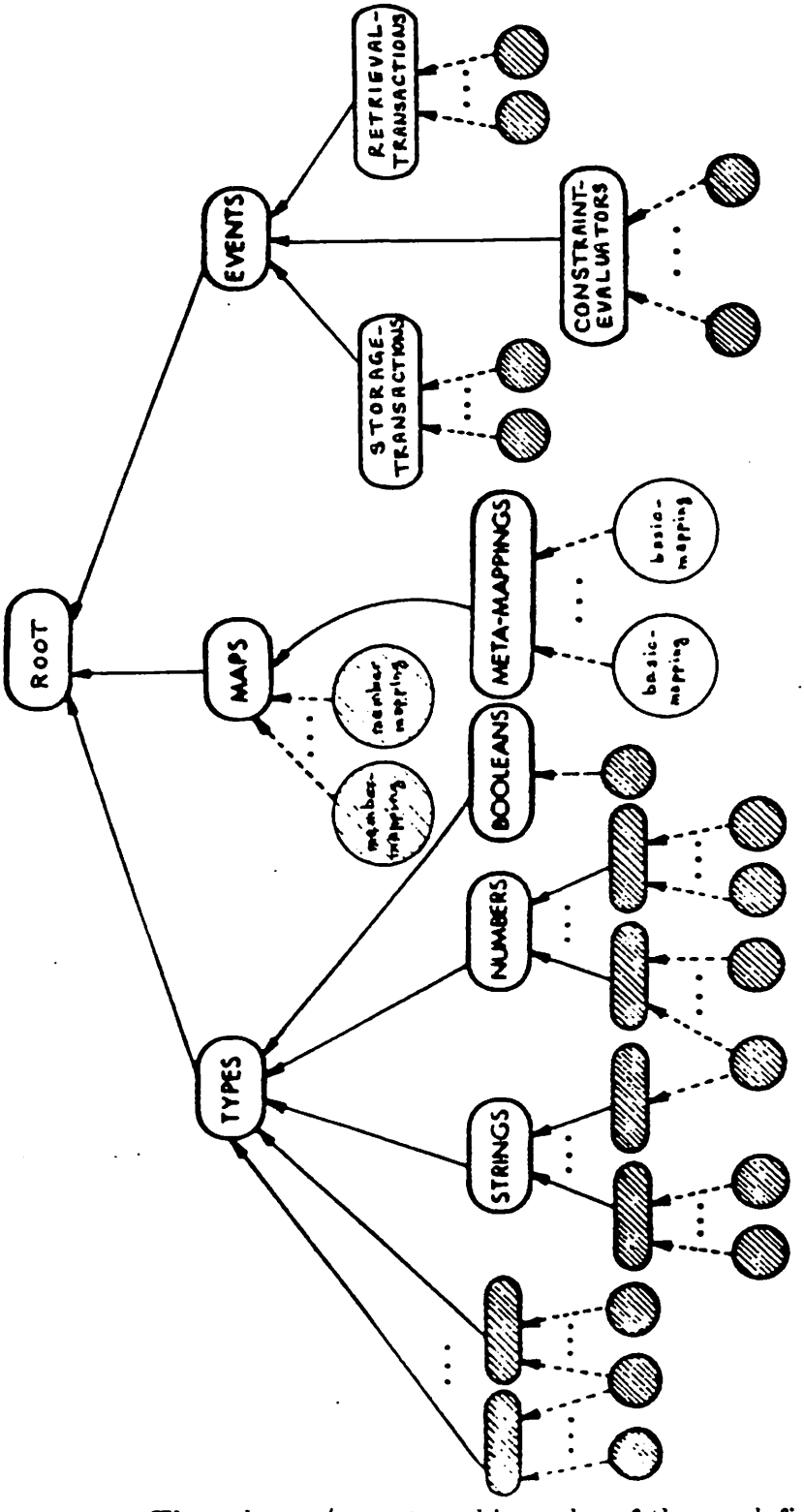


Figure 3-4: The subtype/supertype hierarchy of the predefined type objects connected to a 3DIS database

structures is the type object ROOT. All atomic/composite objects in a database are members of this type object, and all other type objects are its subtypes. ROOT has three immediate subtypes that disjointly partition its members: TYPES, MAPS, and EVENTS. All other type objects in a database are subtypes of exactly one of these three objects.

- *TYPES*

  TYPES is the supertype of several other type objects. *STRINGS*, *BOOLEANS*, and *NUMBERS* are immediate subtypes of TYPES. All numbers in a database are members of NUMBERS. Similarly, all strings and booleans are members of STRINGS and BOOLEANS. STRINGS, BOOLEANS, and NUMBERS may in turn have *user-defined* subtypes[9]. User-defined type objects represent atomic/composite entities and concepts of an application that are grouped by users. User-defined types may be defined as immediate subtypes of TYPES, and may have subtypes of their own.

- *MAPS*

  There is only one subtype for MAPS called META-MAPPINGS.

[9]The type BOOLEANS may only have true or false as its members. Therefore, there are no meaningful user-defined subtypes of BOOLEANS.

Basic-mappings that define type objects are members of META-MAPPINGS. Member-mappings, that define the associations between atomic/composite database objects, are members of MAPS.

- *EVENTS*

  Members of EVENTS are routines representing the dynamics of application environments. Transaction requests issued by users and procedures maintaining the integrity and security of databases are all members of EVENTS. EVENTS has three subtypes: CONSTRAINT-EVALUATORS, STORAGE-TRANSACTIONS, and RETRIEVAL-TRANSACTIONS. Integrity constraint evaluators are routines that verify consistency of databases when they are modified, and are members of CONSTRAINT-EVALUATORS. Storage and retrieval transactions are defined as members of STORAGE-TRANSACTIONS and RETRIEVAL-TRANSACTIONS, respectively. These transactions are procedures that perform users' frequently desired insertion/deletion and manipulation operations on databases. Generally, transactions are procedural elements defined on type objects and describe common operations on their members.

One consequence of organizing the structural information of 3DIS databases via these predefined facilities is that the resulting framework can be

traversed through its root (ROOT). Therefore, investigating the structural information of a database (at any level of abstraction) does not require any preknowledge about its organization. In fact, it is enough for users to know and access the ROOT and then follow the subtype/supertype relationships to access its subtypes. This can lead to the investigation of the entire framework of a database and reach all of its types and their subtypes.

3.2.3. Application Information Management Requirements

The 3DIS information model is extended to include two other abstraction mechanisms that are commonly needed in several application environments, e.g. in engineering design. In this section we describe two abstraction primitives, *recursion* and *generic interrelation*, and show through examples why they are necessary. The use of these two abstraction mechanisms is further discussed in Chapter 6 where we describe a 3DIS database for a specific VLSI design environment [Afsarmanesh 85b].

3.2.3.1. Recursion Abstraction

Recursion abstraction is provided to model recursively defined entities and concepts of application environments. Examples include sets, lists, binary trees, and VLSI design components. Recursion abstraction consists of four specific type objects and the subtype/supertype relationships defined among them. The general format of this abstraction is shown in Figure 3-5. Boxes represent type

objects, arrows represent subtype/supertype relationships, undirected lines that come out of a box lead to mappings that describe members of the type, and what follows a colon is the type of the range element of a mapping.

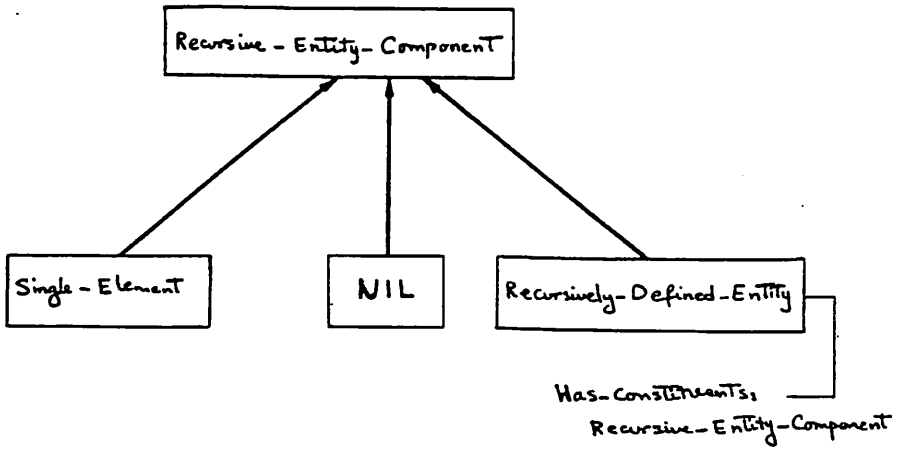


Figure 3-5: The general format of recursion abstraction

The three subtypes of **NIL**, **Single-Element**, and **Recursively-Defined-Entity** disjointly partition the type **Recursive-Entity-Component**. Therefore, any member of **Recursive-Entity-Component** must further be defined as a member of one of these three subtypes. Members of **NIL** and **Single-Element** subtypes "end" the recursive definition of entities, while members of **Recursively-Defined-Entity** are further defined in terms of their constituents that are themselves of the type **Recursive-Entity-Component**. The type **NIL** has exactly one member, **nil**. Figures 3-6, 3-7, and 3-8 respectively show sets, lists, and binary trees defined through recursive abstraction. An example of each of these kinds is given in Figures 3-9, 3-10, and 3-11.

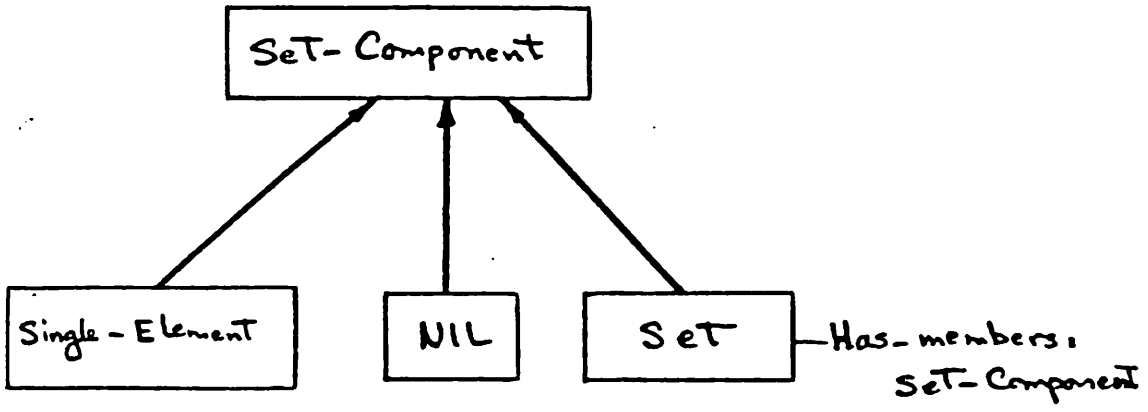


Figure 3-6: Recursive definition of sets

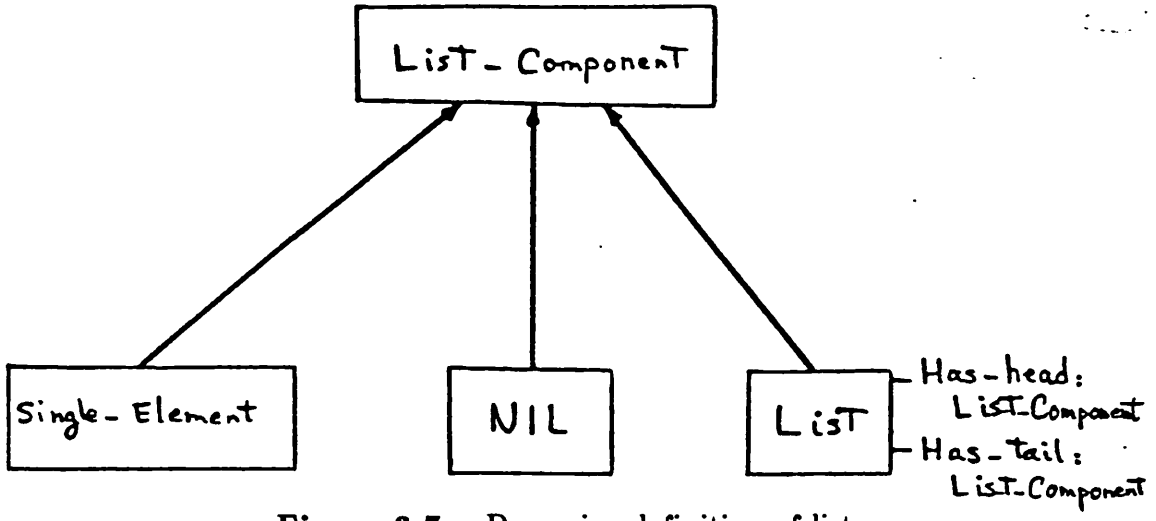


Figure 3-7: Recursive definition of lists

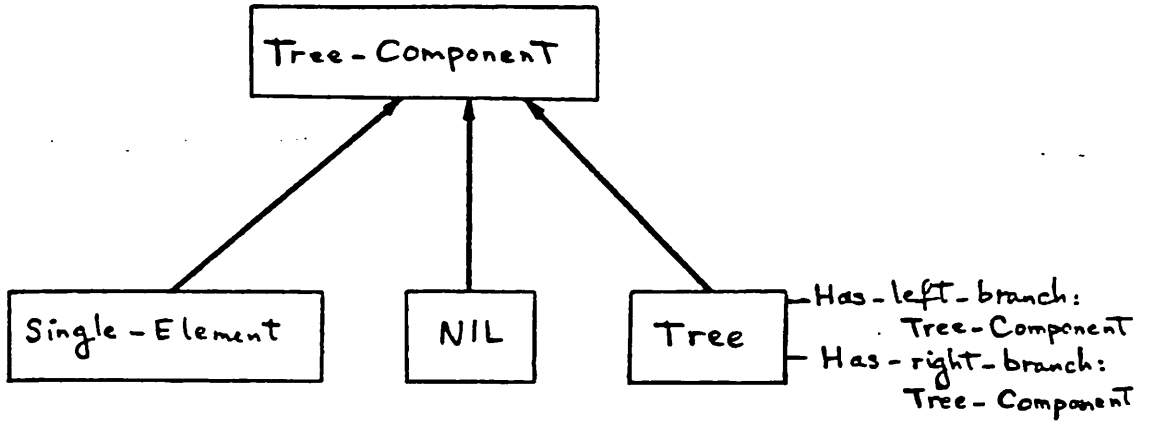


Figure 3-8: Recursive definition of binary-trees

For a set S = {a, {b,c}, {}} :

```
{a, {b,c}, {}}    is a member of          Set
   a                is a member of          Set-Component
   {b,c}            is a member of          Set-component
   {}               is a member of          Set-Component
--------------
   a                is a member of          Single-Element
--------------
   {b,c}            is a member of          Set
      b               is a member of          Set-Component
      c               is a member of          Set-Component
--------------
      b               is a member of          Single-Element
--------------
      c               is a member of          Single-Element
--------------
   {}               is a member of          Set
     Λ               is a member of          Set-Component
--------------
      Λ              is a member of          NIL
```

Figure 3-9: An example set definition

3.2.3.2. Generic Interrelation Abstraction

Generic interrelation abstraction is provided to model those relationships between type objects and composite objects whose semantics cannot be captured by the classical classification/instantiation (member/type) abstraction primitives. Because in 3DIS data and meta-data are treated uniformly as objects, relationships among them need not be restricted to classification/instantiation. A general relationship can be defined between an object and a type as an instance of the generic interrelation abstraction[10].

[10]Other database models that use functions to define relationships among entities, e.g. functional data models, can in principle support generic interrelation abstraction.

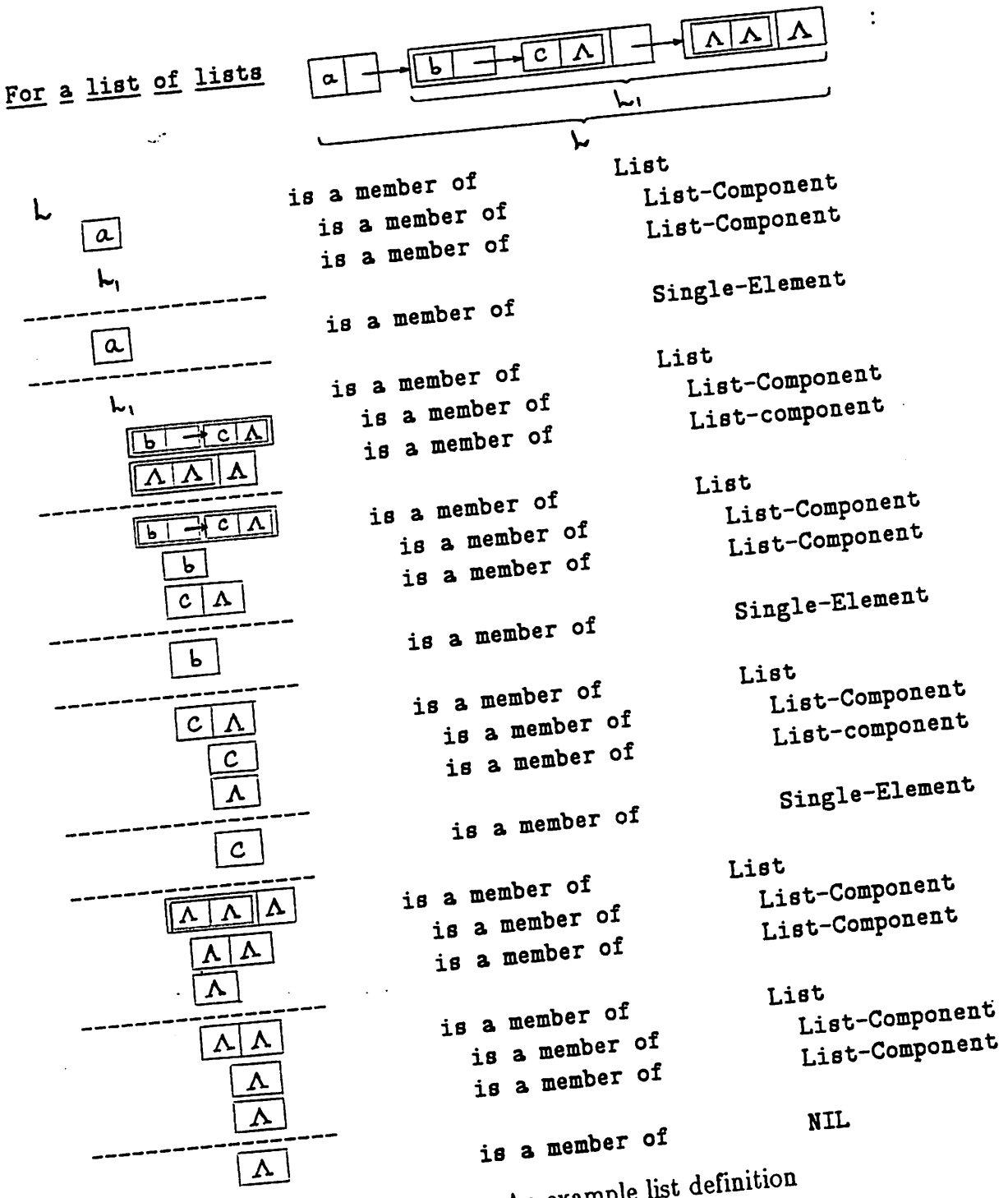
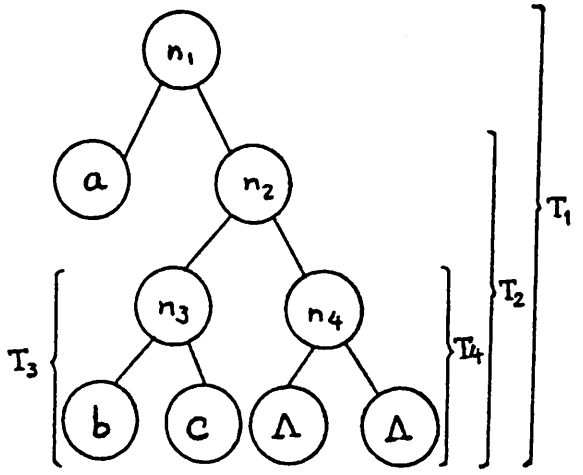


Figure 3-10: An example list definition

Figure 3-12 shows the generalization hierarchy of an example natural language processing application that requires the use of this abstraction.

For a binary-tree T_1 :



T_1 is a member of Tree
- ⓐ is a member of Tree-Component
- T_2 is a member of Tree-Component

ⓐ is a member of Single-Element

T_2 is a member of Tree
- T_3 is a member of Tree-Component
- T_4 is a member of Tree-Component

T_3 is a member of Tree
- ⓑ is a member of Tree-Component
- ⓒ is a member of Tree-Component

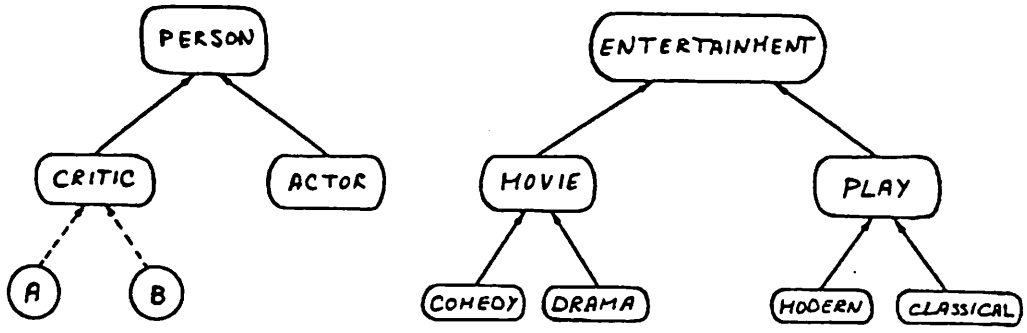
ⓑ is a member of Single-Element
ⓒ is a member of Single-Element

T_4 is a member of Tree
- Ⓐ is a member of Tree-Component
- Ⓐ is a member of Tree-Component

Ⓐ is a member of NIL

Figure 3-11: An example binary-tree definition

In the example of Figure 3-12, suppose that a user wants to represent the



FACTS: A specializes in MOVIEs.
B specializes in CLASSICAL PLAYs.

Figure 3-12: Generalization hierarchy of a natural languages application

fact that: **A specializes in MOVIEs** (or **B specializes in CLASSICAL PLAYs**). A first solution, which is nevertheless the only solution in most other semantic data models, is to use the available fundamental abstraction primitives. First, use the generalization/specialization (subtype/supertype) primitive and define a subtype MOVIE-CRITIC for the type CRITIC (as well as a PLAY-CRITIC, MOVIE-ACTOR, etc.). Second, use the classification/instantiation (member/type) primitive to relate **A** to the type **MOVIE-CRITIC** (and similarly for B). This solution is shown in Figure 3-13. Not only this solution is tedious because of its numerous subtypes, but also semantically less powerful. In Figure 3-14, since **A** is not connected to the type **MOVIE**, it is not related to the semantics of **MOVIE** either. For example, we cannot investigate database objects such as **Gandhi**, **Amadeus**, etc. that are members of **DRAMA MOVIE** and **A** can be a critic of. A better solution is to define the mapping **Specializes-in** between the composite object **A** and the

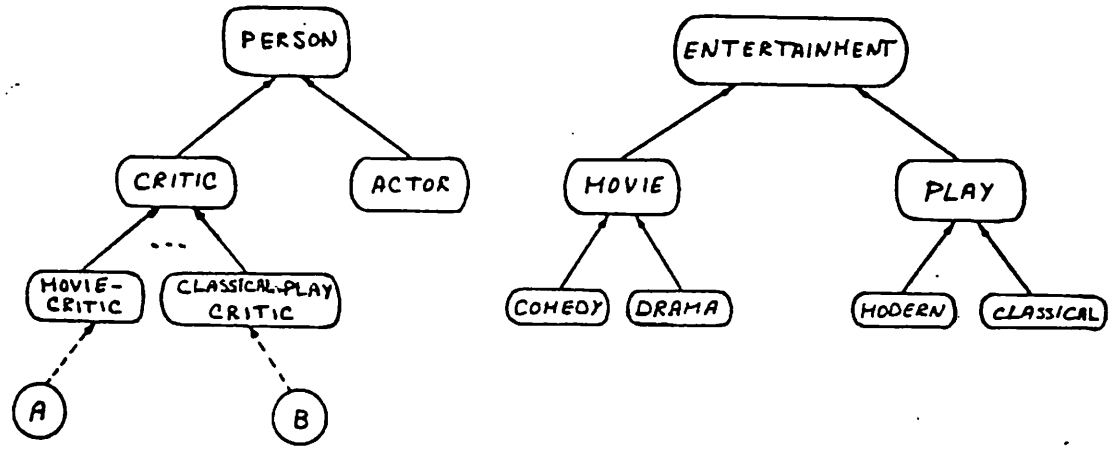


Figure 3-13: The first solution

type object **MOVIE** (and similarly for B), using the generic interrelation abstraction. This solution is shown in figure 3-14.

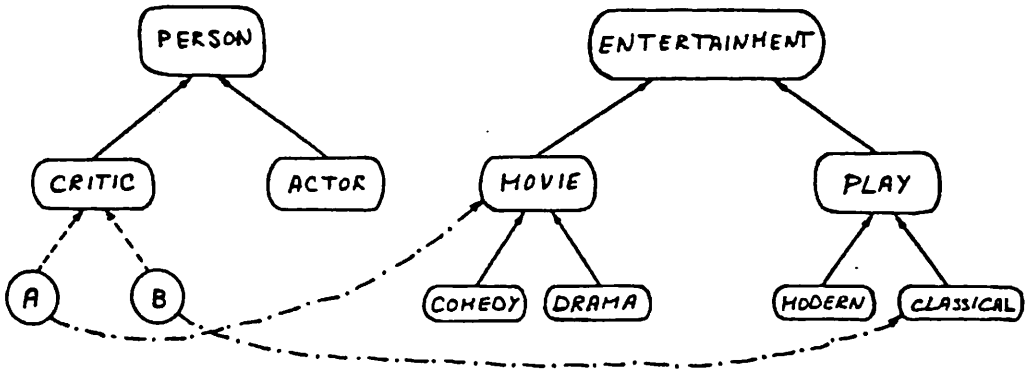


Figure 3-14: The second solution

3.3. Object Specification Operations

A small set of simple but functionally powerful object-oriented primitives, also called the *specification* operations, is defined for the 3DIS model that supports the basic data definition and data manipulation operations. These primitives are assumed to be embedded within a host programming language. The host language supports the data types of objects and sets of objects, the usual set operations, a looping construct to iterate on elements of sets, and a counter for elements of sets.

These operations allow users to add new objects that may be of kind atomic, composite, or type, to remove existing objects from a database, to create and destroy relationships among existing database objects, to retrieve objects and relationships among objects, to invoke behavioral objects, and to display objects on appropriate devices. Once again notice that the separation in object kinds is only important in defining and adding the objects and not for their manipulation and retrieval. These primitives are all defined either on objects or on simple triples that define relationships among objects. However, if a primitive that is defined on triples is applied to a compound triple, the operation first changes the compound triple into a set of simple triples and then iterates on them. Examples for each operation assume that previous examples are executed.

3.3.1. CREATE : object-id

The CREATE operation generates a new composite or type object, adds it to the database, and returns its system generated object-id. The object-id generated for a composite or type object is a globally unique identifier for that object. This identifier is selected from a different domain than the one for atomic objects. In the following examples PERSON, STUDENT, FACULTY, Mary, Has-name, Has-phone#, Has address, and Has-advisor are all variables of the programming environment with type *object-identifiers*:

```
PERSON ⇐ CREATE

STUDENT ⇐ CREATE

FACULTY ⇐ CREATE

Mary ⇐ CREATE

Has-name ⇐ CREATE

Has-phone# ⇐ CREATE

Has-address ⇐ CREATE

Has-advisor ⇐ CREATE
```

The above operations create eight new objects and assign their system generated object-ids to their corresponding variable names. The relationships that will be later defined on these objects (through the RELATE operation) determine the actual semantics of each object. For instance, the fact that PERSON and STUDENT are type objects, John is a member of both types

PERSON and STUDENT, and Has-name is a mapping object, will all be determined when these objects are further defined in terms of their relationships with other database objects.

3.3.2. DEFINE(o:object) : object-id

The DEFINE operation generates a new atomic object and adds it to the database. The system generated object-id for an atomic object is the object itself. Therefore in general, object-ids generated by this operation are of variable length. For example, for a behavioral object, the object-id is the procedure itself, and for an audio object, the object-id is its digitized representation, etc. Furthermore, since object-ids are globally unique identifiers, atomic objects cannot be defined redundantly and if so the system informs users that the object already exists in the database. For example, the number **1324** or the character string **Mary Smith**, appears at most once in a 3DIS database. In the following examples, addr, message, and admit are all variables of the programming environment with type *object-identifiers*:

```
addr ⇐ DEFINE (1080 Marine Ave.)

message ⇐ DEFINE (to: John
                  from:Mary
                  The group has decided ...)
```

```
admit[11] ⇐ DEFINE (Procedure ADMIT-STUDENT
                    VAR  A, B
                      .
                      .
                      .                    )
```

The above operations create three new atomic objects and return their information content as their object-ids.

3.3.3. RELATE (d:object-id,m:object-id,r:object-id)

The RELATE operation generates a relationship among objects and adds it to the database, where d is the domain-element, m is the mapping-element, and r is the range-element in the relationship. The arguments d, m, and r must have been CREATEd, or DEFINEd before this operation. Any primitive operation that returns an object-id (or a set of object-ids in case of PICK operations) can replace any or all of these three arguments. Following examples define some relationships among database objects:

```
RELATE(PERSON, Has-member-mapping, {Has-name, Has-phone#})

RELATE(admit, Is-a-member-of, STORAGE-TRANSACTIONS)

RELATE(STUDENT,Has-supertype,PERSON)

RELATE(STUDENT, Has-member-mapping, Has-address)

RELATE(STUDENT, Has-storage-transaction, admit)
```

---

[11]For an example of the definition of a behavioral object see **g-students** and **s-advisor** under the EXECUTE operation.

```
RELATE(Has-name, Is-a-member-of, MAPS)

RELATE(Has-name, Has-domain-type, PERSON)

RELATE(Has-name, Has-range-type, NAME)

RELATE(Has-advisor, Is-a-member-of, MAPS)

RELATE(Has-advisor, Has-domain-type, FACULTY)

RELATE(Has-advisor, Has-range-type, STUDENT)

RELATE(Has-advisor, Has-maximum-values, DEFINE(1))

RELATE(Has-advisor, Has-minimum-values, DEFINE(0))

RELATE(Mary, Is-a-member-of, STUDENT)

RELATE(Mary, Has-name, DEFINE(Mary Smith))

RELATE(Mary, Has-address, addr)
```

Notice that the RELATE operation is used to define relationships among all kinds of objects. The first RELATE operation in the above example is applied to a compound triple. As it is described before, this triple is first changed into a set of simple triples and then RELATE iterates on them. The result will be:

```
RELATE(PERSON, Has-member-mapping, Has-name)

RELATE(PERSON, Has-member-mapping, Has-phone#)
```

Also, notice that the atomic object "Mary Smith" is generated inside a RELATE operation. Every mapping specified in a RELATE operation must

have been defined in the database previously in terms of its domain-type and range-type, so that the domain and range elements of the specified relationship can be verified. Inverse mappings and inverse relationships are always automatically defined by the system.

3.3.4. UNRELATE(d:object-id, m:object-id, r:object-id)

The UNRELATE operation destroys the specified relationship from the database. If the specified relationship in an UNRELATE operation does not exist in the database, then this operation has no effect. The operation UNRELATE(Mary, Has-address, 1080 Marine Ave.) simply destroys that relationship and leaves the objects **Mary**, **Has-address**, and **1080 Marine Ave.** intact. Notice that all mappings (e.g. Has-address) are assumed to be multi-valued unless the minimum and maximum number of values they may return are explicitly restricted in their definition. If the minimum and maximum number of values for Has-address are specified as 0 and 1 respectively, then in order to insert a new address for Mary, user must first UNRELATE the previous address and then RELATE the new one.

3.3.5. DELETE(i:object-id)

The DELETE operation simply removes the specified object from the database. If the object that is being deleted participates in relationships with other objects then those relationships will also cease to exist, i.e. they will be UNRELATEd. For example, DELETE(Mary) removes the object Mary from the database, and UNRELATEs the following relationships (and their inverses):

```
UNRELATE(Mary, Is-a-member-of, STUDENT)
UNRELATE(Mary, Has-name, Mary Smith)
UNRELATE(Mary, Has-address, addr)
```

However, the objects referred to by the variables STUDENT and addr, and the atomic object Mary Smith remain in the database, and even the fact that Mary was the only object RELATEd to them does not affect their existence.

3.3.6. DISPLAY(i:object-id, dv:device-id)

The DISPLAY operation, displays objects on the specified device. As mentioned earlier, all atomic objects are displayable. For example, the information contents of strings of characters, numbers, digitized audio or video objects, behavioral objects, etc. may all be displayed on appropriate devices. DISPLAY(addr, printer) prints out the string "1080 Marine Ave.". Composite and type objects are not displayable except in terms of the atomic objects RELATEd to them.

3.3.7. RETRIEVE(d:object-id, m:object-id, r:object-id) : set of simple triples

The RETRIEVE operation queries the database by investigating the relationships defined among database objects. Any combination of the d, m, and r elements in a RETRIEVE operation may be replaced by a question mark "?". For instance, RETRIEVE(?, m_1, r_1) where m_1 and r_1 elements are object-ids, returns the set of all simple triples whose mapping and range elements are m_1 and r_1. For example, RETRIEVE(Mary, Has-name, ?) returns {(Mary, Has-name, Mary Smith)}, and RETRIEVE(Mary, ?, ?) returns {(Mary, Has-name, Mary Smith), (Mary, Has-address, 1080 Marine Ave.)}. For the exceptional case where all three elements are "?", RETRIEVE returns the set of all existing relationships in the database. If no element is replaced by "?", then if the specified relationship exists in the database, the relationship itself is returned as answer, otherwise an empty set is returned.

3.3.8. PICK-D(set of simple triples) : set of object-ids

The PICK-D operation projects simple triples on their D (domain) elements, and returns the corresponding object-ids. For example, if the following relationships are defined in a database:

RELATE(John, Has-phone#, DEFINE(743-1234))

RELATE(Mary, Has-phone#, 743-1234)

Then PICK-D(RETRIEVE(?, Has-phone#, 743-1234)) returns {John, Mary}.

3.3.9. PICK-M(set of simple triples) : set of object-ids

The PICK-M operation is similar to PICK-D except that it projects simple triples on their M (mapping) elements.

3.3.10. PICK-R(set of simple triples) : set of object-ids

The PICK-R operation is similar to PICK-D except that it projects simple triples on their R (range) elements.

3.3.11. EXECUTE(i:object-id [, par_1, ..., par_n:object-id])

The EXECUTE operation invokes the specified behavioral object. The system issues an error message if the specified object is not executable. Parameters may be passed to behavioral objects upon invocation. Following examples define a RETRIEVAL-TRANSACTION and a STORAGE-TRANSACTION procedure on the type STUDENT.

```
g-students ⇐ DEFINE (procedure STATUS-IS-GRADUATE
  var  student : object-identifiers
       begin
            for every student in
                PICK-D(RETRIEVE(?, is-a-member-of, STUDENT)) do
              if student is in
                PICK-D(RETRIEVE(?, Has-status, DEFINE(graduate)))
                 then
                   print(student, "is a graduate-student.")
            end
       end
                )
```

```
s-advisor ⇐ DEFINE (procedure SELECT-ADVISOR (student,
                                    advisor: object-id)
   var  number : integer
     begin
       for every number in
            PICK-R(RETRIEVE(Has-advisor, Has-max-value, ?)) do
        if count(PICK-R(RETRIEVE(student, Has-advisor, ?)))
            >= number
            then print ("Enough advisors are already defined
                              for", student,".")
            else RELATE(student, Has-advisor, advisor)
     end
  end
                    )
```

EXECUTE(g-students) invokes the procedure STATUS-IS-GRADUATE. This procedure prints out the name of all graduate students. EXECUTE(s-advisor, Mary, Susan) invokes the procedure SELECT-ADVISOR passing the two parameters of Mary and Susan. This procedure first checks the maximum number of values that can be returned by the Has-advisor mapping and compares it with the number of advisors already assigned for Mary. If the number exceeds the maximum, then the program issues an error message stating that Mary already has enough advisor(s). Otherwise, it defines Mary's advisor

to be Susan. Notice that the maximum number of values for the Has-advisor mapping was previously defined to be "1", so, Mary cannot have more than one advisor. Since behavioral objects are treated as atomic objects, users are responsible to pass the right number and type of parameters to procedures.

A general note on 3DIS's objects definition and modification is that since this is a gradual (stepwise) process and 3DIS does not require object keys to be defined then at any time two (or more) objects may represent the exact same information. However, This is true of any real case application and usually databases contain incomplete information about many existing objects.

Chapter Four

Geometric Representation Space

The 3DIS geometric representation space is a simple and multi-purpose 3-dimensional (3-D) framework that organizes database objects (both structural and non-structural) and their interrelationships graphically. The geometric representation space is discrete, orthogonal, and is assumed to be located in the positive octant of a 3-D space. The three axes in the space represent the domain (D), the mapping (M), and the range (R) axes. All database objects appear on both D and R axes, but the M-axis holds mapping objects only. Relationships among objects are represented by specific points in this geometric space that are defined via their three coordinates, namely (domain-object, mapping-object, range-object) triples. All database objects are treated uniformly and by convention (without loss of generality) information traversal in 3DIS representation space always starts from the D-axis, continues on the M-axis and ends with the R-axis.

In order to illustrate how the 3DIS geometric representation is used to represent database objects and their interrelationships and how its capabilities are used to model, view, and investigate databases, a simple example is given below. Consider a personal database that contains information about people

and their professional activities. A **paper** in this database might have several authors and a title. For instance, a paper P_1 has the title **An Extensible Object-Oriented Approach to Databases for VLSI/CAD**, and is co-authored by **Afsarmanesh**, **McLeod**, **Knapp**, and **Parker**. In Figure 4-1, **Has-title** and **Has-author** are mappings[1] and P_1, **Afsarmanesh**, **McLeod**, **Knapp**, **Parker**, and **An Extensible Object-Oriented Approach to Databases for VLSI/CAD** are all objects, the latter being an atomic object and the others, composite objects. Notice that the object-id of an atomic object carries its information content.

The **(P_1,Has-title,An Extensible Object-Oriented Approach to Databases for VLSI/CAD)** relationship is represented by point A. Points B_1, B_2, B_3, and B_4 represent **(P_1,Has-author,Afsarmanesh)**, **(P_1,Has-author,McLeod)**, **(P_1,Has-author,Knapp)**, and **(P_1,Has-author,Parker)** relationships respectively[2]. The last four relationships can be more compactly expressed as **Has-author(P_1) ={Afsarmanesh,McLeod,Knapp,Parker}**. In general, for any d and m on the D and M axes respectively,

```
m(d) = { r | there exists a point in space with
           coordinates d, m and r }
```

is a one to many mapping. As a convention, we state that in Figure 4-1 the

---

[1]That are in turn objects.

[2]Notice that for simplicity, in Figure 4-1, most of the objects are represented only on one axis.

(P_1,Has-author,{Afsarmanesh,McLeod,Knapp,Parker}) relationship is represented by points B_1, B_2, B_3, and B_4. Furthermore, (P_1,Has-author,?) denotes the line containing B_1, B_2, B_3, and B_4.

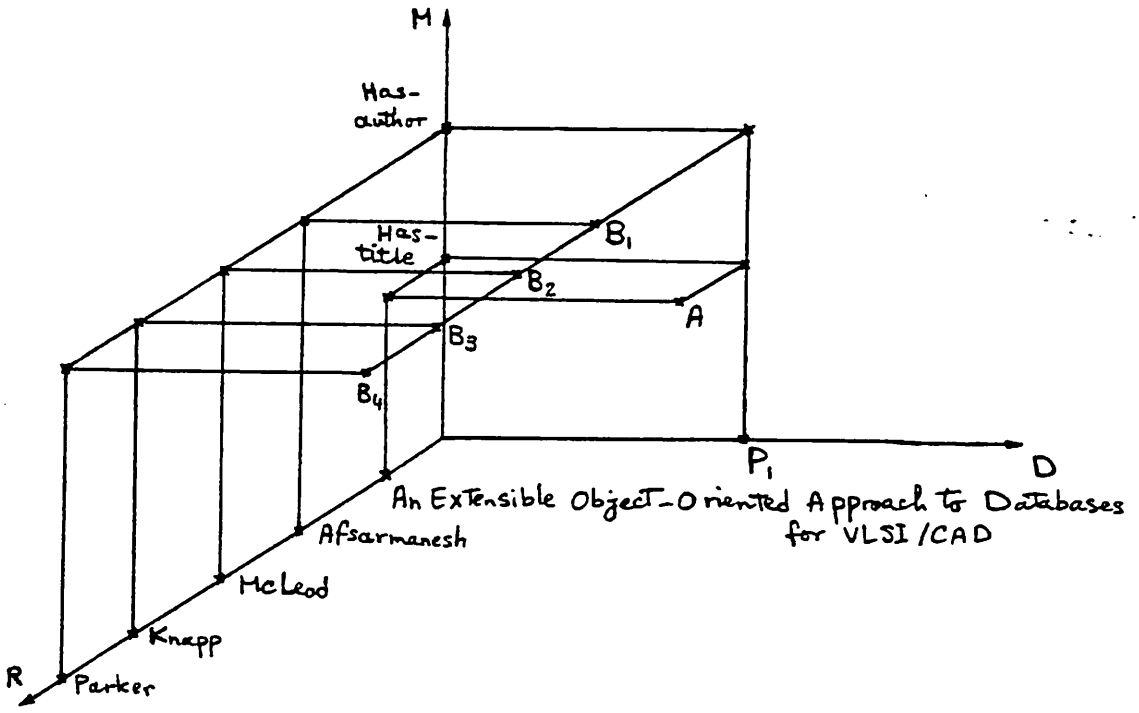


Figure 4-1: Geometric representation of
(P_1,Has-title,An Extensible Object-Oriented
Approach to Databases for VLSI/CAD)
and (P_1,Has-author,{Afsarmanesh,McLeod,Knapp,Parker})

The representation framework explained above is capable of organizing intricate and complex structures and concepts of semantic database models [Afsarmanesh 84]. However, some diagrammatic properties of the 3DIS cubic representation space are very different than those of the graph representations of object-oriented database models. For example, some *implicit* information contained in object-oriented graphs are made *explicit* in

corresponding 3DIS representation diagrams. Consider the example of a person **Person$_1$** has-address **1080 Marine Ave.**. Figure 4-2 represents this relationship in a graph.

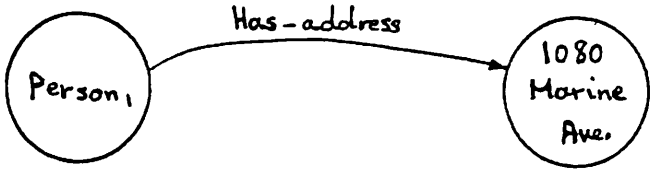


Figure 4-2: The Graph representation of (Person$_1$,Has-address,1080 Marine Ave.)

What is explicit in this graph is the fact that **Person$_1$** and **1080 Marine Ave.** are both objects. The graph also represents that **Has-address** is a mapping, an instance of which is defined from **Person$_1$** to **1080 Marine Ave.**. In a dense graph the mapping **Has-address** can easily become obscured[3]. We argue that since **Has-address** is an arc label --rather than a node-- it can appear in more than one place in the graph. Therefore, to "know all about" **Has-address**, or even whether or not it exists in the graph, requires an exhaustive (logical) search. Thus, the information about **Has-address** is implicit in the graph, and so are the facts that **Person$_1$** is in the domain (where the arc starts), and **1080 Marine Ave.** is in the range (where the arc ends) of **Has-address**. Another "obscure" item of information (implicit) in this graph, is the fact that the two objects **Person$_1$** and **1080 Marine Ave.** are related at all. To ascertain this

[3]Also, note that since **Has-address** is not defined as an object (is not a graph node), it is not possible to define a relationship between **Has-address** and another object; that is, **Has-address** may not appear in the domain or range of any other relationship.

fact requires an exhaustive search of all arcs emanating from both nodes in the graph.

The same relationship (**Person$_1$,Has-address,1080 Marine Ave.**) in a 3DIS diagram is represented by seven points as in Figure 4-3.

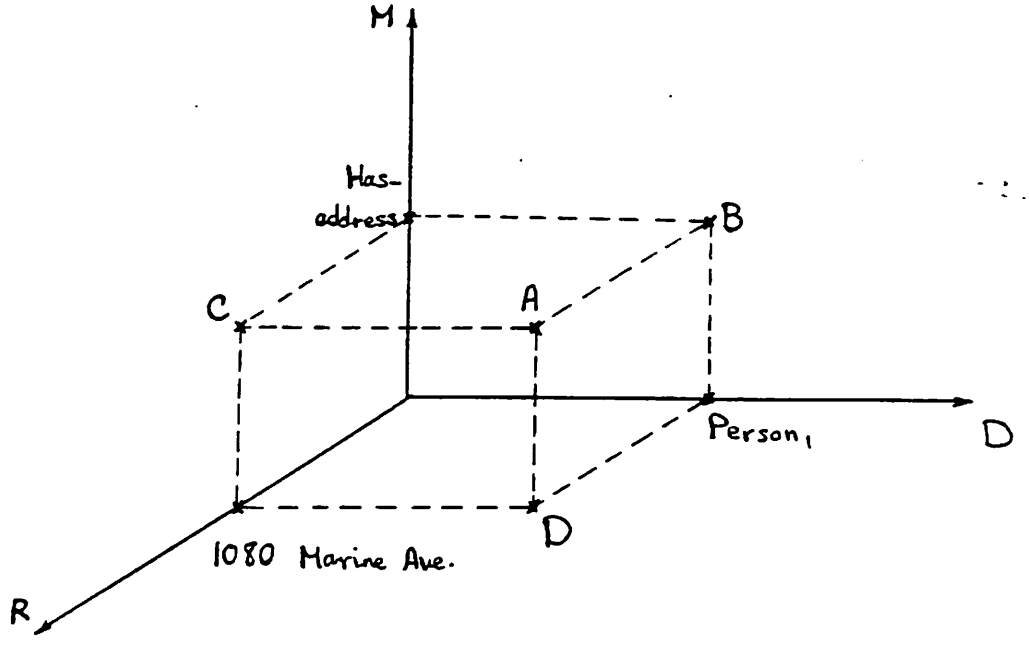


Figure 4-3: The 3DIS geometric representation of (Person$_1$,Has-address,1080 Marine Ave.)

This 3DIS diagram represents both the implicit and explicit information contained in the graph of Figure 4-2 explicitly. More precisely: **Person$_1$**, **1080 Marine Ave.**, and **Has-address** are all defined as objects. Point A represents the relationship (**Person$_1$,Has-address,1080 Marine Ave.**), point B represents that **Person$_1$** is in the domain of **Has-address**, point C represents that **1080 Marine Ave.** is in the range of **Has-address**, and point D represents that objects **Person$_1$** and **1080 Marine Ave.** are (somehow) related.

There is a set of "inherent" constraints associated with the components of the 3DIS representation space that supports the integrity of objects and their interrelationships. The structure defined on the 3DIS representation space itself disallows certain inconsistencies. Consistency rules are enforced through limiting the existence of certain points (relationships) whose existence semantically depend on the existence of other (relationships). For instance, in 3DIS databases, the three coordinates of every "on" point must always be "on" points. Following is the list of integrity constraint rules inherent in the 3DIS geometric representation space when applied to the database in Figure 4-3:

1. Every object, such as **Person$_1$** or **1080 Marine Ave.**, is uniquely represented by a single point on both the D and the R axes.
2. The three coordinates of every on point are on points.
3. If A exists (i.e., it is an on point) then B, C, and D exist.
4. D exists if and only if A exists.

The fact about the uniqueness of database objects is captured by rule (1)[4]. Rule (2) enforces the semantics of relationships defined among objects by stating that every relationship (represented by a point) consists of a domain, a mapping, and a range element, so their corresponding points must exist. Rule (3) expresses another part of the semantics of relationships; for example, if **(Person$_1$, Has-address, 1080 Marine Ave.)** exists in a database, then the

[4]Notice that the uniqueness of object-ids and object-names are also captured by rule (1).

mapping **Has-address** must have been defined on the object **Preson$_1$**. Finally, the fact that if **Person$_1$** and **1080 Marine Ave.** are related, then there must have been a relationship, such as **Has-address**, defined between them is captured by rule (4).

In figure 4-3, the point **Person$_1$** on the D-axis represents the object **Person$_1$** as a domain element, where the point Person$_1$ on the R-axis represents this same object as a range element. Similarly, all objects (which are represented as single nodes in graphs) are "split" into two points on the D and R axes in 3DIS representation space. This double representation of objects may be considered as a drawback for the 3DIS representation space, when compared to the graph representation of object-oriented database models. A consequence of this split representation in 3DIS diagrams is that transitive relationships between objects are not as directly "visible" as they are in their corresponding graphs. For example, *observing* a transitive relationship from A to B to C, something obvious in a graph, requires some dereferencing in a 3DIS diagram. Dereferencing is a consequence of the D-->M-->R convention that we introduced for the traversal of 3DIS representation space.

Geometric components of the 3DIS representation framework such as points, lines, and planes, are units of access and play a meaningful role in encapsulating database information. For example, in Figure 4-1, points located on a vertical line emanating from the object **P$_1$** corresponds to all mappings defined for **P$_1$**. Similarly, a plane passing through an object, perpendicular to

the axis that the object is located on, corresponds to the information about all objects directly related to it. The encapsulation of database information into geometric components provides a natural clustering of related concepts (information) at several levels of abstraction and enables the 3DIS model to reach the object-orientedness goal of its design.

The 3DIS modeling constructs are defined in terms of the geometric components representing them. The geometric components are all defined formally and in a hierarchical manner, namely lines in terms of points, planes in terms of lines, etc.

The geometric representation provides an appropriate foundation for information browsing and serves as an environment for a simple object-oriented graphics-based user interface[5] Database browsing is supported by *views*, that are "display windows" onto the geometric framework, through which users may focus on and investigate an object, or a group of objects ("information neighborhood") encapsulated by geometric components. Browsing within a view is confined to movements in orthogonal directions. Movements have unique meanings relative to their start position in the space. However, moving in each direction has also a specific meaning that is independent of the start position. For example, moving parallel to the R-axis form any point to the next always returns the next range object for the same domain and mapping objects.

---

[5]A specification of the user interface is given in Chapter 5.

The remainder of this chapter presents geometric components of the 3DIS representation framework, provides their formal hierarchical definition, and describes how each component encapsulates a part of database information, in detail.

4.1. Geometric Components

The *primary* kinds of components in the 3DIS geometric representation framework are points, lines, and planes. The *secondary* kinds of components, sublines, subplanes, and subspaces (also called subcomponents), are in turn defined in terms of the primary components.

The 3DIS representation framework is not a solid cube, rather, it is a discrete space with discrete components. For example, Figure 4-4 shows a line component as a collection of discrete point components. A component "exists" in a 3DIS representation space if and only if it represents some information about the application environment. This also means that a "meaning" is associated with every component, defining its semantic role in the database. Points to which a meaning is associated are called "on" points, where points with no corresponding meanings are called "off" points and do not belong to the representation space.

Figure 4-4: A line component

The representation framework is orthogonal, namely all geometric components are either perpendicular or parallel to the three axes.

4.1.1. Point Components

Existing points in a 3-D representation space are the "on" points, and are identified by their three coordinates ordered in a triple, e.g. (d, m, r). The first element of the ordered triple that demonstrates a point **P** corresponds to the D-coordinate, the second element corresponds to the M-coordinate, and the third element corresponds to the R-coordinate of **P**. The origin is a special case point that represents a meaning similar to "no-information". The origin **O** is identified by the triple (-, -, -), where a "-" corresponds to a 0 in analytic geometry and semantically represents that no information is assigned to the coordinate that it is replacing.

Points located on one of the DM, MR, and DR planes, and not on an axis, are denoted by triples that contain one "-" symbol in place of their missing coordinates. Points located on the axes have triples that contain two "-" symbols. For example, in Figure 4-5, point C located on the DR plane is represented by $(d_1,-,r_1)$, where d_1 is the D-coordinate and r_1 is the R-coordinate of C. Similarly, in Figure 4-6, point r_1 located on the R-axis is represented by $(-,-,r_1)$, where r_1 is the R-coordinate of r_1.

Generally, on points represent the existence of relationships among their three coordinates. However, there are two exceptions to this rule. The first

exception is that points located on the D, M, and R axes, such as d_1, m_1, or r_1 in Figure 4-5, with two "-" symbols in their ordered triples, simply represent themselves. The second exception is that, points located on the DM, MR, or DR planes and not on an axes, such as A, B, or C in Figure 4-5, with one "-" symbol in their ordered triples, represent the existence of a two-way relationship between objects on their other coordinates. For instance, point C located on the DR plane, corresponds to the existence of a two-way relationship between d_1 and r_1. All other points in the representation space represent existence of three-way relationships among objects on their coordinates. For example, point E in Figure 4-5 corresponds to the existence of a relationship between d_1, m_1, and r_1[6]. "Off" points in a representation space represent the nonexistence of relationships among the objects on their coordinates.

Every object in a 3DIS database appears as a point on both D and R axes and the two axes are exact copies of each other[7]. Mappings, that are a subset of database objects, also appear on the M-axis.

A relationship represented by an ordered triple (d,m,r) is uniquely represented in the geometric space by a point **P** that corresponds to that

---

[6]By the integrity constraint rules inherent in the geometric representation framework, points on the D, M, and R coordinates of every point are always on points.

[7]Further investigation to determine a "good" ordering technique to arrange objects on the axes so that their *spatial* ordering and distance can convey certain semantic interdependencies was considered outside the scope of this dissertation. However, a dynamic rearrangement of objects by users is supported by the 3DIS user interface and described in Chapter 5 of this dissertation.

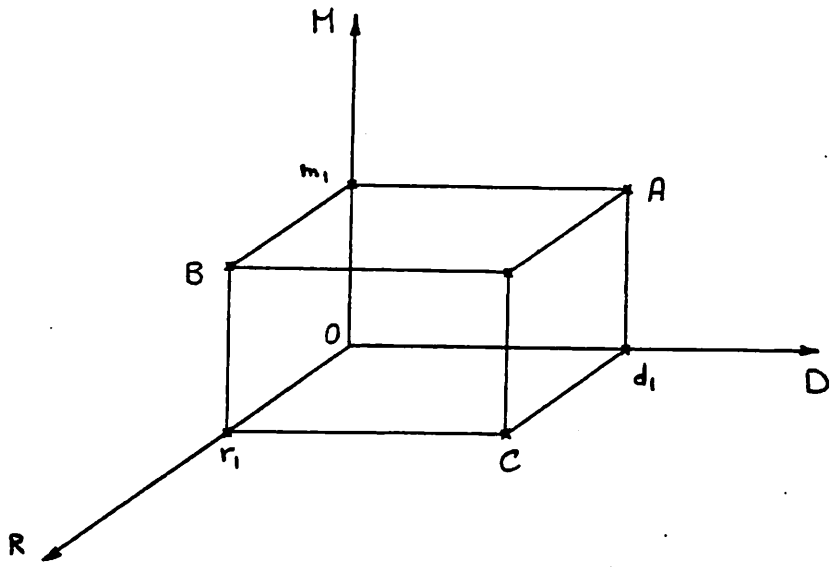


Figure 4-5: A 3DIS geometric representation

ordered triple. For example, in Figure 4-6, the point **F** denoted by (d_1, m_1, r_1) relates the domain element d_1 to the range element r_1, via the mapping m_1, and similarly **G** denoted by (d_1, m_1, r_2) relates d_1, m_1, and r_2.

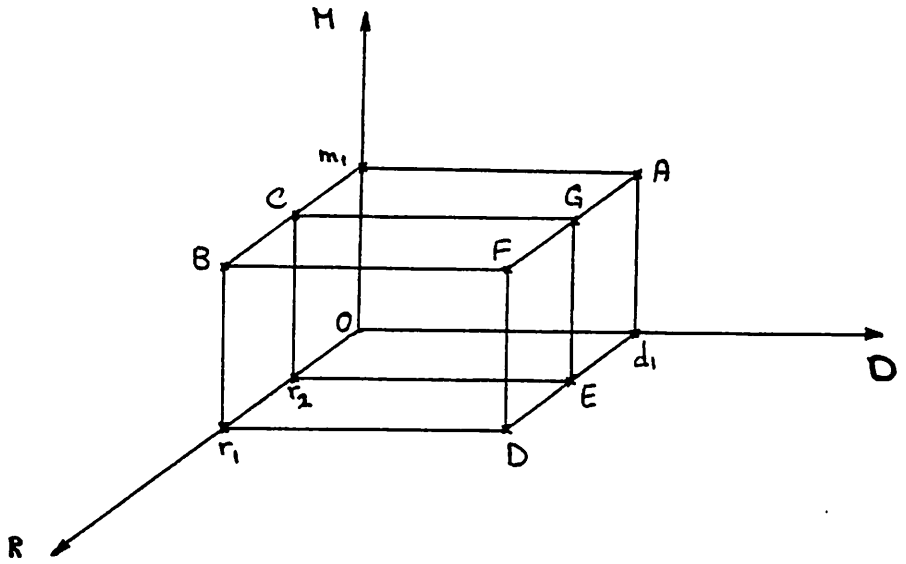


Figure 4-6: A 3DIS geometric representation

4.1.2. Line Components

All line components in 3DIS representation space are discrete and parallel to an axis. Every line emanates from a point located on one of the DM, MR, and DR planes, called its *initial point*[8]. Therefore, lines contain at least one "on" point, viz. their initial point. Two lines in the representation space **intersect** if and only if there is an "on" point on their geometric intersection. All lines are either parallel or perpendicular to the DM, MR, and DR planes. Lines contain sets of points with certain common characteristics. Therefore, every line encapsulates a part of database information[9]. In general, points located on a line introduce values for a "-" coordinate of the line's initial point.

Lines are represented uniquely by ordered triples with a single question

---

[8]The D, M, and R axes are exceptions to this rule.

[9]The details of this information encapsulation are provided later in this section for every category of lines.

mark symbol "?" in place of **one** of their elements, denoting the one-dimensionality of lines. For example, $(d_1,m_1,?)$ corresponds to the line at the intersection of two (geometric) planes normal to the D and M axes at d_1 and m_1, respectively, and contains the set of points whose D and M coordinates are d_1 and m_1, respectively. The "-" symbol as defined previously, is used in the definition of certain lines. For instance, the D, M and R axes are represented by (?,-,-), (-,?,-) and (-,-,?), respectively. In figure 4-6, the line containing F and G is represented by $(d_1,m_1,?)$, where $(d_1,m_1,?)=\{(d_1,m_1,r_1), (d_1,m_1,r_2),(d_1,m_1,-)\}$.

The D, M, and R axes are the three major lines in the 3DIS representation space. They are defined as:

D-axis = $(?,-,-) = \{(d_1,-,-) \mid d_1$ **is an object**$\}$

M-axis = $(-,?,-) = \{(-,m_1,-) \mid m_1$ **is a mapping**$\}$

R-axis = $(-,-,?) = \{(-,-,r_1) \mid r_1$ **is an object**$\}$

There are nine other generic categories of lines in the 3DIS representation space: $(d_1,?,-)$, $(-,?,r_1)$, $(?,m_1,-)$, $(-,m_1,?)$, $(d_1,-,?)$, $(?,-,r_1)$, $(d_1,m_1,?)$, $(?,m_1,r_1)$, and $(d_1,?,r_1)$, each encapsulating certain information about its initial point. In the remainder of this section, we describe each category of lines by defining the encapsulation role it plays in the representation space.

- $(d_1,?,-)$: Locus of all mappings defined on a given domain object

A $(d_1,?,-)$ line, also named L_{MD}[10], emanates from an object d_1 on the D-axis, parallel to the M-axis, and located on the DM plane. This line represents the result of applying the function

has-mappings : objects --> P(mappings)

to object d_1. A L_{MD} line then, is a collection of points that correspond to all mappings on the M-axis, defined on the object d_1. See Figure 4-7[11]. L_{MD} lines are defined as:

$$L_{MD}(d_1)=(d_1,?,-)=\{(d_1,m,-) \mid m \text{ is a point on the M-axis}\}$$

- $(-,?,r_1)$: Locus of all mappings for which a given object is in their range

A $(-,?,r_1)$ line, also named L_{MR}, emanates from an object r_1 on the R-axis, parallel to the M-axis, and located on the MR plane. This line represents the result of applying the function

---

[10]The convention for naming of line components is that for the first six categories of lines two of the three axes (D, M, and R) are used as subscripts to L. The first subscript is the axis to which the line is parallel, and the second is the axis containing the initial point of the line. For example, L_{MD} corresponds to a line parallel to the M-axis that emanates from a point on the D-axis. For $(d_1,m_1,?)$, $(?,m_1,r_1)$, and $(d_1,?,r_1)$ lines, only one subscript is used, denoting the axis to which the line is parallel.

[11]Notice that in Figures 4-7 through 4-15 for every d, m, and r involved, an example data is given inside the parenthesis.

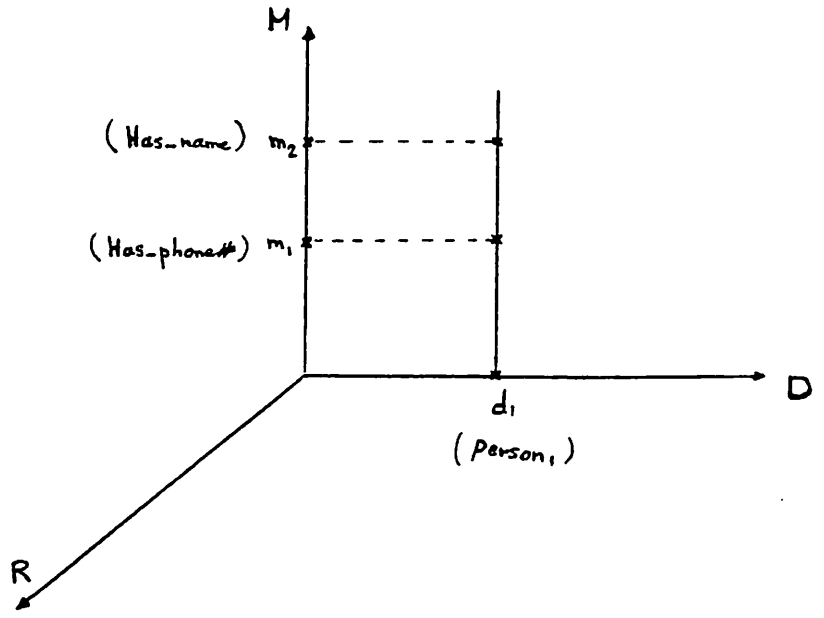


Figure 4-7: $L_{MD}(d_1)$, all mappings defined on domain object d_1

```
is-in-range-of : objects --> P(mappings)
```

to object r_1. A L_{MR} then, is a collection of points that correspond to all mappings on the M-axis, that the object r_1 is in their range (see Figure 4-8). L_{MR} lines are defined as:

$L_{MR}(r_1)=(-,?,r_1)=\{(-,m,r_1)$ |
m **is a point on the M-axis**$\}$

- $(?,m_1,-)$: Locus of all domain elements for a given mapping

A $(?,m_1,-)$ line, also named L_{DM}, emanates from a mapping m_1 on the M-axis, parallel to the D-axis, and located on the DM plane. This line represents the result of applying the function

```
has-domain : mappings --> P(objects)
```

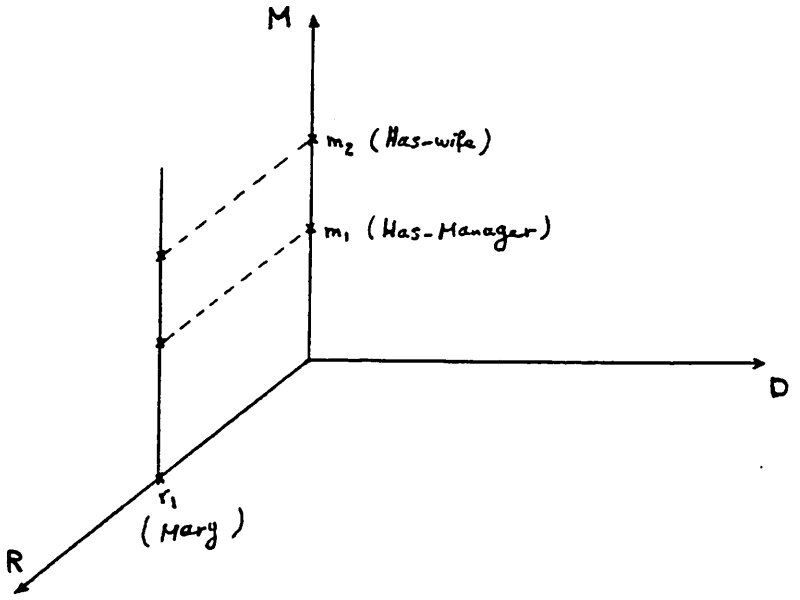


Figure 4-8: $L_{MR}(r_1)$, all mappings which have r_1 in their range

to mapping m_1. A L_{DM} then, is a collection of points that correspond to all objects on the D-axis, that are in the domain of a mapping m_1 (see Figure 4-9). L_{DM} lines are defined as:

$L_{DM}(m_1)=(?,m_1,-)=\{(d,m_1,-) \mid$ *d* **is a point on the D-axis}**

- $(-,m_1,?)$: Locus of all range elements for a given mapping

A $(-,m_1,?)$ line, also named L_{RM}, emanates from a mapping m_1 on the M-axis, parallel to the R-axis, and located on the MR plane. This line represents the result of applying the function

has-range : mappings --> P(objects)

to mapping m_1. A L_{RM} then, is a collection of points that

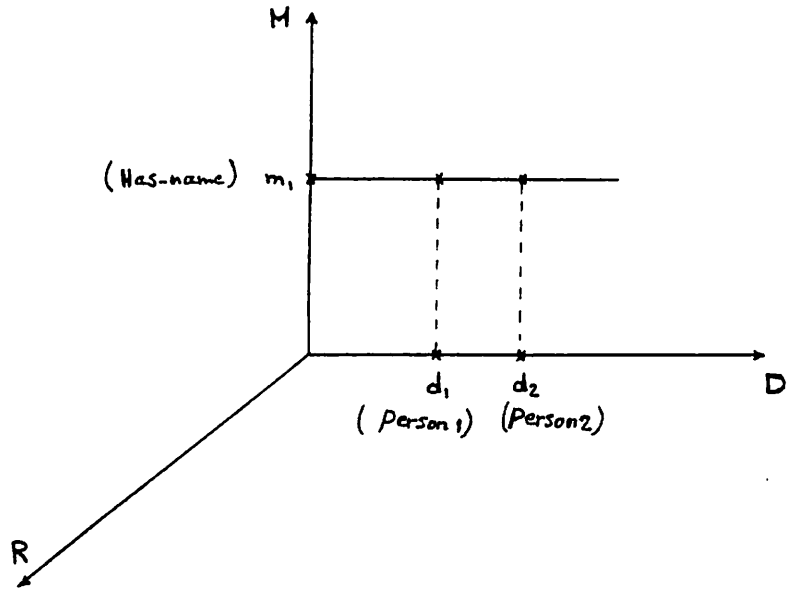


Figure 4-9: $L_{DM}(m_1)$, all domain elements for mapping m_1

correspond to all objects on the R-axis, that are in the range of a mapping m_1 (see Figure 4-10). L_{RM} lines are defined as:

$$L_{RM}(m_1)=(-,m_1,?)=\{(-,m_1,r) \mid r \text{ is a point on the R-axis}\}$$

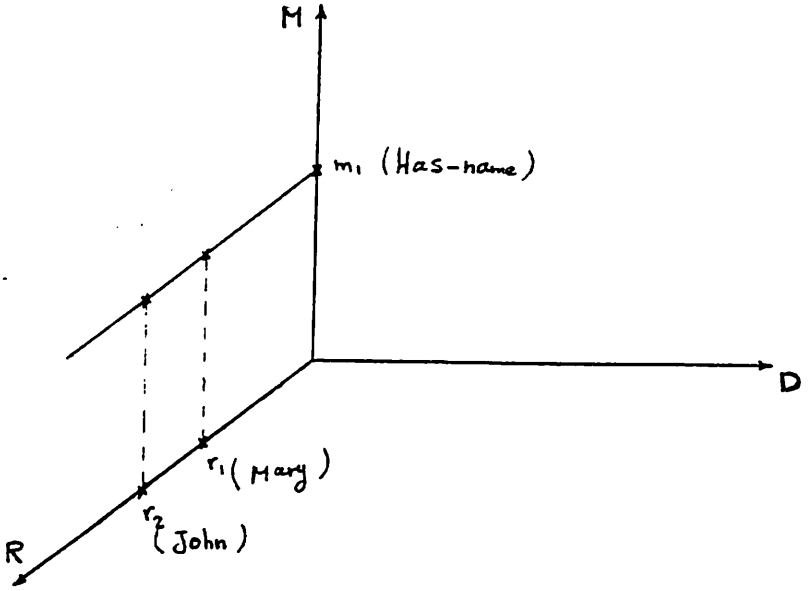


Figure 4-10: $L_{RM}(m_1)$, all range elements for mapping m_1

- $(d_1,-,?)$: Locus of all objects related to a given domain object

A $(d_1,-,?)$ line, also named L_{RD}, emanates from an object d_1 on the D-axis, parallel to the R-axis, and located on the DR plane. This line represents the result of applying the function

```
has-image : objects --> P(objects)
```

to object d_1. A L_{RD} then, is a collection of points that correspond to all objects on the R-axis that are in the image of d_1 under any mapping (see Figure 4-11). L_{RD} lines are defined as:

$L_{RD}(d_1)=(d_1,-,?)=\{(d_1,-,r)$ | *r* is a point on the R-axis$\}$

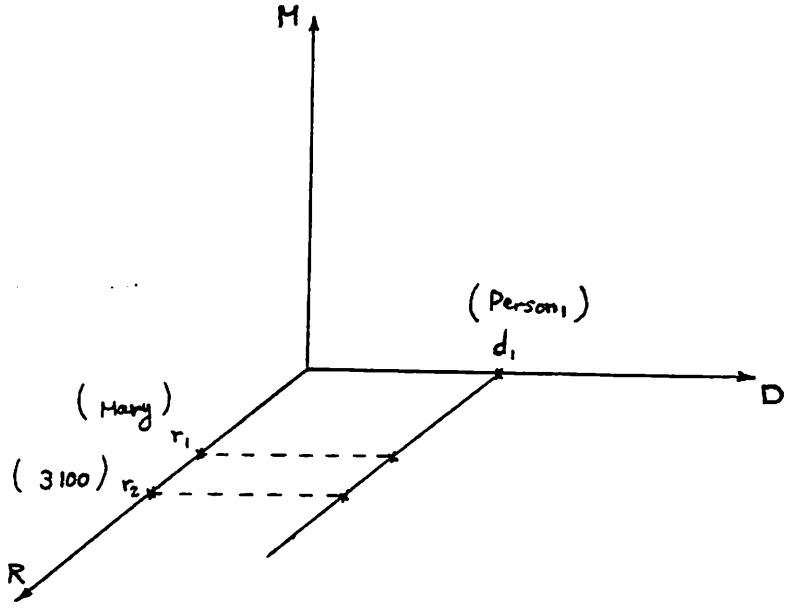


Figure 4-11: $L_{RD}(d_1)$, all objects related to domain object d_1

- $(?,-,r_1)$: Locus of all objects related to a given range object

A $(?,-,r_1)$ line, also named L_{DR}, emanates from an object r_1 on the R-axis, parallel to the D-axis, and located on the DR plane. This line represents the result of applying the function

```
is-in-image : objects --> P(objects)
```

to object r_1. A L_{DR} then is a collection of points that correspond to all objects on the D-axis, that r_1 is in their image under any mapping (see Figure 4-12). L_{DR} lines are defined as:

$$L_{DR}(r_1)=(?,-,r_1)=\{(d,-,r_1) \mid d \text{ is a point on the D-axis}\}$$

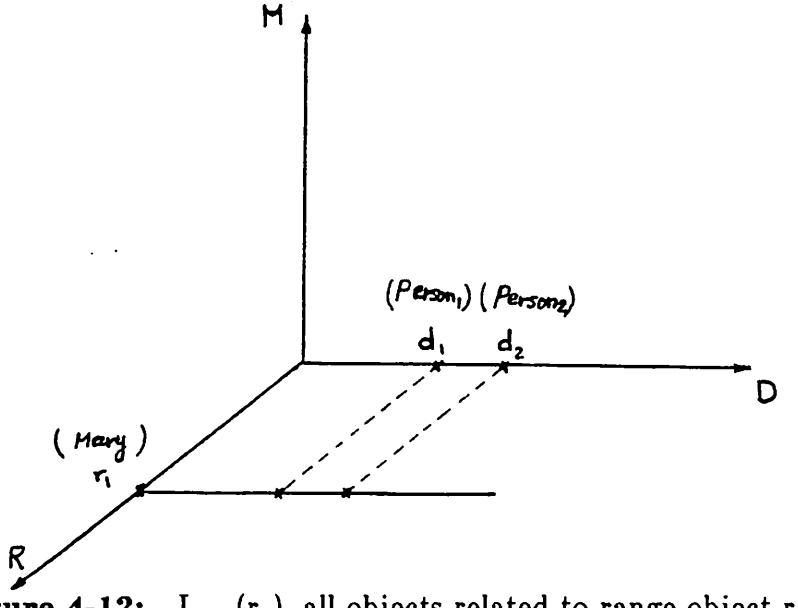


Figure 4-12: $L_{DR}(r_1)$, all objects related to range object r_1

- $(d_1,m_1,?)$: Locus of all range objects related to a given domain object under a given mapping

A $(d_1,m_1,?)$ line, also named L_R, emanates from a point A on DM plane, perpendicular to that plane. This line represents the result of applying the mapping m_1 to object d_1 (the M and D coordinates of point A, respectively). A L_R then, is a collection of points that correspond to all objects on the R-axis, that are in the image of d_1 under m_1 (see Figure 4-13). L_R lines are defined as:

$$L_R(d_1,m_1)=(d_1,m_1,?)=\{(d_1,m_1,r) \mid r \text{ is a point on the R-axis}\}$$

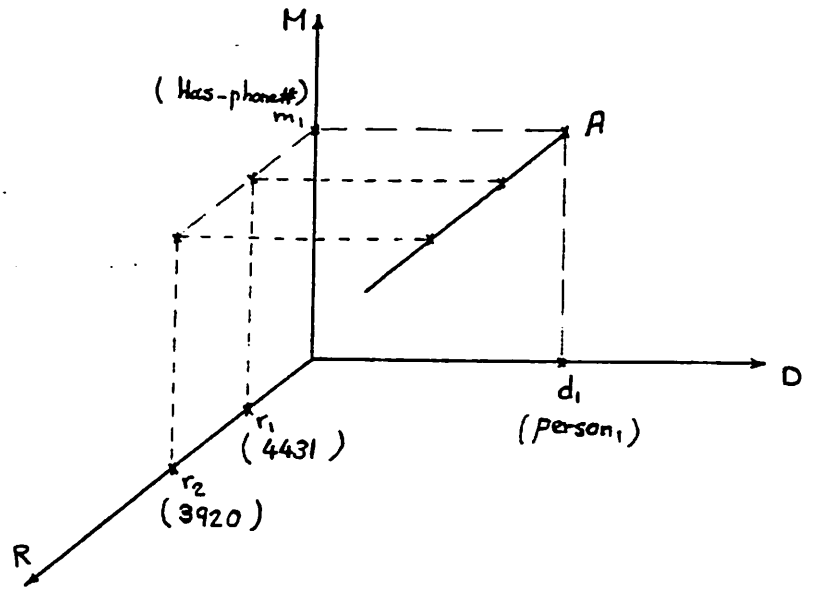


Figure 4-13: $L_R(d_1,m_1)$, all range objects related to domain object d_1 under mapping m_1

- $(?,m_1,r_1)$: Locus of all domain objects related to a given range object under a given mapping

A $(?,m_1,r_1)$ line, also named L_D, emanates from a point B on the MR plane, perpendicular to that plane. This line represents the result of applying the mapping $inv(m_1)$ to object r_1, where m_1 and r_1 are the M and R coordinates of point B, respectively. A L_D then, is a collection of points that correspond to all objects on the D-axis that are in the image of r_1 under $inv(m_1)$ (i.e. all objects on the D-axis that r_1 is in their image under m_1. See Figure 4-14). L_D lines are defined as:

$$L_D(m_1,r_1)=(?,m_1,r_1)=\{(d,m_1,r_1) \mid d \text{ is a point on the D-axis}\}$$

- $(d_1,?,r_1)$: Locus of all mappings defined between a given pair of domain and range objects

A $(d_1,?,r_1)$ line, also named L_M, emanates from a point C on the DR plane, perpendicular to that plane. This line is the locus of the intersection of all L_Rs defined on d_1, with all L_Ds defined on r_1, where d_1 and r_1 are the D and R coordinates of point C,

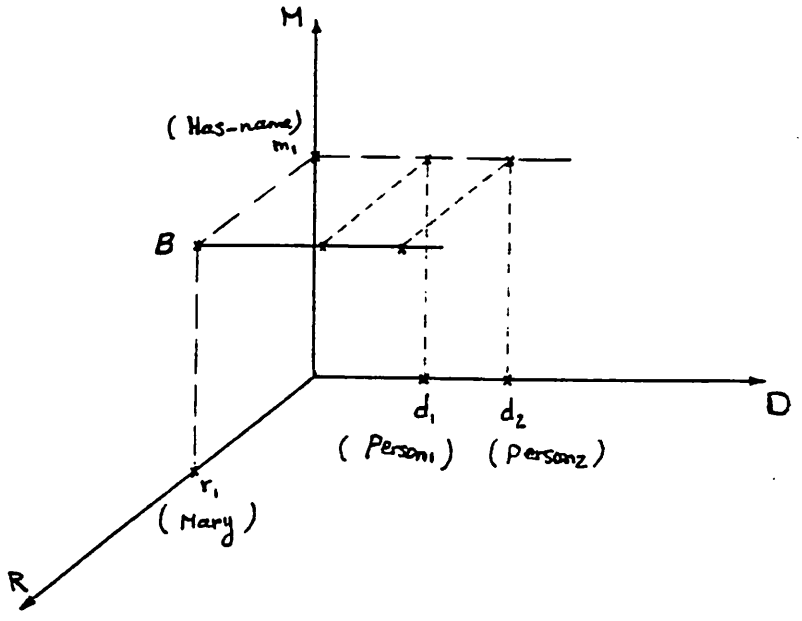


Figure 4-14: $L_D(m_1,r_1)$, all domain objects related to range object r_1 under mapping m_1

respectively. A L_M then, is a collection of points that correspond to all mappings on the M-axis under which the image of d_1 contains r_1 (see Figure 4-15). L_M lines are defined as:

$$L_M(d_1,r_1)=(d_1,?,r_1)=\{(d_1,m,r_1) \mid m \text{ is a point on the M-axis}\}$$

Once again notice that the first six categories of lines represent the application of certain mappings to database objects, and so they eliminate the need to explicitly define any of these functions in 3DIS databases.

See Figure 4-16 for a summary of all lines in the 3DIS.

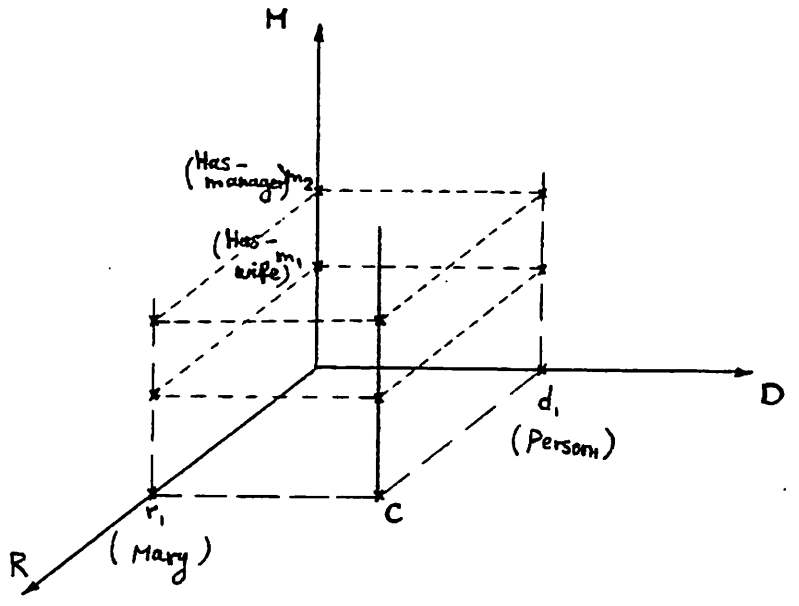


Figure 4-15: $L_M(d_1,r_1)$, all mappings defined between objects d_1 and r_1

4.1.3. Plane Components

All plane components in a 3DIS representation space are discrete and perpendicular to an axis. Every plane emanates from a point located on one of the axes, called its initial point[12]. Therefore, planes contain at least one "on" point, viz. their initial point. A line and a plane *intersect* if and only if their geometric intersection is an "on" point. The intersection of two planes is a line, if and only if there is at least one "on" point on their geometric intersection. Every plane is parallel to one and perpendicular to the other two of the DM, MR, and DR planes. Every plane intersects at lines with two of the DM, MR, and DR planes.

Every plane contains a set of lines that describe its initial point. In fact, a

---

[12]The DM, MR, and DR planes are exceptions to this rule.

D-axis = $(?,-,-) = \{(d_1,-,-) \mid d_1$ is an object$\}$

M-axis = $(-,?,-) = \{(-,m_1,-) \mid m_1$ is a mapping$\}$

R-axis = $(-,-,?) = \{(-,-,r_1) \mid r_1$ is an object$\}$

$L_{MD}(d_1)=(d_1,?,-)=\{(d_1,m,-) \mid m$ is a point on the M-axis$\}$

$L_{MR}(r_1)=(-,?,r_1)=\{(-,m,r_1) \mid m$ is a point on the M-axis$\}$

$L_{DM}(m_1)=(?,m_1,-)=\{(d,m_1,-) \mid d$ is a point on the D-axis$\}$

$L_{RM}(m_1)=(-,m_1,?)=\{(-,m_1,r) \mid r$ is a point on the R-axis$\}$

$L_{RD}(d_1)=(d_1,-,?)=\{(d_1,-,r) \mid r$ is a point on the R-axis$\}$

$L_{DR}(r_1)=(?,-,r_1)=\{(d,-,r_1) \mid d$ is point on the D-axis$\}$

$L_{R}(d_1,m_1)=(d_1,m_1,?)=\{(d_1,m_1,r) \mid r$ is a point on the R-axis$\}$

$L_{D}(m_1,r_1)=(?,m_1,r_1)=\{(d,m_1,r_1) \mid d$ is a point on the D-axis$\}$

$L_{M}(d_1,r_1)=(d_1,?,r_1)=\{(d_1,m,r_1) \mid m$ is a point on the M-axis$\}$

Figure 4-16: A summary of the 3DIS geometric representation lines

plane encapsulates the information about its initial point (an object) by representing all objects directly related to it. Since lines define sets of objects, planes are supersets of objects, defining sets of sets of objects. Every plane consists of two main intersection lines (with two of the DM, MR, and DR planes) and a set of other lines that are of two other categories of lines.

Planes are represented uniquely by ordered triples with question mark symbols "?" in place of two of its elements showing the two-dimensionality of the plane. The "-" symbol is used in the definition of certain planes. For

instance, the DM, MR, and DR planes are represented by (?,?,-), (-,?,?), and (?,-,?) respectively.

DM, MR, and DR are three major planes in the 3DIS representation space. The DM plane, (?,?,-), contains D and M axes, and all of the L_{MD} and L_{DM} lines. The major plane MR, (-,?,?), contains M and R axes, and all of the L_{MR} and L_{RM} lines. The DR plane, (?,-,?), contains D and R axes, and all of the L_{RD} and L_{DR} lines. The three major planes are defined as[13]:

$$DM = (?,?,-) = \{l \mid l \text{ is the D-axis} \vee l \text{ is the M-axis} \vee l \text{ is an } L_{MD} \vee l \text{ is an } L_{DM}\}$$

$$MR = (-,?,?) = \{l \mid l \text{ is the M-axis} \vee l \text{ is the R-axis} \vee l \text{ is an } L_{MR} \vee l \text{ is an } L_{RM}\}$$

$$DR = (?,-,?) = \{l \mid l \text{ is the D-axis} \vee l \text{ is the R-axis} \vee l \text{ is an } L_{RD} \vee l \text{ is an } L_{DR}\}$$

There are three other generic categories of planes in the 3DIS representation space, each encapsulating certain information about its initial point: $(d_1,?,?)$, $(?,?,r_1)$, and $(?,m_1,?)$.

- $(d_1,?,?)$: Locus of all mappings and objects related to a given domain object

[13]The symbol "$\vee$" stands for logical OR.

A $(d_1,?,?)$ plane, also named P_D[14], is the locus of all mappings defined on a given domain object, and all range objects related to that domain object through those mappings. This plane emanates from an object d_1 on the D-axis and is perpendicular to that axis. The plane contains $L_{MD}(d_1)$ and $L_{RD}(d_1)$. All L_Rs for points on L_{MD}, and all L_Ms for points on L_{RD} are also contained in this plane. Therefore, a P_D for a domain object d_1, represent all the information about mappings defined on d_1 and their range elements (see figure 4-17). P_D planes are defined as:

$$P_D(d_1) = (d_1,?,?) = \{\, l \mid l \text{ is } L_{MD}(d_1) \vee l \text{ is } L_{RD}(d_1) \vee l \text{ is an } L_R(d_1,m), \text{ where } m \in L_{MD}(d_1) \vee l \text{ is an } L_M(d_1,r), \text{ where } r \in L_{RD}(d_1) \,\}$$

- $(?,?,r_1)$: Locus of all mappings and domain objects related to a given range object

A $(?,?,r_1)$ plane, also named P_R, is the locus of all mappings for which a given object is in their range, and all domain objects related to that range object through those mappings. This plane emanates

---

[14]The convention for naming of plane components is that one of the three axes (D, M, and R) is used as subscript of **P**, denoting the axis containing the initial point of the plane, i.e. the axis normal to the plane.

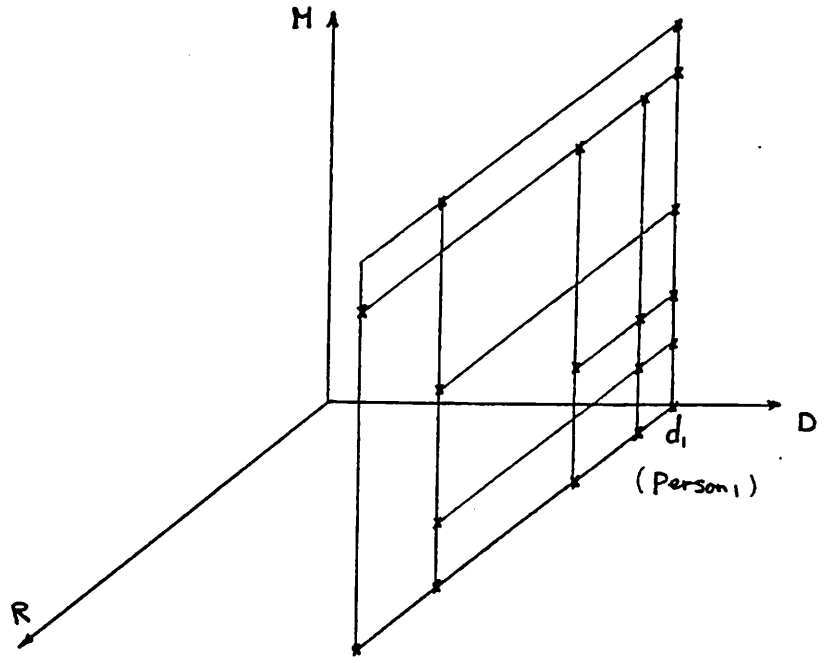


Figure 4-17: $P_D(d_1)$, all mappings and range objects related to d_1 from an object r_1 on the R-axis and is perpendicular to that axis. The plane contains $L_{MR}(r_1)$ and $L_{DR}(r_1)$. All L_Ds for points on L_{MR}, and all L_Ms for points on L_{DR} are also contained in this plane. Therefore, all the information about mappings and their domain elements defined for a range object r_1 is represented by a P_R defined for it (see figure 4-18). P_R planes are defined as:

$$P_R(r_1) = (?,?,r_1) = \{\, l \mid l \text{ is } L_{MR}(r_1)$$
$$\vee\ l \text{ is } L_{DR}(r_1)$$
$$\vee\ l \text{ is an } L_D(r_1,m), \text{ where } m \in L_{MR}(r_1)$$
$$\vee\ l \text{ is an } L_M(r_1,m), \text{ where } m \in L_{DR}(r_1)\}$$

In a P_D and a P_R plane for a specific object the same information is represented differently. For example, in Figure 4-19, a part of the information on the P_D for object A is:

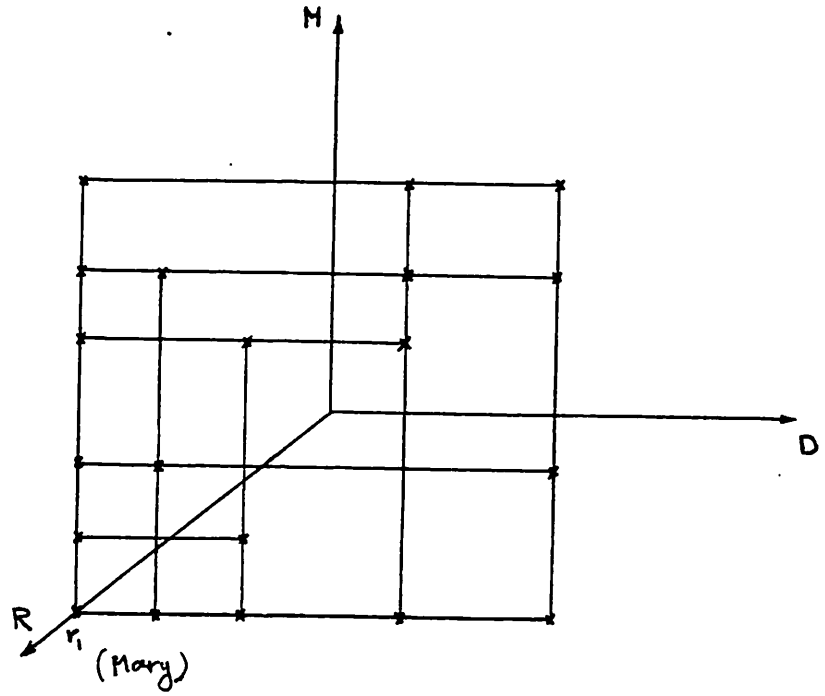


Figure 4-18: $P_R(r_1)$, all mappings and domain objects related to r_1

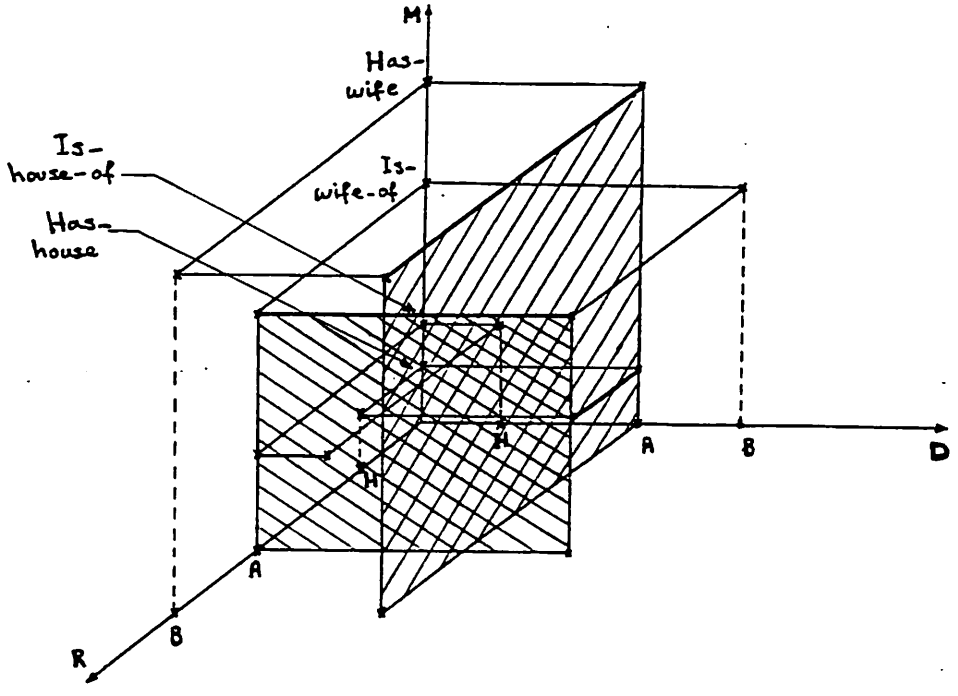


Figure 4-19: Object A is both a domain and a range element in a 3DIS database

A has-wife B

A has-house H

While the same information on the P_R for object A is:

B is-wife-of A

H is-house-of A

Although the meaning of the information kept in both planes is the same, they are certainly represented differently, and one uses the inverse mappings used in the other one.

- $(?,m_1,?)$: Locus of all objects related through a given mapping

A $(?,m_1,?)$ plane, also named P_M, is the locus of all domain and range objects related through a given mapping. This plane emanates from a mapping m_1 on the M-axis, and is perpendicular to that axis. The plane contains $L_{DM}(m_1)$ and $L_{RM}(m_1)$. All L_Rs for points on L_{DM}, and all L_Ds for points on L_{RM} are also contained in this plane. Therefore, a P_M for a mapping m_1 represents all the information about the domain and range elements related through m_1 (see figure 4-20). P_M planes are defined as:

$$P_M(m_1) = (?,m_1,?) = \{\, l \mid l \text{ is } L_{DM}(m_1) \vee l \text{ is } L_{RM}(m_1) \vee l \text{ is an } L_R(m_1,d), \text{ where } d \in L_{DM}(m_1)$$

$\forall$ l is an $L_D(m_1, r)$, where $r \in L_{RM}(m_1)$}

See figure 4-21 for a summary of all planes in the 3DIS representation space.

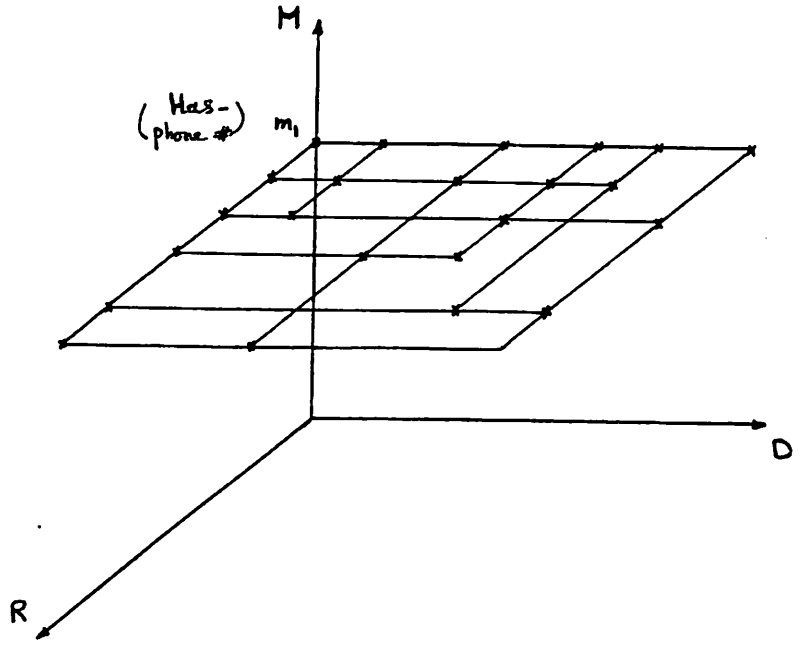


Figure 4-20: $P_M(m_1)$, all domain and range objects related to mapping m_1

4.2. Subcomponents

Subcomponents are defined over the primary geometric components and encapsulate a part of the information represented by those components. Subcomponents are formulated by imposing *conditions* on triples representing geometric components. For example, conditions defined on triples that represent lines and planes result in sublines and subplanes. Generally, conditions can be defined on any element of an ordered triple that is replaced by a "?". Conditions are placed inside brackets and follow the "?" symbol that they apply to. Conditions can be specified using the specification operations

$$DM = (?,?,-) = \{l \mid l \text{ is the D-axis} \vee l \text{ is the M-axis} \vee l \text{ is an } L_{MD} \vee l \text{ is an } L_{DM}\}$$

$$MR = (-,?,?) = \{l \mid l \text{ is the M-axis} \vee l \text{ is the R-axis} \vee l \text{ is an } L_{MR} \vee l \text{ is an } L_{RM}\}$$

$$DR = (?,-,?) = \{l \mid l \text{ is the D-axis} \vee l \text{ is the R-axis} \vee l \text{ is an } L_{RD} \vee l \text{ is an } L_{DR}\}$$

$$P_D(d_1) = (d_1,?,?) = \{ l \mid l \text{ is } L_{MD}(d_1) \vee l \text{ is } L_{RD}(d_1) \vee l \text{ is an } L_R(d_1,m), \text{ where } m \in L_{MD}(d_1) \vee l \text{ is an } L_M(d_1,r), \text{ where } r \in L_{RD}(d_1)\}$$

$$P_R(r_1) = (?,?,r_1) = \{ l \mid l \text{ is } L_{MR}(r_1) \vee l \text{ is } L_{DR}(r_1) \vee l \text{ is an } L_D(r_1,m), \text{ where } m \in L_{MR}(r_1) \vee l \text{ is an } L_M(r_1,m), \text{ where } m \in L_{DR}(r_1)\}$$

$$P_M(m_1) = (?,m_1,?) = \{ l \mid l \text{ is } L_{DM}(m_1) \vee l \text{ is } L_{RM}(m_1) \vee l \text{ is an } L_R(m_1,d), \text{ where } d \in L_{DM}(m_1) \vee l \text{ is an } L_D(m_1,r), \text{ where } r \in L_{RM}(m_1)\}$$

Figure 4-21: A summary of the 3DIS geometric representation planes described in Section 3.3, to define sets of object-ids (see Section 4.2.1 for examples). Three sets of operators are used to define conditions:

1. Comparison operators

<	less than
<=	less than or equal to
=	equal to
¬=	not equal to
>=	greater or equal to
>	greater than

2. Boolean operators

AND	logical and
OR	logical or
n NOT	logical negation

3. Set-membership operators

$\in$	a member of
$\notin$	not a member of

There are three major categories of subcomponents in the 3DIS representation space: sublines, subplanes, and subspaces.

4.2.1. Subline Components

Every subline component is defined over a specific line and contains a subset of the "on" points located on that line. A subline SL of a line N encapsulates a part of the information represented by N, for which a certain condition holds.

Every line N contains a set of points, so for a subline SL of N, SL $\subseteq$ N. An example of a subline SL_1 of the D-axis= (?,-,-), is the line that contains points only for those objects that are members of a certain type object. Figure 4-22 illustrates a subline of the D-axis that contains members of the type **PERSON**.

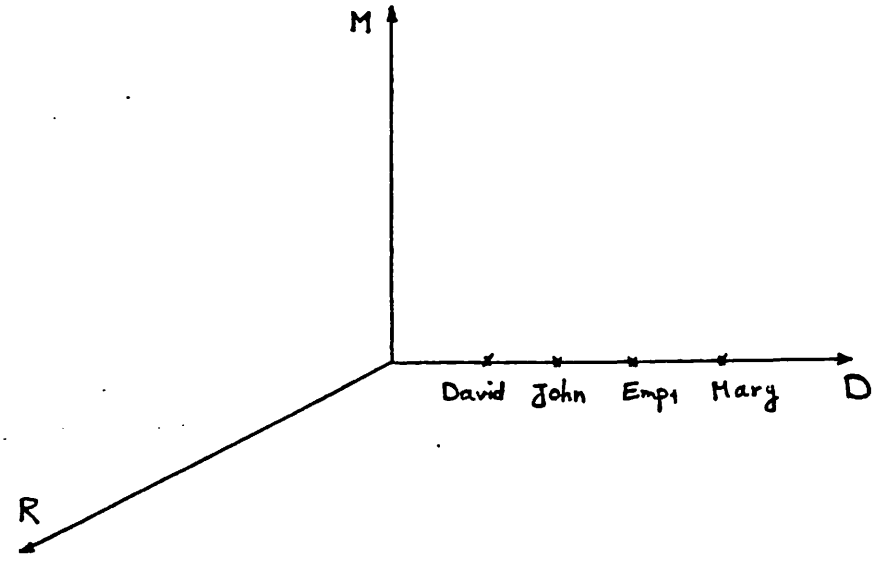


Figure 4-22: SL_1, a subline of D-axis

This subline is denoted by:

$$SL_1 = (?[\in PICK\text{-}R(PERSON, Has\text{-}member, ?)], -, -)$$

An interesting example of sublines is SL_2 defined on $L_R(Dress_1,$

Has-size)=(Dress$_1$, Has-size, ?), such that it contains points only for the sizes between **7** and **13**. Figure 4-23 illustrates L_R(Dress$_1$, Has-size) and Figure 4-24 illustrates the SL_2 subline that is denoted by:

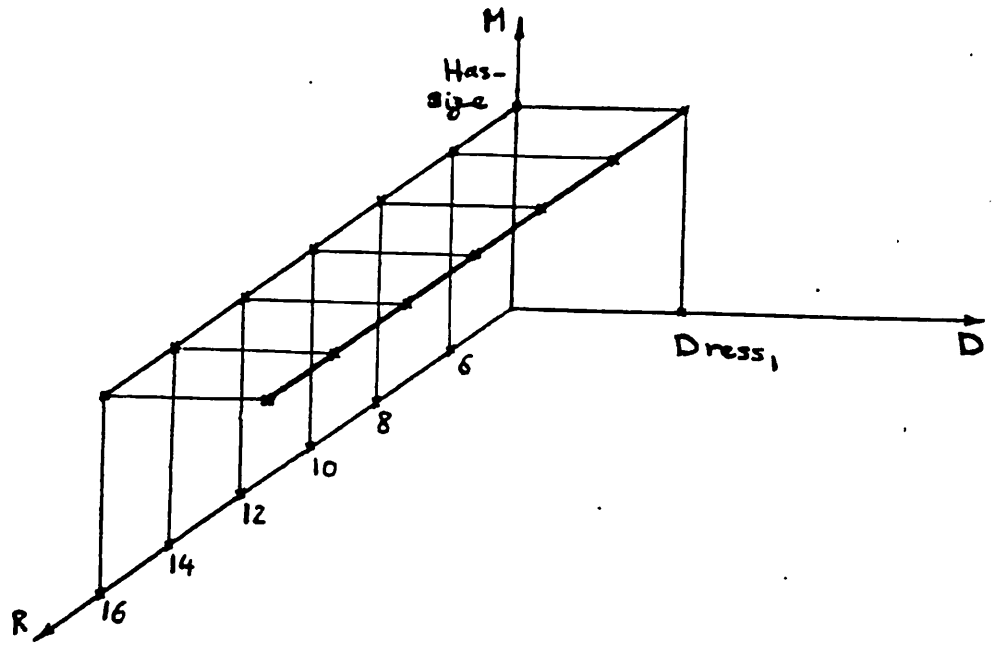


Figure 4-23: A representation of L_R(Dress$_1$, Has-size)

SL_2 = (Dress$_1$, Has-size, ?[>=7 AND <=13]).

4.2.2. Subplane Components

Every subplane component is defined over a specific plane, and contains a subset of the sublines located on that plane. A subplane SP of a plane T encapsulates a part of the information represented by T, for which certain conditions hold.

Every plane T contains a set of points, so for a subplane SP of T, SP $\subseteq$ T. An example of a subplane SP_1 of the P_M(Has-phone#)=(?, Has-phone#, ?)

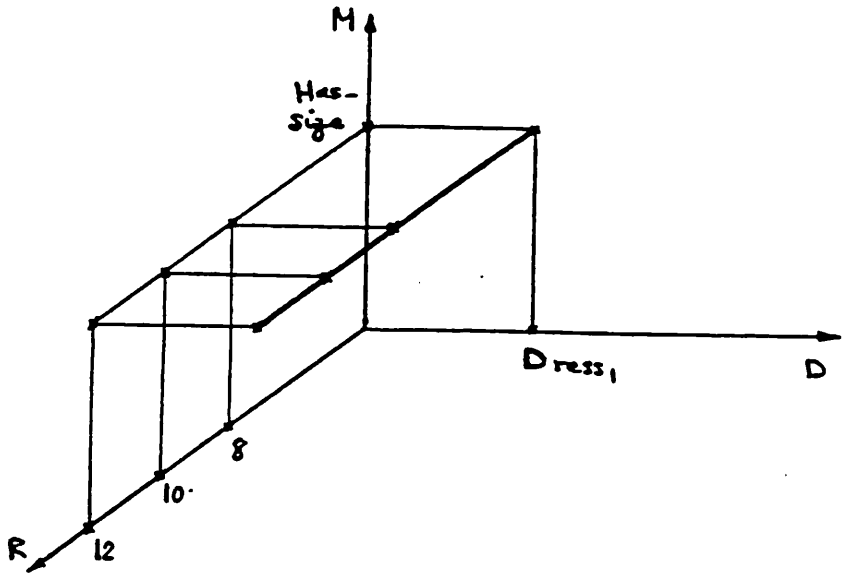


Figure 4-24: SL_2, a subline of L_R(Dress$_1$, Has-size)

plane, is the plane that contains only those L_R(Has-phone#,d) where d is either **John** or **Mary** and those L_D(Has-phone#,r) where r is not 623-1444. This subplane represents all phone numbers of **John** and **Mary** other than **623-1444**. Figure 4-25 illustrates this subplane that is denoted by:

SP_1 = (?[John OR Mary], Has-phone#, ?[¬= 623-1444])

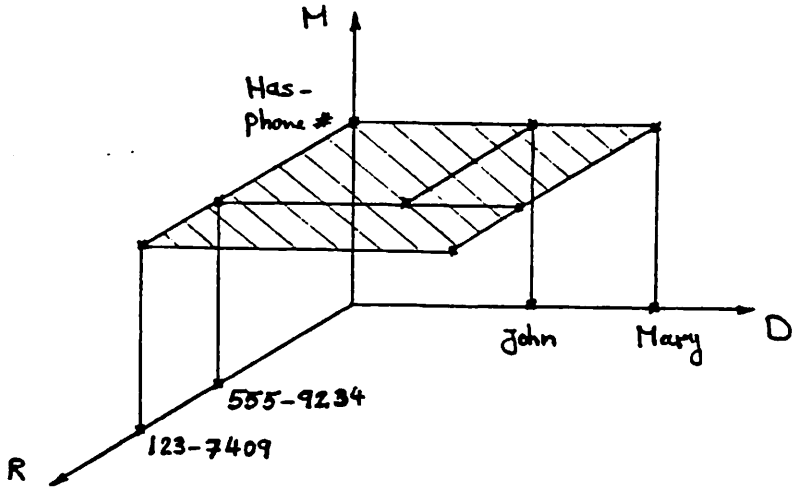


Figure 4-25: SP_1, a subplane of P_M(Has-phone#)

4.2.3. Subspace Components

Subspace components are discrete, orthogonal sub-representation-spaces. Every subspace corresponds to a certain type object and contains a specific subset of the subplanes in the 3DIS representation space that describe members of that type. Therefore, subspaces encapsulate the information about (members of) type objects.

Subspaces of type objets are defined as sets of subplanes defined on P_Ds of members of those type objects. The subspace SS defined for a type T contains all lines on the geometric intersections of all the P_Ds of the members of T with all the P_Ms of the member-mappings defined on T (mappings defined on T's members).

SS(T) ={ SP_1 | SP_1 is a subplane of a $P_D(d)$, where d is
a member of T, containing only those lines
on the intersection of $P_D(d)$ with any $P_M(m)$,
where m is a member-mapping defined on T }

This subspace is denoted by:

SS(T) = (?[∈PICK-R(T, Has-member, ?)]
,?[∈PICK-R(T, Has-member-mapping, ?)], ?)

The reason for defining subspaces in terms of subplanes of p_Ds (instead of P_Ds themselves) is that in general, $P_D(d_1)$ may define d_1 as a member of more

than one type. The 3DIS data model allows objects to be members of more than one type object, e.g. d_1 can be a member of both PERSON and EMPLOYEE. By definition, the $P_D(d_1)$ in a 3DIS representation space contains all information about d_1. Therefore, $P_D(d_1)$ describes d_1 as a member of both PERSON and EMPLOYEE.

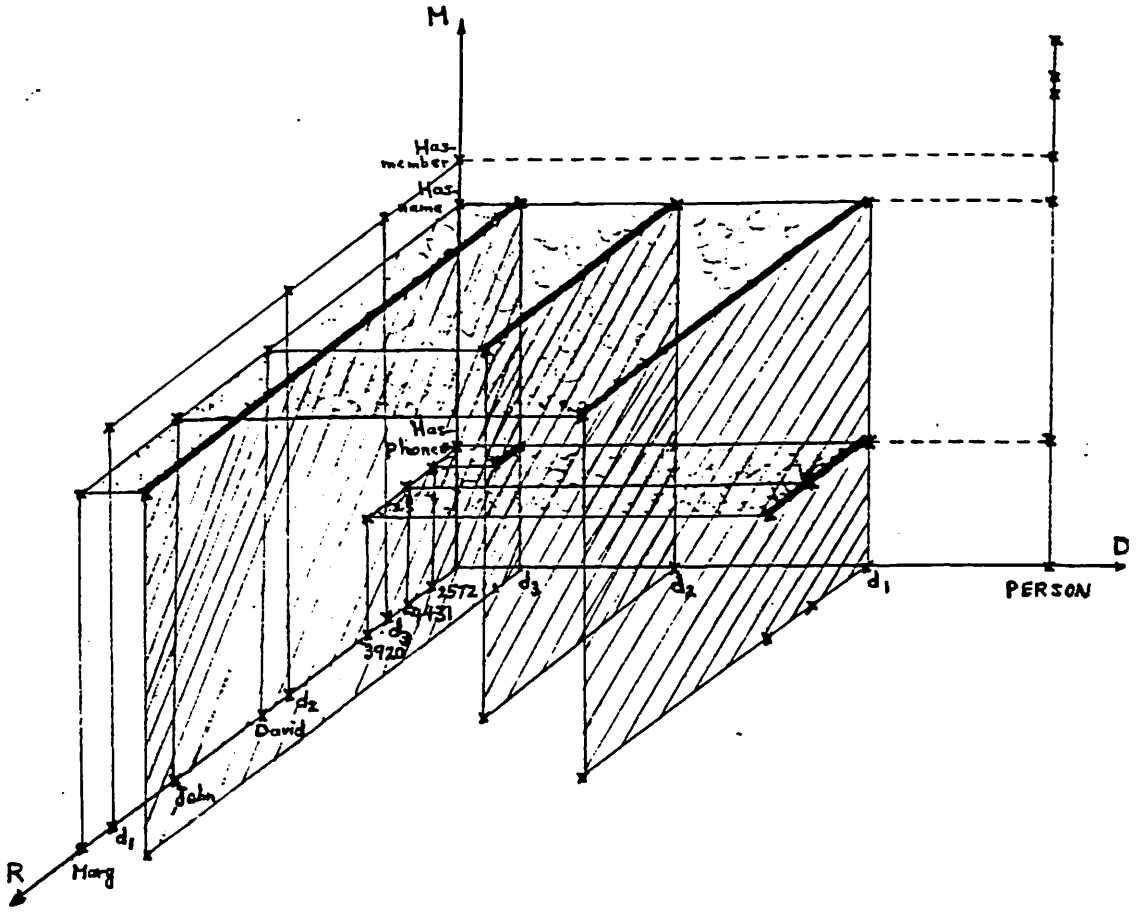
Suppose EMPLOYEE is a subtype of PERSON. Furthermore, assume that has-name and has-phone# are member-mappings defined on PERSON, and has-salary and has-manager are member-mappings defined on EMPLOYEE[15]. By the definition of subspaces, if we intersect the P_Ds of all members of PERSON (including d_1) with the P_Ms of the two member-mappings defined on PERSON, has-name and has-phone#, the result will be the subspace of PERSON as shown in Figure 4-26.

For the subspace of EMPLOYEE however, the P_Ms of member-mappings of EMPLOYEE contain planes for has-name, has-phone#, has-salary, and has-manager as illustrated in Figure 4-27. Therefore, for d_1, as a member of PERSON (in PERSON's subspace), we only get its name and phone numbers (see Figure 4-26), while for d_1 as a member of EMPLOYEE (in EMPLOYEE's subspace) we get its name, phone numbers, salary, and manager (see Figure 4-27).

The subspaces (SS) of PERSON and EMPLOYEE are:

---

[15]By the definition of subtypes, has-name and has-phone# are also defined on EMPLOYEE.



represents P_Ds of members of PERSON

represents P_Ms of member-mappings defined on PERSON

x represents on points

represents the information content of PERSON's subspace

Figure 4-26: SS(PERSON), the subspace of PERSON

SS(PERSON) ={ SP_1 | SP_1 is a subplane of a $P_D(d)$, where d is a member of PERSON (i.e. d_1, d_2, or d_3), containing only $L_R(d, m)$s, where m is a member-mapping defined on PERSON (i.e. has-name or has-phone#) }

SS(EMPLOYEE) ={ SP_2 | SP_2 is a subplane of a $P_D(d)$ where d is a member of EMPLOYEE (i.e. d_1 or d_3), containing only $L_R(d, m)$s, where m is a member-mapping defined on EMPLOYEE (i.e. has-name, has-phone#, has-salary, or has-manager) }

These subspaces are denoted by the following ordered triples:

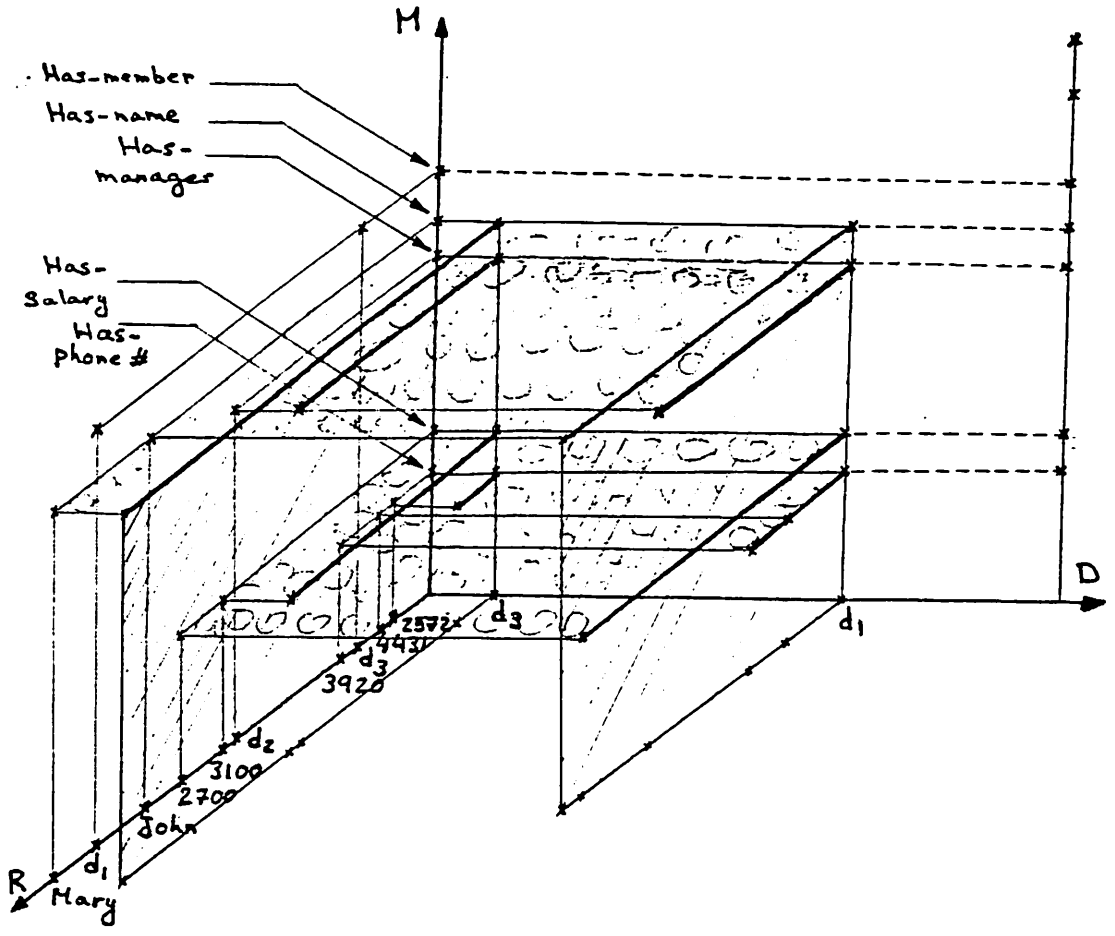
SS(PERSON) = (?[∈PICK-R(PERSON, Has-member, ?)]
,?[∈PICK-R(PERSON, Has-member-mapping, ?)], ?)

SS(EMPLOYEE) = (?[∈PICK-R(EMPLOYEE, Has-member, ?)]
,?[∈PICK-R(EMPLOYEE, Has-member-mapping, ?)], ?)

In Figure 4-26, d_1, d_2, and d_3 are members of the type PERSON. Has-name and has-phone# are the member-mappings defined on PERSON. The following relationships are defined on d_1, d_2, and d_3[16]:

- (d_1, has-name,{"John"})
- (d_1, has-phone#,{"4431","3920"})
- (d_2, has-name,{"David"})

[16]Notice that no phone# is defined for d_2.



represents P_Ds of members of EMPLOYEE

represents P_Ms of member-mappings defined on EMPLOYEE

x represents on points

represents the information content of EMPLOYEE's subspace

Figure 4-27: SS(EMPLOYEE), the subspace of EMPLOYEE

- (d_3, has-name,{"Mary"})
- (d_3, has-phone#,{"2572"})

In Figure 4-27, d_1 and d_3 are members of the type EMPLOYEE. Has-name, has-phone#, has-salary, and has-manager are the member-mappings defined on EMPLOYEE. The following relationships are defined on d_1 and d_3:

- (d_1, has-name,{"John"})
- (d_1, has-phone#,{"4431","3920"})
- (d_1, has-salary,{"2700"})
- (d_1, has-manager,{d_2})
- (d_3, has-name,{"Mary"})
- (d_3, has-phone#,{"2572"})
- (d_3, has-salary,{"3100"})
- (d_3, has-manager,{d_2})

Some notes on Figures 4-26 and 4-27:

- For simplicity, inverse mappings are not shown in these figures, and all objects are represented only on one of the axes.
- Quoted names on the axes represent atomic objects, e.g. "3100".
- Names in capital letters, e.g. PERSON and EMPLOYEE, represent type objects.
- Names on the M-axis, e.g. has-name, etc., represent mapping objects.
- All other names on the axes represent composite objects, e.g. d_1, d_2, and d_3.

Chapter Five

A Specification of the ISL User Interface

The ISL (Information SubLanguage) is a simple object-oriented user interface that provides a small set of primitives to uniformly define, access, retrieve, modify, activate (for procedural objects), and graphically view and browse through the structural (meta-data) and non-structural (data) database contents. It is designed for novice users, however, experienced users are provided with efficient ways to use this interface. User interface commands are presented in menu format, thus eliminating the need to remember their names and format. ISL provides a *browsing-oriented database retriever* and a *menu-oriented database editor.*

The browsing-oriented database retriever is defined on top of the 3DIS geometric representation, provides a graphical spatial display of database information, and supports *database navigation* and *database query.* Database retriever takes advantage of the capabilities available on graphic systems to provide a user-friendly interface. In particular, database information is displayed on the screen of a graphics device and the navigational and query operations are activated through graphic input devices such as light pens or mice to make browsing simple and intuitive. Database retriever provides

capabilities for displaying only that part of the database information that is most relevant to what a user wants to investigate. Therefore, users have dynamic rearrangement tools that allow them to concentrate on and "see" the relevant objects only.

A novice user can start browsing at the ROOT object[1], browse through the descriptive information about the database (meta-data), and continue to explore the data to express a query, using the same set of navigational commands all along. This differs from the conventional approach, where users are required to access the meta-data first, remember names and formats of attributes, and then express a query in a query language. For the experienced users however, an alternative is provided that allows them to search and access objects directly by their *object-ids*. Therefore, experienced users may quickly locate the desired object(s) and then proceed to move around and express queries.

The database editor provides a small set of primitives for object-definition, object-classification, object-interrelation, object-manipulation, and object-activation (for procedural objects). These operations are provided in a hierarchy of menus and users can activate them by pointing, e.g. with a mouse.

As a part of this research, an experimental ISL user interface prototype is developed that uses a high-resolution bit-map display along with a keyboard for

---

[1]See Section 3.2.2 for definition of the ROOT.

input and a mouse as a pointing device. In the remainder of this chapter a specification of the ISL user interface is given in terms of its database retriever and database editor, and the ISL prototype implementation is described.

5.1. Browsing-Oriented Database Retriever

The database retriever is a simple, user-friendly tool that supports display (viewing), browsing, and retrieval of database information within the framework of the 3DIS geometric representation. The database retriever consists of two sets of operations: 3-D navigational operations and query operations.

A standard paradigm for use of the screen and the pointing device is established. A "display window" shows views of "information neighborhoods" of a database on the screen. A menu of navigational commands provides choices of *viewing operations* and *moving operations* that apply to the displayed information.

Browsing inside a 3DIS database imaginatively resembles navigating a space-ship through a cubic cut of a galaxy. Every piece of information is represented by a fixed star in space (that corresponds to an "on" point in the geometric representation space), and every database object is represented by a "cluster" of stars (that corresponds to a plane component in the geometric representation space). If users know and specify which star they want to reach, the system can automatically take their ships to the destination. However, users may only know some of the properties of a specific star (or a cluster of

stars) in question. In such cases, users may browse through the information space and explore the nearby stars in a cluster or among different clusters, in order to discover the specific stars they wish to investigate. Movements in the 3DIS representation space, and consequently, browsing through ISL is confined to navigation between discrete points parallel to an axis.

5.1.1. 3-D Navigational Operations

Navigational operations consist of a few simple generic commands that support object-search and object-viewing. These operations are defined independent of their start position. For less complicated queries, database navigation can replace the use of a query language. Not only such replacement decreases the amount of knowledge users must acquire before they can query a database, it also increases their performance since the ability to map operations onto actions needs the minimum amount of thinking. Navigational operations consist of the two sets of *viewing* and *moving* operations.

5.1.1.1. Viewing Operations

Viewing operations support displaying database information on the screen. These operations consist of seven commands. *R-view*, *L-view*, *T-view*, and *P-view* that illustrate various views of the database information as organized in the 3DIS geometric representation space. The *Scroll* command enables users to focus on the specific portion of the information they want to investigate.

Add-view and *Del-view* allow merging of several views of the same kind into one display and removing views from displays, respectively.

- The R-view (Right-view) command takes the object-id of an object located on the D-axis (specified by user) and displays a 2-dimensional view that is the P_D plane of that object. The P_D plane of a given (domain) object contains all (mapping and range) information directly related to that object. This view is called the R-view since it represents the view of an object as seen from the "right-side" (looking in the direction of the D-axis). Figure 5-1 represents the R-view of **John**.

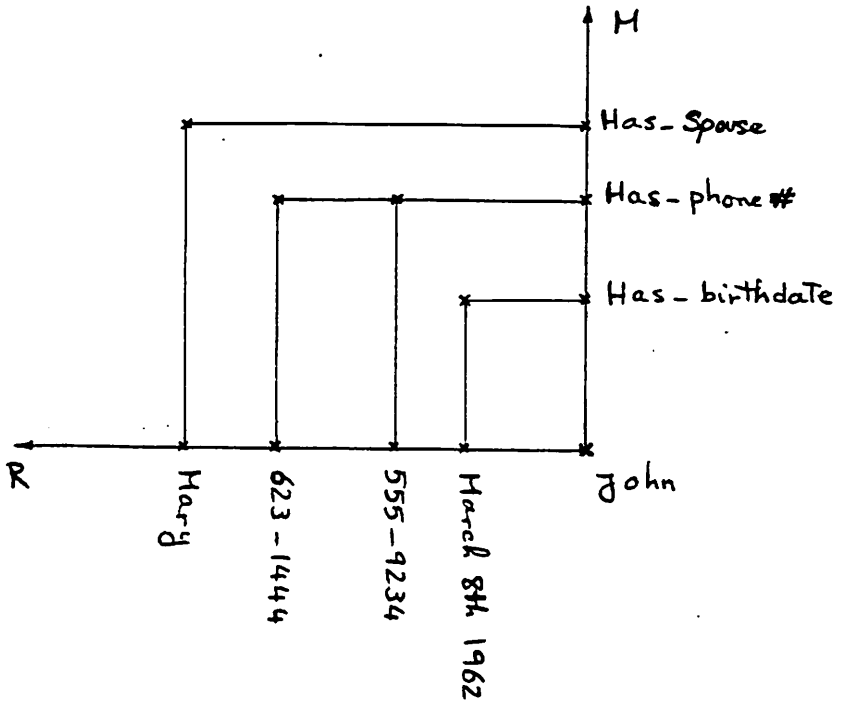


Figure 5-1: The R-view of **John**

- The L-view (Left-view) command takes the object-id of an object

located on the R-axis (specified by user) and displays a 2-dimensional view that is the P_R plane of that object. The P_R plane of a given (range) object contains all (mapping and domain) information directly related to that object. This view is called the L-view since it represents the view of an object as seen from the "left-side" (looking in the direction of the R-axis). Figure 5-2 represents the L-view of **March 8th 1962**.

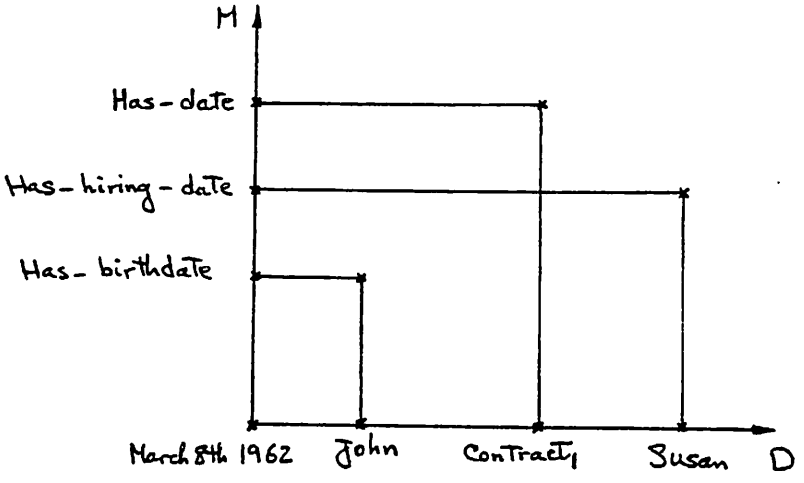


Figure 5-2: The L-view of **March 8th 1962**

- The T-view (Top-view) command takes the object-id of an object located on the M-axis (specified by user) and displays a 2-dimensional view that is the P_M plane of that object. The P_M plane of a given (mapping) object contains all (domain and range) information directly related to that object. This view is called the T-view since it represents the view of an object as seen from the "top" (looking in the direction of the M-axis). Figure 5-3 represents the T-view of **Has-phone#**.

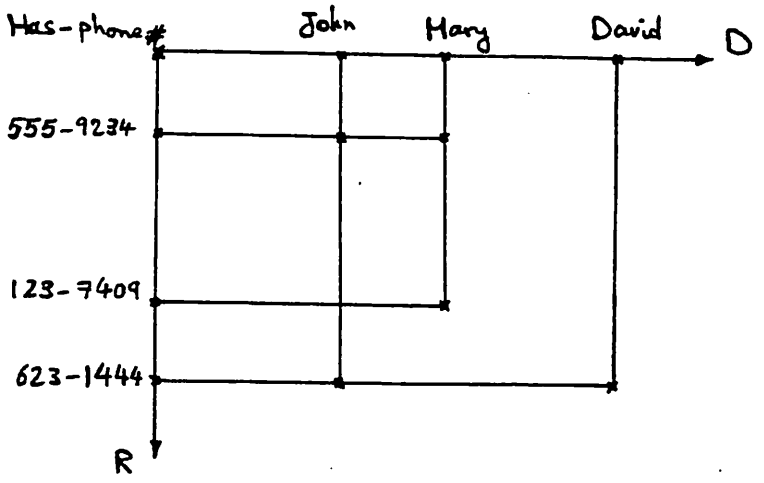


Figure 5-3: The T-view of **Has-phone#**

- The P-view (Perspective-view) command takes the object-id of a type object (specified by user) and displays a 3-dimensional view of the subspace of that object. The subspace of a given type object contains the subset of all "on" points in the 3DIS representation space that describe members of that type. Perspective views may get very crowded and are useful only for the high level investigation and viewing of a type. Detailed information about type members are better viewed and accessed by other viewing operations. Figure 5-4 represents the P-view of the type **CLOSE-FRIENDS** which has two member-mappings of Has-phone# and Has-birth-date, and the two members **John** and **Mary**.
- The scroll command is a 3-D navigational operation. This operation is necessary because any graphics device terminal has a restricted display capacity. Therefore, often the information content of a view

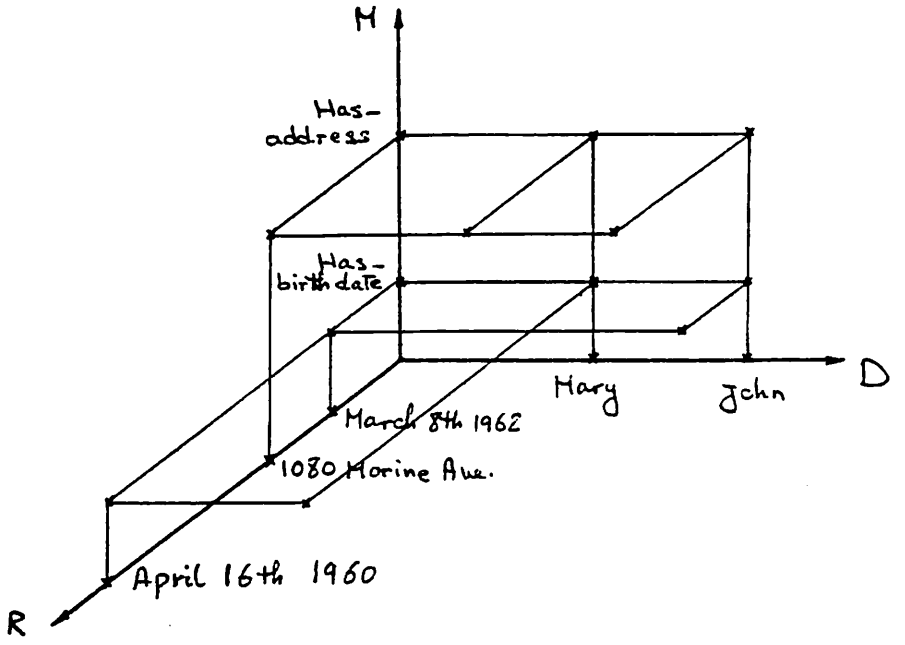


Figure 5-4: The P-view of **CLOSE-FRIENDS**

cannot be clearly displayed on one screen. The scroll operation treats the objects located on each axis as if they were arranged on a scroll. Users can move objects in and out of the screen by scrolling the axes up and down, and focus on that portion of the geometric representation space that contains the information they want to investigate.

- The Add-view command allows merging of views with the current view displayed on the screen. Merging views of objects shows the information that is common among them. For instance, Figure 5-5 represents the Add-view of the R-views of **John** and **David**. In this figure, points representing the common information between the two R-views, e.g. the phone number **623-1444**, are highlighted by a different icon: a bullet ("•").

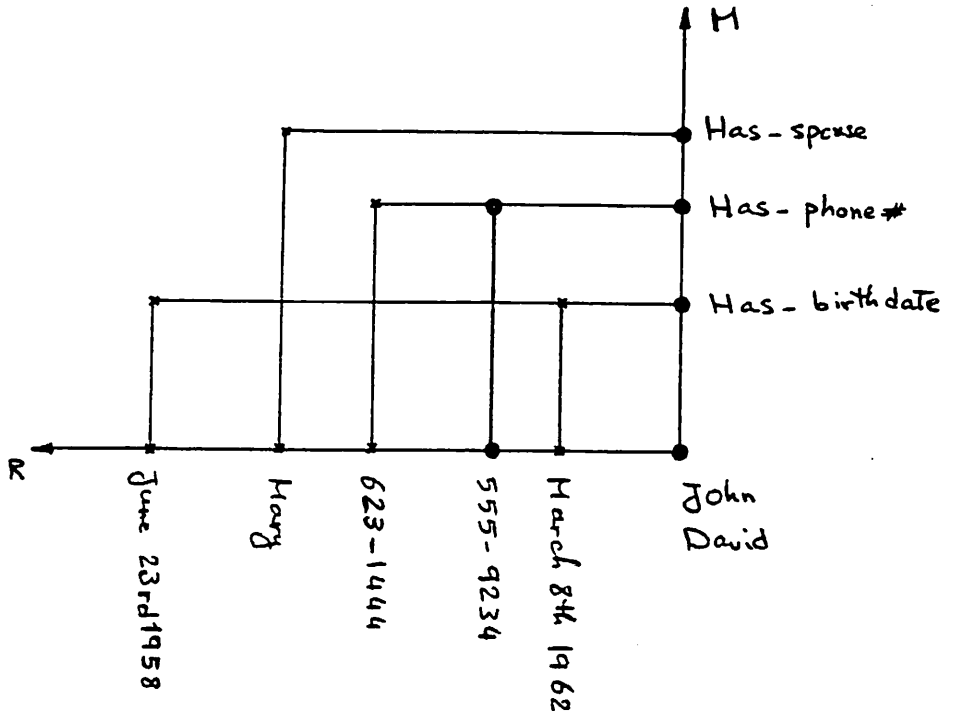


Figure 5-5: The R-views of **John** and **David** merged

- The Del-view command supports removing views from the current display. Removing the R-view of **David** from the view in Figure 5-5, leaves the R-view of **John** shown in Figure 5-1.

5.1.1.2. Moving Operations

Moving operations are cursor-oriented commands that support browsing through the database information shown in the current view. There are eight such commands. *Move-DP*, *Move-DN*, *Move-MP*, *Move-MN*, *Move-RP*, and *Move-RN* allow moving the cursor parallel to the three axes in either *Positive* (away from the origin) or *Negative* (towards the origin) directions. The *Locate* operation moves the cursor to a specified point. The *Ask-coordinates* operation

returns the coordinates of the point located under the cursor. Some simple queries can be answered by this set of operations. For example, investigating the existence of a relationship may be checked either by using the Locate operation or by simply moving in the space between the points that represent this relationship. Another example is to move between the points in the space and use the Ask-coordinates operation repetitively to answer atomic queries described in Section 5.1.2.1.

- The Move-DP command issued at any point in a view moves the cursor parallel to the D-axis in the positive direction to the next point. If no next point exists, cursor remains at the starting point and the user is notified.
- The Move-DN command issued at any point in a view moves the cursor parallel to the D-axis in the negative direction to the next point. If no next point exists, cursor remains at the starting point and the user is notified.
- The Move-MP command is similar to Move-DP, but the cursor moves parallel to the M-axis instead of the D-axis.
- The Move-MN command is similar to Move-DN, but the cursor moves parallel to the M-axis instead of the D-axis.
- The Move-RP command is similar to Move-DP, but the cursor moves parallel to the R-axis instead of the D-axis.
- The Move-RN command is similar to Move-DN, but the cursor moves parallel to the R-axis instead of the D-axis.

- The Locate command moves the cursor to the point in a view whose coordinates match those specified by the user. This command supports direct access to database information shown in a view when users know the exact coordinates of a point, e.g. an object.
- The Ask-coordinates command returns the coordinates of the point under the cursor. This operation eliminates the need to remember coordinates of points and is especially helpful when the information about the coordinates of a point are outside the screen boundaries.

5.1.2. Query Operations

Query operations consist of simple but functionally powerful primitive operations defined on objects and uniformly support the data/meta-data information investigation. The simple generic format of ordered triples is used for simple queries. The ordered triples also denote geometric components as answer to queries. For instance, the query (Emp_1,Has-children,?) that denotes a L_R line, retrieves all children of Emp_1. Query operations consist of *retrieving* and *set-manipulation* operations. At the end of this section, several example queries are provided. These queries are meant to represent the power of these simple query operations in formulating database queries.

5.1.2.1. Retrieving Operations

Retrieving operations support database query and are integrated with the viewing operations to display the result of queries on the screen. These operations all have the format of ordered triples, are defined in terms of geometric components, and return a set of simple triples representing points. Retrieving operations consist of a set of thirteen *atomic* commands and a number of *pseudo-atomic* commands.

Atomic commands are ordered triples defined in terms of geometric components. The first nine commands in this set are ordered triples each denoting a *line* from the nine categories of lines:

$(d_1,?,-)$ denotes the $L_{MD}(d_1)$ line

$(-,?,r_1)$ denotes the $L_{MR}(r_1)$ line

$(?,m_1,-)$ denotes the $L_{DM}(m_1)$ line

$(-,m_1,?)$ denotes the $L_{RM}(m_1)$ line

$(d_1,-,?)$ denotes the $L_{RD}(d_1)$ line

$(?,-,r_1)$ denotes the $L_{DR}(r_1)$ line

$(d_1,m_1,?)$ denotes the $L_{R}(d_1,m_1)$ line

$(?,m_1,r_1)$ denotes the $L_{D}(m_1,r_1)$ line

$(d_1,?,r_1)$ denotes the $L_{M}(d_1,r_1)$ line

For example, (John, ?, Mary) is an atomic query denoting L_M(John,Mary) that returns the set of relationships (geometric points) defined between **John** and

Mary. The next three commands in the set of atomic operations are ordered triples each denoting a *plane* from the three categories of planes:

$(d_1,?,?)$ denotes the $P_D(d_1)$ plane

$(?,?,r_1)$ denotes the $P_R(r_1)$ plane

$(?,m_1,?)$ denotes the $P_M(m_1)$ plane

For example, (John, ?, ?) is an atomic query denoting P_D(John) that returns the set of relationships (geometric points) defined on **John**. The last atomic command in the set is the triple (?,?,?) that denotes the entire database.

Pseudo-atomic commands are ordered triples defined in terms of subcomponents. They too have the format of ordered triples, but impose conditions on those elements of the triples replaced by "?". The conditioned triples with one "?" symbol denote *sublines* and the conditioned triples with two "?" symbols denote *subplanes*. Some conditioned triples with three "?" symbols denote *subspaces* of certain type objects, and the others represent arbitrary sub-representation-spaces.

For pseudo-atomic commands, the system first breaks these commands into their atomic commands and then applies the conditions, while re-enforcing the inherent consistency rules in the geometric representation space. An example of a pseudo-atomic command is (John, ?[¬= Has-spouse AND ¬= Has-birth-date], ?) that denotes a subplane of P_D(John) that does not include the information about his **spouse** and his **birth-date**. This operation returns the

set of relationships (geometric points) defined on **John** not including **Has-spouse** and **Has-birth-date** mappings.

5.1.2.2. Set-manipulation Operations

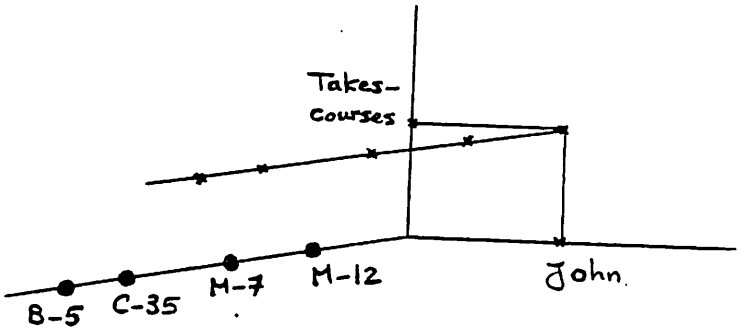
Set-manipulation operations are used together with the retrieving operations to formulate database queries. These operations are *Union*, *Intersection*, and *Difference*, each defined on two sets of points and returns a set of points. Set-manipulation operations also include three other operations: *Pick-d*, *Pick-m*, and *Pick-r* whose detail definitions are given in Sections 3.3.8, 3.3.9, and 3.3.10 respectively.

The following examples show the use of query operations in formulating database queries. The perspective-view in every example shows how the query is posed in the geometric representation space and denotes the answer by "•" symbols. Note that:

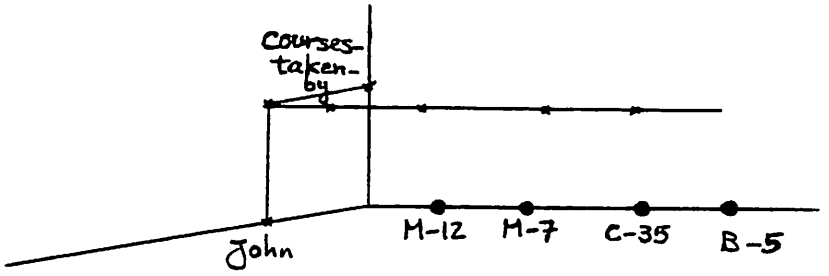
1. The same set of commands is used all along to formulate queries on both data and meta-data.
2. There is always more than one way to formulate a query.
3. Queries that are not easily formulated in most other query languages, e.g. Examples 4 and 5, take the same amount of effort to formulate with the query operations as the simple ones as in Examples 1 to 3.

Example 1- Find all the courses taken by John.

a) Result ⇐ Pick-r(John, Takes-courses, ?)

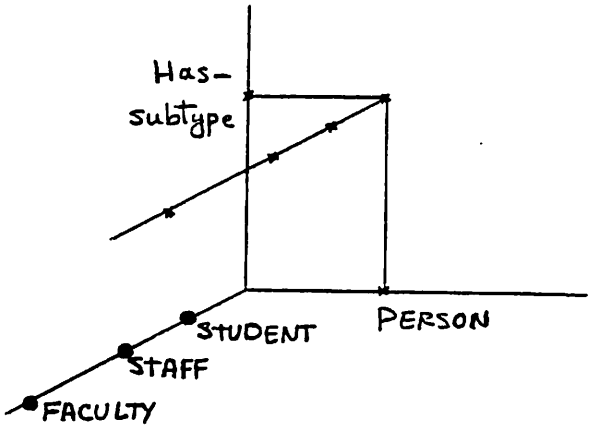


b) Result ⇐ Pick-d(?, Courses-taken-by, John)



Example 2- Find the subtypes of type PERSON.

Result ⇐ Pick-r(PERSON, Has-subtype, ?)

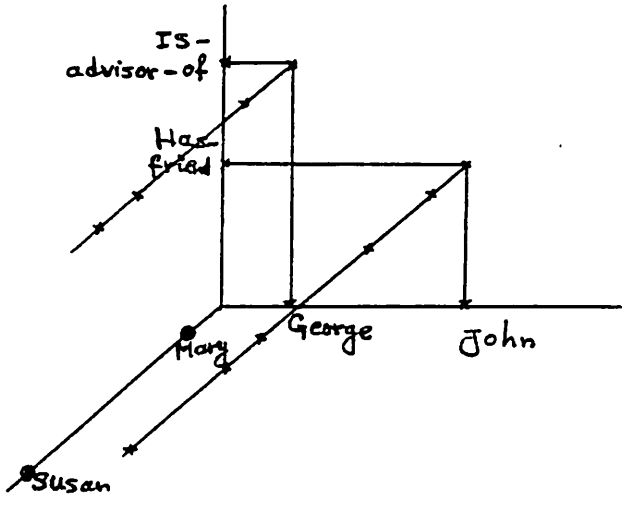


Example 3- Find all friends of John who are advised by George.

JF ⇐ Pick-r(John, Has-friend, ?)

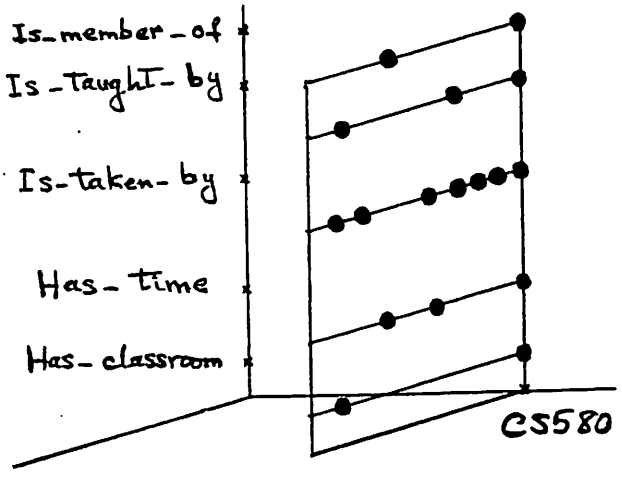
GA ⇐ Pick-r(George, Is-advisor-of, ?)

Result ⇐ Intersection(JF, GA)

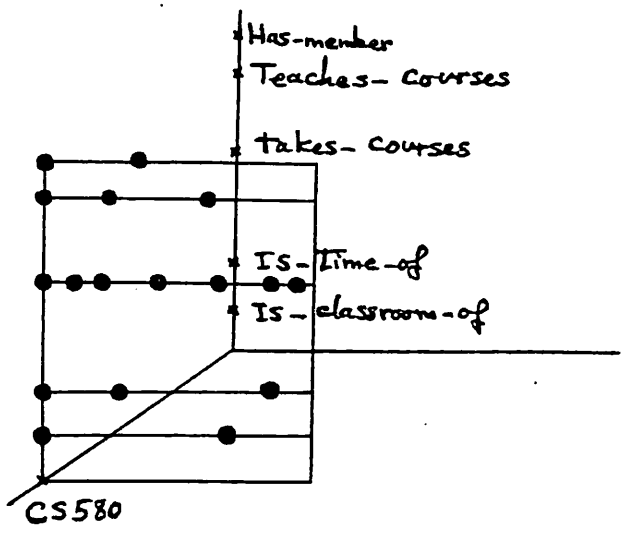


Example 4- What do we know about the course CS580?

a) Result ⇐ (CS580, ?, ?)

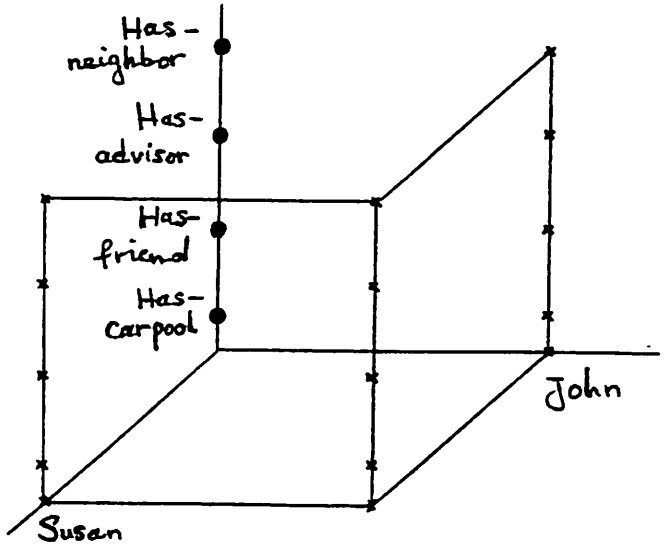


b) Result ⇐ (?, ?, CS580)



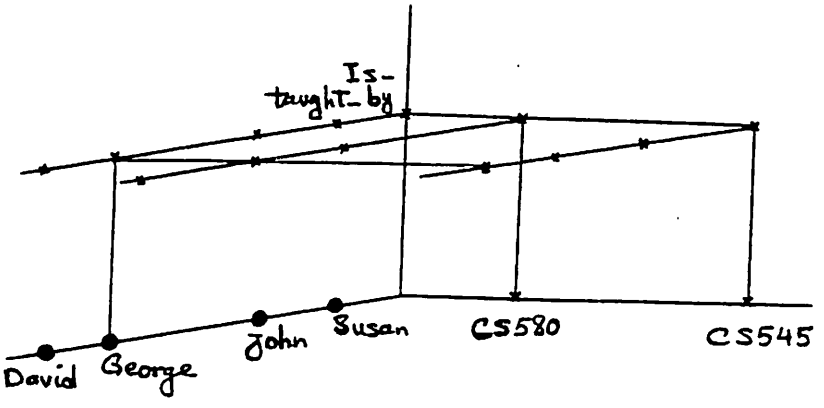
Example 5- What are the direct relationships between John and Susan?

Result ⇐ Pick-m(John, ?, Susan)



Example 6- Find all instructors who teach at least one course taught by George.

```
Result ⇐ Pick-r(?[∈ Pick-d(?, Is-taught-by, George)],
                                    Is-taught-by, ?)
```



The simple query operations directly formulate most frequently used database queries. These operations implicitly supportf the use of *existential quantifier*, as illustrated by Example 6. However, formulation of some complicated queries may require additional operations such as the *universal quantifier*. The current set of simple query operations does not support this quantifier. However, the **For-all** operation can be defined as follows and added to this set. If S is a set, x is a variable ranging over members of S, and R is a retrieving operation, then:

For-all x in S do R does the following:

(1) It repeats R for n>=0 times, where n is the number of members of S, instantiating x to a different member of S each time.

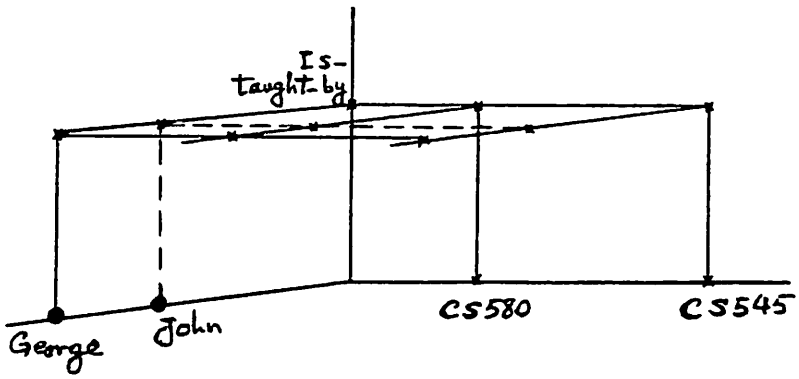
(2) It returns a set whose members are the result of the application of R as described in step (1).

Two more set-manipulation operations can also be defined: *Union** and *Intersection**. These two operations are defined on a set of sets and return a set which is the result of the union or intersection of the members of the given set, respectively.

The following examples show how the extended operations are used.

Example 7- Find all instructors who teach all the courses taught by George.

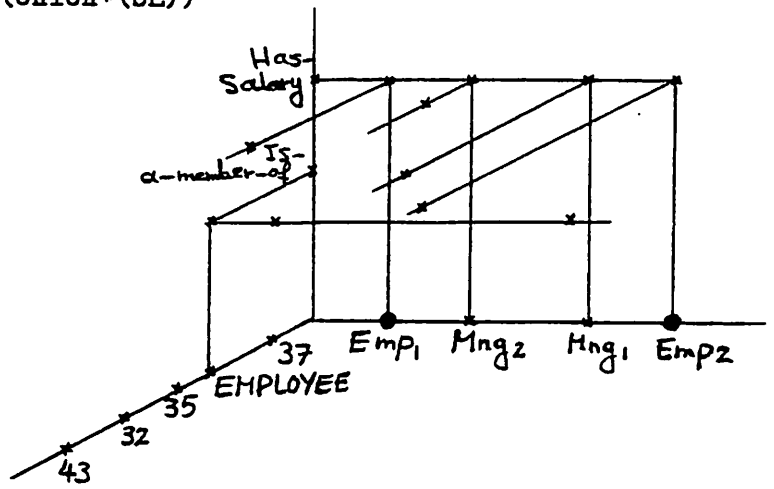
```
CG ⇐ Pick-d(?, Is-taught-by, George)
CI ⇐ For-all x in CG do
          Pick-r(x, Is-taught-by, ?)

Result ⇐ Intersection*(CI)
```



The definition of an existential quantifier seems redundant since queries that typically need to use this quantifier can be formulated using the other operations, as in Examples 6 and 8.

Example 8- Find all employees whose salaries are more than their managers' salaries.

```
EM ⇐ (?, Is-a member-of, EMPLOYEE)
SE ⇐ For-all x in EM do
        (x, Has-salary, ?[> Pick-r
                (Pick-r(x, Has-manager, ?), Has-salary,?)])
RESULT ⇐ Pick-d(Union*(SE))
```



Notice that in Example 8, if the salary of an employee e_1 is not more than his manager's salary, then the

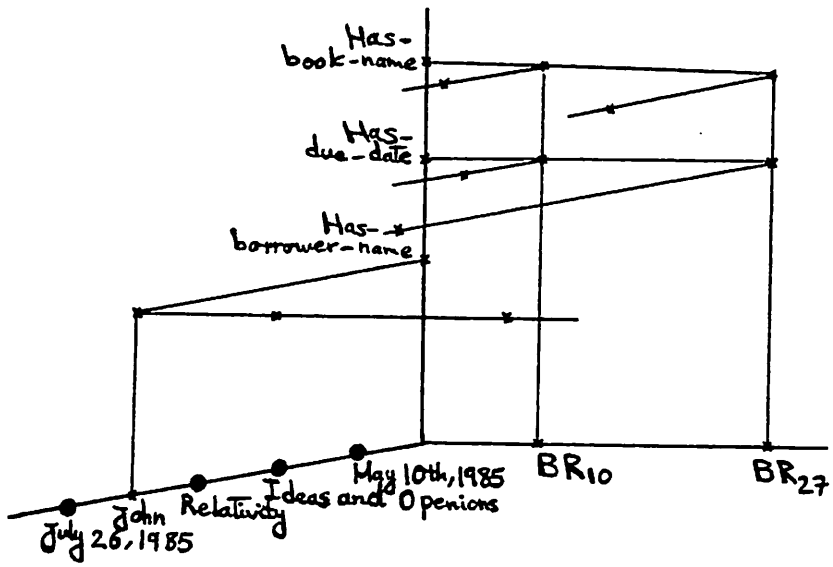
```
(e1, Has-salary, ?[>Pick-r
        (Pick-r(e1,Has-manager, ?), Has-salary, ?)])
```

returns an empty set.

A last operation *Merge* supports merging the pieces of results into the format requested by users. The Merge operation is defined on sequence of sets of points and returns a set that is the cartesian product [Date 81] of those sets. Example 9 shows the use of Merge. Consider a ternary relationship representing a library's book records. Suppose every borrowed-book-record, e.g. $\mathbf{BR_1}$, has a book-name, e.g. **Ideas and Opinions**, a borrower-name, e.g. **John**, and a due-date, e.g. **8/23/85**.

Example 9- For all books borrowed by John, find their names and due-dates.

```
BR ⇐ Pick-d(?, Has-borrower-name, John)
RESULT ⇐ For-all x in BR do
              Merge(Pick-r(x, Has-book-name, ?),
                    Pick-r(x, Has-due-date,?))
```



Example 10 shows how the same query of Example 9 would have been answered without the use of Merge. With the results gained from Example 10, users have to look into two sets of simple triples and match the borrowed-book-record numbers to find out the due-dates of each book borrowed by John.

Example 10- For all books borrowed by John, find their names and due-dates.

```
BR ⇐ Pick-d(?, Has-borrower-name, John)
RESULT1 ⇐ (?[∈BR], Has-book-name, ?)
RESULT2 ⇐ (?[∈BR], Has-due-date, ?)
```

5.2. Menu-Oriented Database Editor

The database editor provides a small set of object-oriented primitives to support editing of data/meta-data information. This simple set of primitives is defined in terms of the *Specification* operations described in Section 3.3 of this dissertation. However, the editor operations are designed for users' convenience and are much easier to learn and use than the specification operations.

Specification operations work with system generated object-ids that uniquely identify objects. The database editor allows users to define *object-names* for objects. Object-names[2] must be globally unique and are surrogate-ids defined by users that can be used in place of the (system generated) object-ids. Supporting object-names in the database editor is especially useful for two reasons:

1. Avoiding the burden of using long object-ids

   The object-id of atomic objects are the information content of those objects. For some atomic objects such as text objects and behavioral objects, the object-ids are very long and thus impractical to use as object identifiers.

2. Avoiding the burden of remembering strange object-ids

   The system generated object-ids of composite and type objects are selected from a different domain than the one describing atomic

[2]Object-names themselves are treated as atomic objects.

objects. This domain is implementation system specific and is quite unfamiliar to users. Therefore, object-ids are generally hard to remember.

In the definition of the database editor operations the term *object-ref* represents a globally unique object identifier that can be either the object-id or an object-name. Therefore, users can specify objects with their object-refs. However, all relationships defined among objects of 3DIS databases are established through their object-ids.

Users of the database editor receive a good deal of help and guidance from the system. Database editor operations are presented in menus and users can activate them by a pointing device. Therefore, users do not need to remember the name and format of the structural information of their databases. Little data needs to be entered through the keyboard, because the system prompts with common defaults.

Database editor operations are organized in a hierarchy of menus. When an operation is selected, the system provides a template for users to fill[3] . The default format for the data to be entered for a template entry is a list of object-refs separated by commas, unless a different format is specified inside brackets, e.g. **specify exactly one object-ref**. Objects must have been created before their object-refs can be specified as values for template entries. However,

---

[3]In all example templates in this section boldfaced words represent system prompts, and italicized words represent users' input.

object-refs can be created dynamically using the *specification* operations. For example in Figure 5-8 "0" and "1" are created on the spot. The result of a specification operation can also replace an object-ref, as in Figure 5-9 where Pick-r(Mary, Has-department, ?) replaces an object-ref. Whenever data is unavailable, template entries can be left blank.

The last item in all menus is *completed*. When the user completes entering data into a template, he chooses the menu-item Completed and the system starts processing that operation. The completion of an operation is indicated by the system prompt **DONE**. The first level of menu hierarchy contains *Create*, *Destroy*, *Connect*, *Disconnect*, *Display*, *Invoke*, *Addname*, *Dropname*, and *Completed*.

5.2.1. Create

A second level of menu is associated with this command that contains *create-a-type*, *Create-an-atomic-object*, *Create-a-composite-object*, and *Completed*.

5.2.1.1. Create-a-type

This command defines type objects. Figure 5-6 illustrates the system generated template for creating a type object. The information for creation of the type object **STUDENT** is filled in this template. When at least some information is filled in the template, the execution of this operation returns a system generated object-id. The user-defined object-name is also returned to user to indicate its acceptance by the system as a globally unique identifier.

```
Create-a-type
```

Has-object-name: *STUDENT*

Has-supertype: *PERSON*

Has-member-mapping: *Has-phone#, Has-status, Has-department*

Has-constraint-evaluator:

Has-storage-transaction: *Select-advisor*

Has-retrieval-transaction: *Status-is-graduate*

Has-object-id: &&$#

Has-object-name: STUDENT

DONE

Figure 5-6: Create the type **STUDENT**

5.2.1.2. Create-an-atomic-object

This command defines atomic objects. Figure 5-7 illustrates the system generated template for creating an atomic object. The information for creation of the atomic object **admit** is filled in this template. Notice that an entry in this template reads **object/object-id** to emphasize the fact that the object-id of atomic-objects are the objects themselves. After the completion of this operation, the system responds if this atomic object is new, or if it has already been defined by prompting either **The object-id is unique** or **The object-id already exists**. The user-defined object-name is also returned to indicate its acceptance by the system as a globally unique identifier.

5.2.1.3. Create-a-composite-object

This command defines composite objects. Figure 5-8 and 5-9 illustrate two system generated templates for creating composite objects. The information for creation of the composite objects **Has-advisor** and **John** are filled in these two templates respectively. Notice that composite objects are defined in two steps. In the first step the user is asked about the type of the composite object and in the second step the system prompts for the instances of the member-mappings of the specified type. When this operation is complete the system generated object-id is returned to the user. User-defined object-names are also returned to indicate their acceptance by the system as globally unique identifiers.

```
Create-an-atomic-object
```

Has-object-name: *admit*

object/object-id[specify exactly one object]:
procedure ADMIT-STUDENT

var A, B

begin

.

.

.

end

Is-a-member-of: *STORAGE-TRANSACTION*

The object-id is unique.

Has-object-name: admit

DONE

Figure 5-7: Create the atomic object **admit**

Create-a-composite-object

Has-object-name: *Has-advisor*

Is-a-member-of: *MAPS*

Has-domain-type: *FACULTY*

Has-range-type: *STUDENT*

Has-inverse-mapping-object-name: *Is-advisor-of*

Has-maximum-values: *DEFINE(1)*

Has-minimum-values: *DEFINE(0)*

Has-object-id: $#$&&&&&$

Has-object-name: Has-advisor

Has-inverse-mapping-object-name: Is-advisor-of

DONE

Figure 5-8: Create the composite object **Has-advisor**

Create-a-composite-object

Has-object-name: *John*

Is-a-member-of: *STUDENT*

Has-phone#: *743-1234, 743-5678*

Has-status:

Has-department: *Pick-r(Mary, Has-department, ?)*

Has-object-id: ##$$&

Has-object-name: **John**

DONE

Figure 5-9: Create the composite object **John**

5.2.2. Destroy

This command is similar to the DELETE operation described in Section 3.3.5. Destroy simply removes an object and all relationships that it participates in. Figure 5-10 illustrates the system generated template for destroying an object. The information about an atomic object **1080 Marine Ave.** is filled in this template.

```
Destroy
```

Has-object-ref: *1080 Marine Ave.*

DONE

Figure 5-10: Destroy the object **1080 Marine Ave.**

5.2.3. Connect

This command is similar to the RELATE operation described in Section 3.3.3. Connect simply generates a relationship among the specified objects and adds it to the database. Figure 5-11 illustrates a template for generating a connection between objects. The information about (John, Has-status, DEFINE(graduate)) is filled in this template.

5.2.4. Disconnect

This command is similar to the UNRELATE operation described in Section 3.3.4. Disconnect simply destroys the specified relationship from the database. The system generated template for Disconnect is similar to the one for Connect shown in Figure 5-11.

Connect

Has-domain-object-ref[specify exactly one object-ref]: *John*

Has-range-object-ref[specify exactly one object-ref]: *Has-status*

Has-mapping-object-ref[specify exactly one object-ref]:
DEFINE(graduate)

DONE

Figure 5-11: Generate the connection
(John, Has-status, graduate)

5.2.5. Display

This command is similar to the DISPLAY operation described in Section 3.3.6. Display simply displays objects on the specified devices. Figure 5-12 illustrates the system generated template for displaying objects with the information to display the atomic object **admit** on the **printer** filled in.

5.2.6. Invoke

This command is similar to the EXECUTE operation described in Section 3.3.11. Invoke simply executes behavioral objects. Figure 5-13 illustrates the system generated template for invoking objects. The information for invoking the behavioral object **S-advisor**, defined in Section 3.3.11, is filled in this template.

```
Display
```

Has-object-ref: *admit*

Device-name: *Printer*

DONE

Figure 5-12: Display object **admit** on device **printer**

```
Invoke
```

Has-object-ref: *S-advisor*

Has-parameter: *Mary, Susan*

DONE

Figure 5-13: Invoke the behavioral object **S-advisor** with parameters **Mary** and **Susan**

5.2.7. Addname

This command defines new *object-names* for database objects. Figure 5-14 illustrates the system generated template to add object-names. The information for adding a new object-name for **John** is filled in this template. The user-defined object-name is also returned to indicate its acceptance by the system as a globally unique identifier.

```
Addname
_______

Has-object-ref: John

Has-object-name: Johnny

------------------------

Has-object-name: Johnny

DONE
```

Figure 5-14: Add a new object-name for **John**

5.2.8. Dropname

This command deletes object-names from databases. The system generated template for Dropname is similar to the one for Addname.

5.3. ISL User Interface Prototype

An ISL interface prototype is being developed on the IBM PC/XT with a half mega byte main memory, a standard IBM graphics display, and a mouse. The standard IBM color monitor works in one of the two modes of monochrome (black and white) or color. In the color mode the display resolution is 320*200 pixels, which does not allow much text on the screen. In the monochrome mode however, the resolution is 640*200 pixels, which allows more text. For the purpose of the ISL interface, we have sacrificed the use of color to get more resolution and use the monochrome mode. Obviously, with a better resolution color display we can improve the quality of the interface.

The interface is developed under the DOS operating system. A graphics package called HALO [Halo 84] and the MICROSOFT mouse [Microsoft 83] are used in this prototype. The prototype is coded in IBM PASCAL 2.0.

In the current version of the ISL user interface [Priddy 85] most of the commands in the *browsing-oriented database retriever* are implemented. A detailed description of how to implement the commands in the *menu-oriented database editor* is also provided in the current version of the ISL user interface manual [Priddy 85].

The prototype is menu-driven which means the input is most frequently an item selected from the menus that pop up on the screen. Very seldom is any input made from the keyboard. There is always a help message prompted on the screen to guide users on their next step.

5.3.1. Initial Screen

The initial display screen as shown in Figure 5-15 has two *menu-boxes* on the top, with two *roll-boxes* in each.

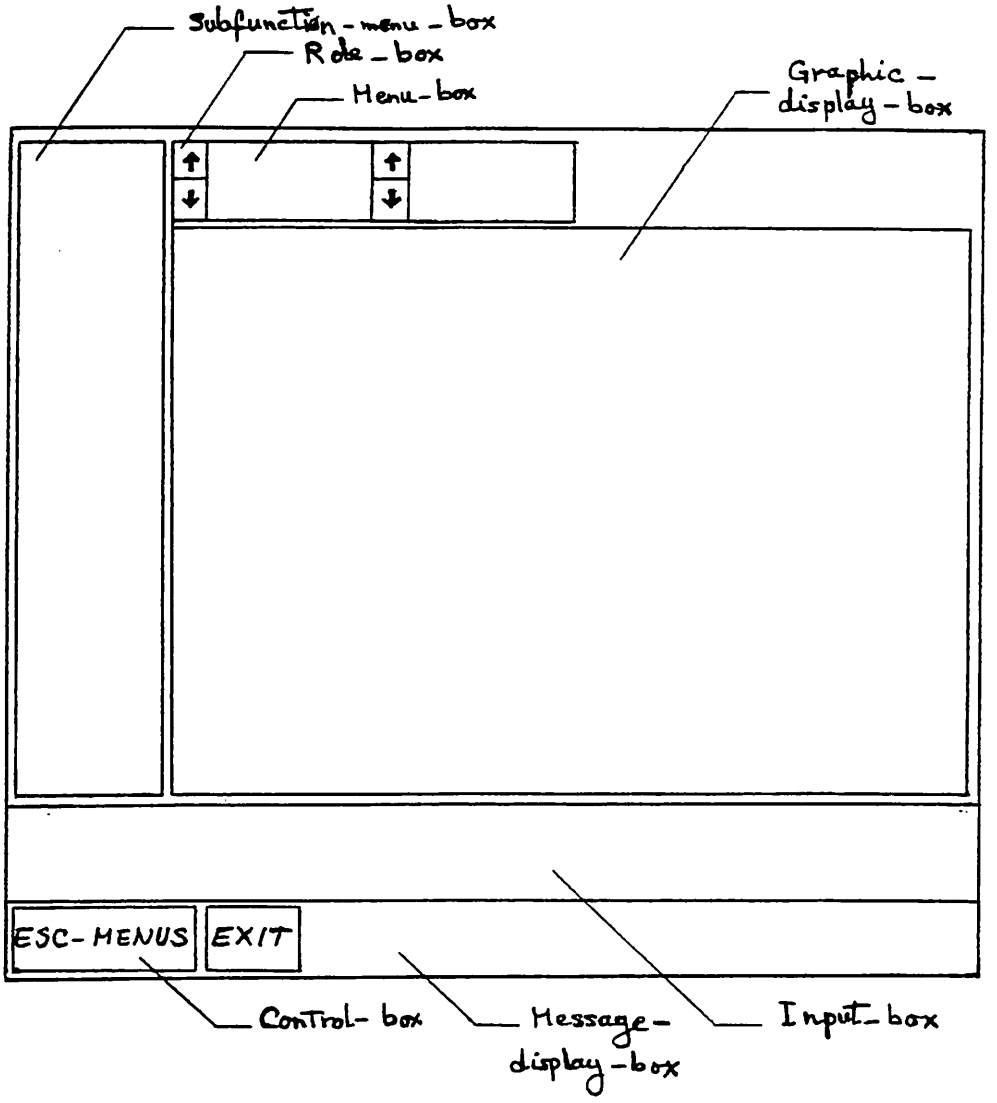


Figure 5-15: The initial display screen of the ISL prototype

The two menu-boxes correspond to the first level of menu commands in the

browsing-oriented database retriever, the **RETRIEVE** environment, and the menu-oriented database editor, the **EDIT** environment. Each menu-box always contains a command from one of these two environments or the names of these environments, **RETRIEVE** or **EDIT**, themselves. The two roll-boxes are used to move up and down on these menus in order to see other commands.

Selecting a command in a menu-box causes another menu to pop-up in the *subfunction-menu-box* on the left side of the screen. Selecting a command from the subfunction-menu-box can also cause another subfunction-menu to pop-up. Each level of the menu hierarchy specifies a function more completely.

At the bottom of the screen there are two *control-boxes* with the commands ESC-MENUS and EXIT. When ESC-MENUS is selected the system escapes the user from the hierarchy of subfunction-menus and returns him to the top-level of the program where a command in a menu-box can be chosen. When EXIT is selected the system exits from the ISL interface, writes the error file (if errors have occurred in the course of the program execution), and blanks out the screen. Help messages are always provided in the *Message-display-box* that explain what the interface expects the user to do next. *Graphics-display-box* is the display window that shows views of the geometric representation space. Any input entered though the keyboard is echoed in the *input-box* on the screen.

In the initial display screen, an *arrow* representing the location of the mouse appears at the center of the graphics-display-box. This arrow moves on

the screen as user moves the hand-controlled mouse by rolling it along a flat surface. In order to choose a function (or command) from a menu, the arrow (hereafter called the mouse) must be moved to the command. Commands that blink under the mouse represent valid choices and can be executed. The mouse button is depressed. If a command name does not blink under the mouse it cannot be executed. Commands in the two control-boxes, namely ESC-MENUS and EXIT can always be selected. When a user does not wish to select any operation in a subfunction-menu-box then he can do one of the followings:

1. Select the **Escape** command, which is always the last item of all subfunction-menu-boxes. This selection removes the current subfunction-menu and returns to the previous menu in the hierarchy.
2. Select the ESC-MENUS in the control-box. This selection returns the control to the top level menu-boxes.

Appendix A contains several print outs from the display screen while the ISL prototype was running on a personal database. In particular, Figure A-1 represents a **P-view** of the personal database. Figure A-2 represents the **Addview** of **Hamideh** to the **R-view** of **Chuck**. Figure A-3 represents the **L-view** of **2745**. Figure A-4 represents the **T-view** of **Has-phone#**. Figure A-5 represents a second **P-view** of this personal database, from which the user wishes to delete the object **Chuck**, and Figure A-6 shows the result of this deletion. Figure A-7 illustrates the same view as in Figure A-6, with the

Addview operation selected and its corresponding subfunction-menu-box displyed.

5.3.2. A Simple Scenario for Using the ISL Prototype

Suppose that a user wishes to see the right-view of a person **Chuck** on the screen. The user starts the interface and gets the initial display screen. The message "Select a command from a control-box" appears in the message-display-box. He moves the mouse inside the menu-box showing **RETRIEVE** to invoke the browsing-oriented database retriever. He moves the mouse inside the down-roll-box (showing an arrow pointing down). He depresses the button on the mouse and the next entry in this menu, **Navigate**, appears in the menu-box[4]. He selects **Navigate**. Then a menu pops up in the subfunction-menu-box with the three entries **Viewing**, **Moving**, and **Escape**. The message "Select a command from the subfunction-menu-box" appears in the message-display-box. The user selects the **Viewing**. Another menu replaces the one in the subfunction-menu-box with the following entries: **R-view**, **L-view**, **T-view**, **P-view**, **Scroll**, **Add-view**, **Del-view**, and **Escape**. The previous message remains in the message-display-box. The user selects **R-view**, and the message "Enter an object reference" appears in the message-display-box. The user enters "Chuck" through the keyboard which is echoed in the input-box.

---

[4]The only other entry in this menu is **Query**.

At this time the right view of Chuck will be displayed in the graphic-display-box.

5.3.3. The 3-Dimensional Linked List

In our study to develop the prototype implementation of the ISL user interface, a suitable physical organization was designed to support both the object-oriented nature of the 3DIS data model and the browsing-oriented nature of its user interface [Priddy 85]. The basic data structure used in this organization is a 3-Dimensional (3-D) linked list. This linked list simply represents the 3-D geometric representation of 3DIS databases and supports the access, query, manipulation, and modification of database objects. The 3-D linked list uses only a moderate amount of space per database object, supports efficient database access, browsing, and query, and automatically enforces some of the desirable constraints inherent in our orthogonal 3-D geometric representation space. However, using this data structure involves many linked list traversals and several inter-twined (two-sided) links must be created for object insertion.

The 3-D linked list consists of several interrelated doubly-linked lists. The organization of the 3-D linked list completely resembles the structure of the 3DIS geometric representation space. Every point in a 3DIS representation space is represented by one node in the 3-D linked list, where D, M, and R of that node denote the D, M, and R coordinates of the point it represents.

Consecutive nodes in a doubly-linked list differ in exactly one of these three coordinates. Every node participates in exactly three linked lists, as every point in a 3DIS representation space is located on exactly three lines parallel to the three axes.

Figure 5-16 represents the format of a node in these doubly-linked lists.

D	M	R
Next - D	Next - M	Next - R
Prev - D	Prev - M	Prev - R

Figure 5-16: The format of a list-node

Next-D, Next-M, and Next-R entries of a node are pointers to the next nodes in their corresponding lists, moving in the positive direction (away from the origin) parallel to the D, M, and R axes, respectively. Prev-D, Prev-M, and Prev-R entries of a node are pointers to the previous node in their corresponding lists.

Figure 5-17 represents an example 3-D linked list, where the number **3** denotes a student **John**, the number **4** denotes the mapping **Has-advisor**, the number **5** denotes the mapping **Has-committee-member**, and the number **6** denotes a faculty **Susan**. Note that in order to simplify this figure, each node contains only its D, M, and R information and its "Next" and "Prev" entries are not shown. However, the double-head arrows between the nodes in this figure represent their "Next" and "Prev" pointers. More detail on the functionality and properties of the 3-D linked list is described in [Priddy 85].

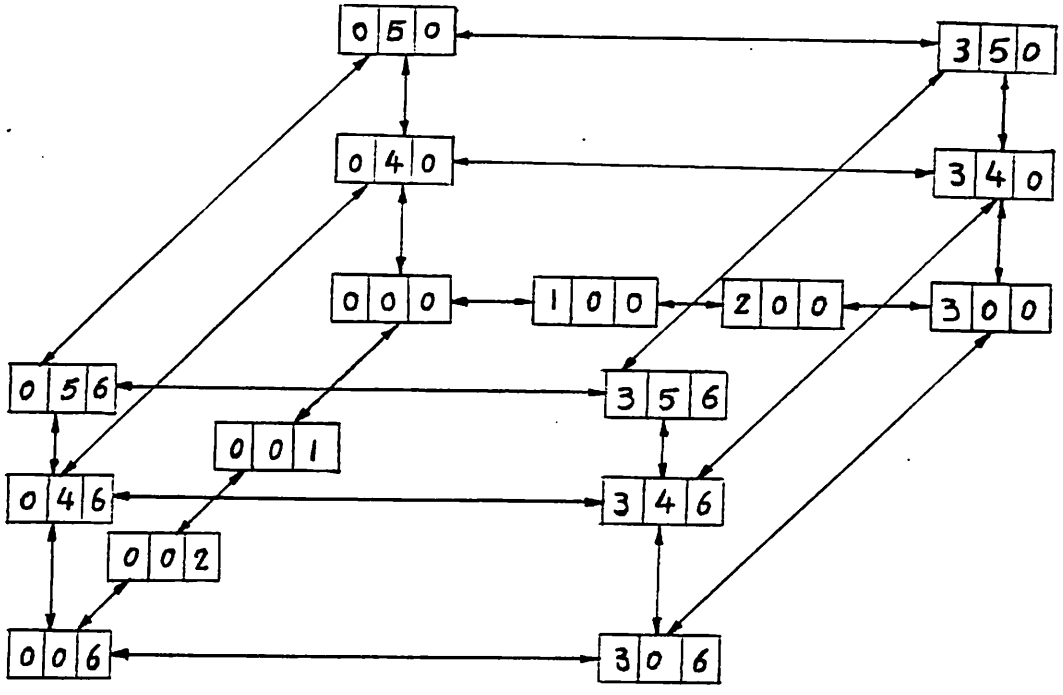


Figure 5-17: An example 3-D linked list

Chapter Six

A 3DIS Application: The ADAM VLSI Design Environment[1]

A VLSI design system developed under the ADAM (Advanced Design AutoMation) project at the USC Electrical Engineering Department was studied as an application environment for the 3DIS information model and its user interface. The outcome was an approach for specification and modeling of information associated with the design and evolution of VLSI components. This approach combines structural and behavioral descriptions of a component. In this chapter, database modeling requirements specific to the VLSI design are considered and techniques to address them are described. Necessary extensions were made to the 3DIS information management framework to support the modeling needs of this environment; e.g. to allow the recursive definition of VLSI cells. More details on our approach to information modeling for VLSI CAD applications is given elsewhere [Afsarmanesh 85c, Afsarmanesh 85a].

---

[1]The terms "model" and "subspace" used in this chapter and in Appendix B are not related to the concepts of model and subspace in a 3DIS database. "Model" is used as a generic term in this chapter, and "subspace" refers to a subset of a VLSI design space. See Section 6.4 for how concepts in VLSI design environments are mapped into constructs of a 3DIS database.

6.1. The VLSI Circuit Design Environment

The Very Large Scale Integrated circuit (VLSI) design environment is characterized by a large volume of data, with diverse modalities and complex data descriptions [Bushnell 83], [Davis 82], and [Knapp 85]. Both data and descriptions of data are dynamic, as is the underlying collection of design techniques and procedures. Design engineers, who are normally not database experts, nevertheless become the designers, manipulators, and evolvers of their databases. A final distinctive property of VLSI design environments is a requirement to model both the dynamic behavior of a circuit and its static structure.

The VLSI circuit design process typically begins with a descriptive high-level specification of the design, consisting primarily of dataflow and timing graphs, which together describe the data-transformation and timing behavior of the desired hardware. Less detailed structural (i.e., schematic) and physical specifications are given, describing static properties of the target circuit. The descriptive graphs are "hierarchical" in that their components can be recursively decomposed into simpler components. For example, a dataflow node **multiply** can be decomposed into simpler **shift** and **add** constructs.

Several relationships might be specified among the components of a high level design specification; e.g., among specific time intervals and data operations in the timing and dataflow graphs. Various constraints can be attached to the graphs; for example, the duration of a time interval can be limited, a schematic

wire can be specified to be a bidirectional bus connection, and the area of a physical bounding box can be limited. The descriptive graphical representations contain both numeric and symbolic attributes on their arcs and vertices.

The descriptive specification is usually large and complex. Many kinds of data are involved and it is in a large part recursively defined. Furthermore, the specification must be checked for completeness and consistency before the design process begins.

VLSI circuit design typically utilizes a design library, which contains components to be used in the construction of new components. It can also contain designs that are themselves under construction; these may be subparts of a larger design (e.g., the control unit for a CPU), or independent projects. Selecting the appropriate library component may be difficult. For example, if an adder is desired, there might be several components named **adder**, a few named **ALU**, and a few **complex standards** (i.e., microprocessors). In other words, the behavior desired may not match the stated behavior of any component in the library without some transformations.

The output of the design system includes a set of graphs, relationships, and constraints similar to those of the descriptive specification, but with a much more detailed physical description.

6.1.1. ADAM: A Unified System for VLSI Design

The ADAM system [Granacki 85] is envisioned to become a unified system for VLSI design, starting with a functional and timing specification and proceeding to circuit layout via automatic synthesis routines. The ADAM system describes VLSI circuits by means of four recursively defined and explicitly interrelated hierarchies. In ADAM, the representational formalisms of the input descriptive specification, the library components, and the output design are identical. This in turn facilitates the task of design verification and validation, e.g., testing the equivalence of specified and implemented dataflow graphs.

ADAM supports several major circuit design activities. These activities comprise the main part of the process by which the dataflow and timing descriptive specifications are mapped into the physical output components [Parker 84], [Director 81]. An appropriate information modeling environment for ADAM must support these tasks:

- *Algorithm Synthesis*: The dataflow graph is transformed in order to optimize speed, area, power, and other tradeoffs.
- *Partitioning*: Some part of the specification is partitioned so that the parts can be dealt with separately.
- *Floor Planning*: Given partitions and constraints, high-level chip plans can be constructed that aid in the prediction and optimization of area and performance.

- *Data Path and Control Synthesis*: Data paths are allocated hardware resources and the order of operations is fixed. Controllers are specified and synthesized. Interconnections are synthesized.
- *Built-In Test Synthesis*: Hardware is added to make the end product testable.
- *Module Selection*: Design library elements are introduced to implement operators, memories, and random logic.
- *Placement and Routing*: Modules are allocated physical positions on the layout, and interconnect wires are routed.
- *Validation and Verification*: At any step of the design process, performance and function may be validated using an appropriate simulator or formal verification tool.

6.1.2. Information Management Requirements

Given the above general characterization of the VLSI design process, the fundamental characteristics of digital VLSI design environments can be summarized as follows:

- The design data is of large volume and various modalities, e.g., graphical, symbolic, numeric, textual and formatted data.
- Structural information (e.g., data-description, data-interrelation, and data-classification) is complex, large, and dynamic. Structures must allow programs, documents, messages, constraints, and graphs to coexist.

- The end-users are design engineers and CAD application programmers who are familiar with their application environment, but are not likely to have expertise in databases or programming.

6.2. Information Modeling for VLSI CAD

Much of the reported work in the VLSI database design literature describes management of design information as collections of raw data in files. Interpretation of the stored design data is completely hidden in the application programs and the users' minds. These database systems are costly to maintain and evolve. Record-oriented database models, such as the relational data model, have also been applied to VLSI CAD design environments [Wong 79], [Eastman 80]. However, these models are not very suitable for non-database-expert VLSI designers who intend to build, use, and maintain their own databases. Recently, the suitability of *semantic database models* and *object-oriented database models* as tools to help in the construction and use of design databases has been examined [Katz 82], [McLeod 83], [Batory 84], and [Dittrich 85].

6.3. A 3DIS Database for VLSI

The digital circuit design process can be regarded as a search of a "design space" [Director 81] for a particular solution that meets constraints on functionality, timing, power, cost, etc. The entire design space can be broken down into subspaces which are "near-orthogonal" in the sense that decisions

taken in one subspace affect decisions taken in another subspace weakly across some region of interest. For example, a single functional specification can be mapped into several different implementations with varying speeds, power dissipations, and costs.

The 3DIS database described below is based on four hierarchical subspaces (also called *models*) chosen for their near-orthogonality [Knapp 83a] and [Knapp 85]. For example, one of the subspaces is used to represent schematic (structural) information; this subspace is a hierarchy with block diagrams at the top, registers and ALUs at the middle, gates at a lower level, and transistors at the bottom. Design entities are described in terms of these subspaces and a set of relationships across them.

6.3.1. The Definition of a Component

The fundamental structure of the ADAM 3DIS database is a *component*. A component represents either a specification, a design in progress, or a member of the design library. A specification is represented as an incomplete component. The information that is present in the specification usually represents the operations the target design must perform and the constraints it must meet.

A design in progress is also an incomplete component. The design gradually becomes more and more complete, until it can be manufactured. In the initial stages of a design, the target component primarily contains dataflow

and timing information; in the later stages it contains more schematic information, and finally a physical layout. The original dataflow, timing, and schematic information are preserved for documentation and verification/validation purposes.

The design library is used to store both procured components (OEM components) and "In-house" components. An in-house component may be either complete or incomplete.

6.3.2. The Four Models of a Component

A component is described in terms of four *models* and a set of relationships (*bindings*) across the constituents of these models. The models correspond to the four subspaces of the design space. They are:

1. *The dataflow model* describes the data transformation operations performed by a component. Its primitives are *nodes* and *values*. Nodes represent data transformations; values represent data passed between nodes.
2. *The timing and sequencing model* describes time-domain and branching behavior of a component. Its primitives are *ranges* and *points*[2]. A range represents a time interval during which an operation can take place; points represent infinitesimal "events",

[2]These points (timing events) are not to be confused with the points in a 3DIS geometric representation space.

which are partially ordered because the ranges have signs as well as durations.

3. *The structural model* describes the schematic diagram of a component. Its primitives are *modules* and *carriers*. A module represents a schematic block, gate, transistor etc.; a carrier represents a schematic wire.
4. *The physical model* describes the layout, position, size, packaging and power dissipation of a component. Its primitive elements are *blocks* and *nets*, which represent layout cells and interconnect respectively.

For example, the OEM-component **74181**, which is a 4-bit TTL ALU slice, has a dataflow model with **add**, **subtract**, **AND**, and **OR** nodes, which represent its data transformations. This component has a timing-and-sequencing model that describes its propagation delays for various combinations of inputs. It has a schematic diagram that either consists of a box with connection points or a gate diagram. It also has a physical description that signifies that its package is a 14-pin DIP.

6.3.2.1. Hierarchy within the Subspaces

Each of the four models is hierarchically structured. For example, a dataflow node is either primitive or it is defined recursively in terms of other nodes and values. Similarly, a value is either primitive or defined recursively in terms of other values. Similar recursive definitions are used in all four hierarchies.

6.3.2.2. Models and Links in the Four Subspaces

The generic name *Model* is used for nodes, ranges, modules, and blocks. The dataflow model of a component is therefore a *Node*, which can be recursively decomposed into *Nodes* and *Values*. The generic name *Link* is used for values, points, carriers, and nets. These too can be decomposed, with the exception of points, which represent atomic events of infinitesimal duration.

6.3.2.3. Relationships across Subspaces

All relationships among models and links of different subspaces are explicitly represented by means of *bindings*. There are two basic types of bindings, which are general enough to cover all cases of interest:

1. *Operation bindings*, which relate dataflow elements to structural elements and time ranges.
2. *Realization bindings*, which relate structural elements to physical elements.

For example, an operation binding expresses the relationship between an **add** operation (dataflow), an ALU (structure), and the time interval during which it

happens. A realization binding represents the correspondence between a particular layout region and the ALU.

6.3.3. The Target, the Specification, and the Library

A design under construction is the *target*. The target is functionally equivalent to the *specification*; it is composed of primitive elements and members of the *design library*. Near the top of the hierarchies, the dataflow of the target might be syntactically identical to the dataflow of the specification, but at the low levels this is unlikely. For example, suppose the specification contains a multiplication node. The definition of multiplication can be regarded as a series of shifts and conditional additions. But under a given set of timing, power, and area constraints, the dataflow actually implemented might be radically different. Therefore, the specification and the target are considered to be two completely different components. In general the relationships among constituents of the target and the specification can be complex.

Furthermore, there may be substantial differences between the way in which a library component is used and its actual capabilities. In a target, an **addition** might be implemented via a general-purpose **ALU**. If no other operations are performed in that ALU, then the only node bound to it in the context of the target is the addition. But the physical ALU that is present in the design is capable of far more than that: it can also subtract and perform logical operations. This "unused behavior" can be found by examining the dataflow and timing models of the library component that represents the generic ALU.

Various kinds of verification and validation methods can be implemented using the three different representations of the target, the specification and the design library. For example, the method of "inductive assertion" can be used to verify that a multiplication in the specification is really decomposed properly in the target. Moreover, the unused behavior can be useful both in determining proper control signals and in verifying that the proper control signals are used.

6.4. An Example

Consider the design of a particular component, a two-bit binary adder, which can be represented as in Figure 6-1. First the structure that defines the component is discussed; then the dataflow model of the component is examined in detail. The timing, structural, and physical models of the component are given in Appendix B. Finally, the way in which bindings are used to unify the four subspaces is described.

6.4.1. The Component

The subtype/supertype (generalization) hierarchy of component definitions is shown in Figure 6-2, where boxes represent type objects, the arrows represent subtype/supertype relationships, and the undirected lines that come out of the boxes lead to mappings (properties) that describe members of the types. The type *Component* has properties that denote its name, four *Models* of the component, and two sets of *Bindings*.

There are two subtypes of *Component*: *OEM-Component* and

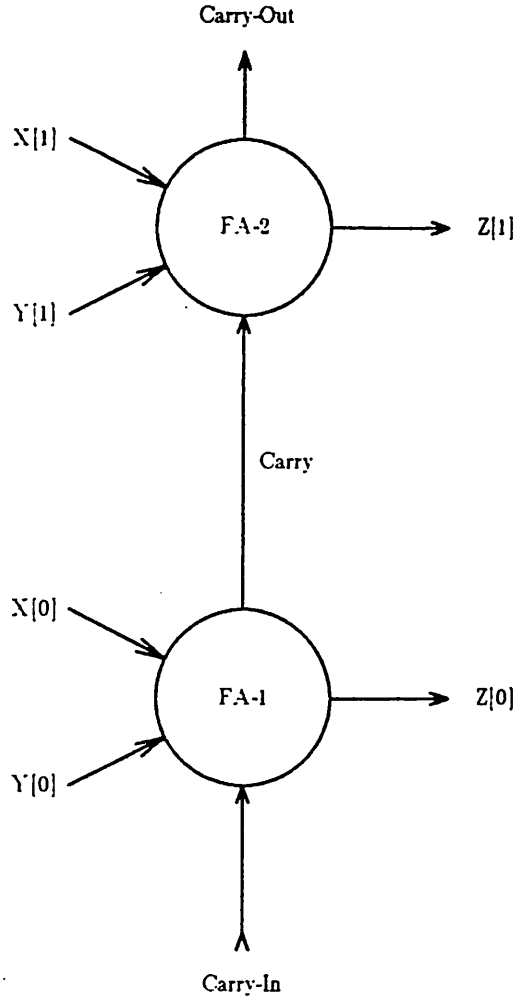


Figure 6-1: Two-bit adder example

In-House-Component. The *OEM-Component* represents a component supplied by an OEM (Outside Equipment Manufacturer). As such it is characterized by the name of the manufacturer, the manufacturer's designation (*Kind*), and a list of *Suppliers*. Other properties, such as *Price*, have been omitted from the figure in the interest of simplicity.

In-House-Component represents a component that is manufactured in-house. It may not even be a complete design; this information is captured by a

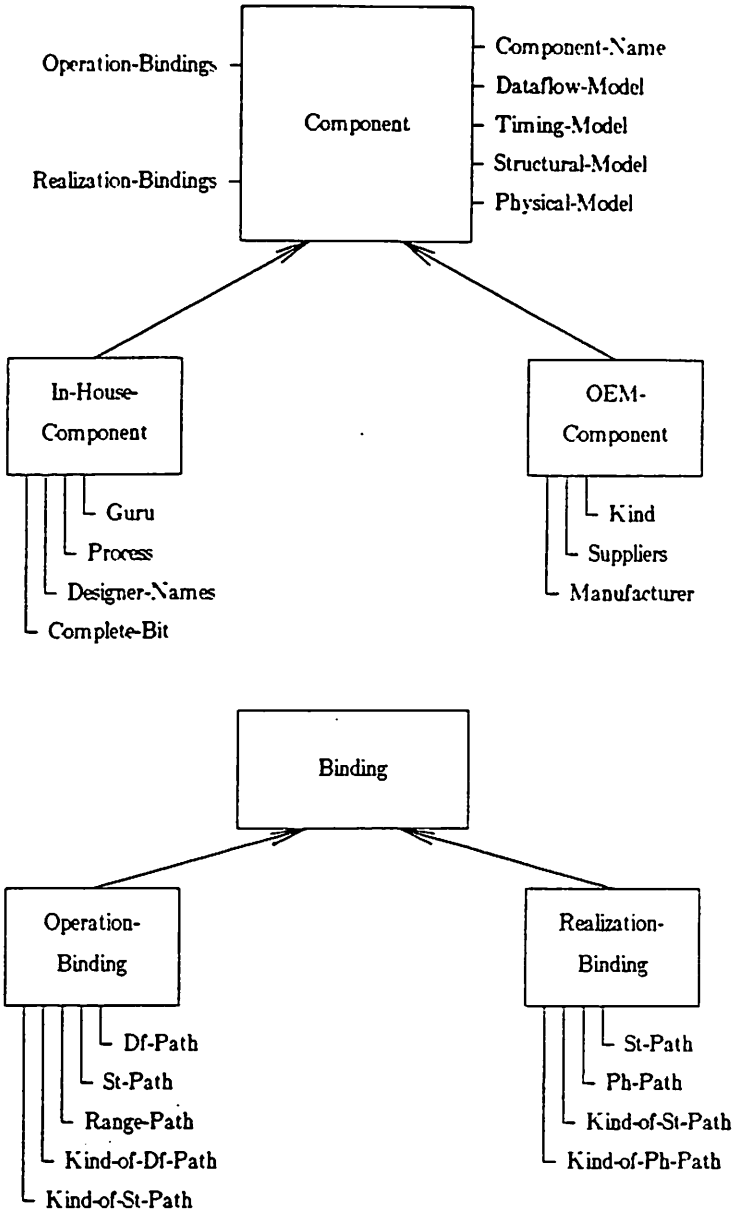


Figure 6-2: The subtype/supertype hierarchy of Components and Bindings

Complete-Bit[3]. An in-house component also has a set of *Designer-Names*, denoting the people responsible for its construction, a *Process*, which identifies a particular fabrication process, and a *Guru*.

[3]More complex historical information could be attached, e.g., a *Verification-History*.

A member of the *Component* type for our example is shown in Figure 6-3.

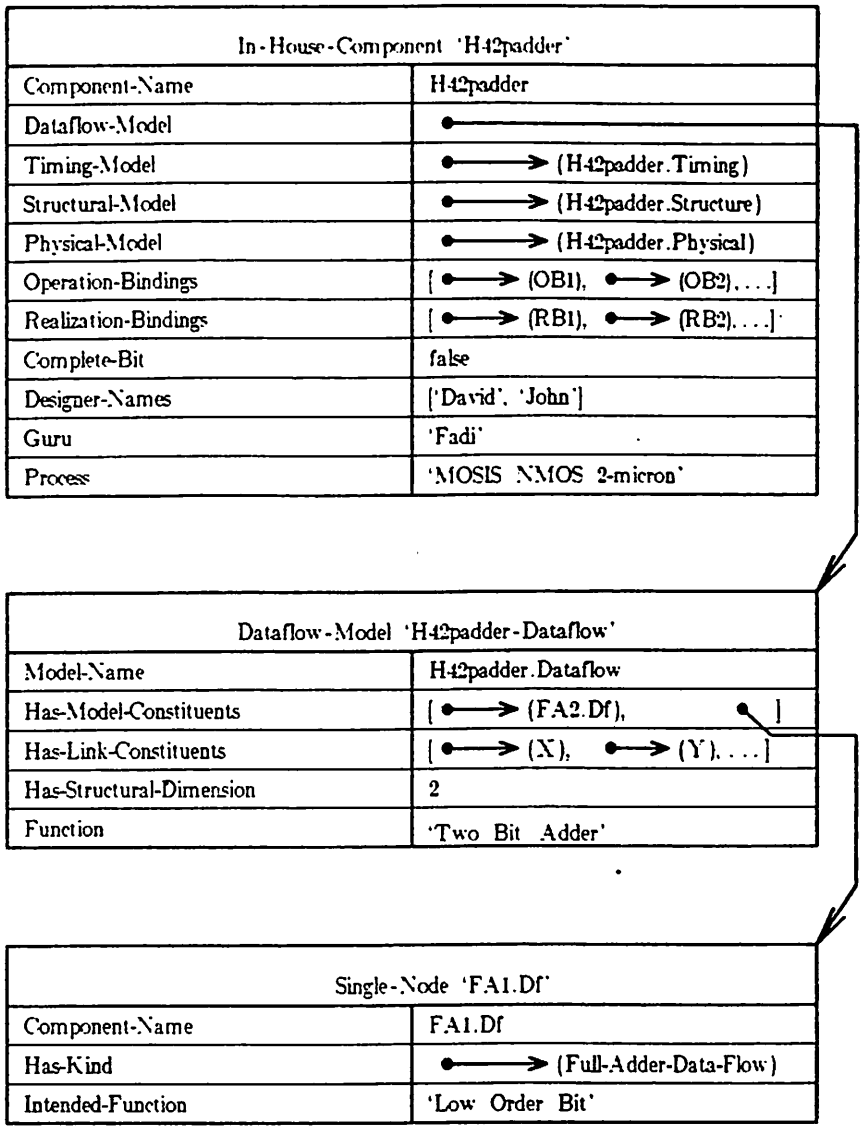


Figure 6-3: A Component member and its partial dataflow model

This *Component* is an *In-House-Component*. Its name, a property inherited from its supertype, is **H42padder**. The four *Models* are similarly named; Figure 6-3 shows only the **H42padder.Dataflow** *Model* in detail, where again

some mappings such as *Complete-Bit* and *Designer* have been omitted in the interest of simplicity[4]. *Operation-Bindings* and *Realization-Bindings* are also shown schematically as lists of logical references to the actual binding objects, as discussed in Section 4.3. The other properties of **H42padder** are self-explanatory. The dataflow graph of **H42padder** is given in Figure 6-1.

6.4.2. Models and Subspaces

The 3DIS model has been extended to accommodate the kinds of abstractions that are useful in VLSI design applications. In particular, for the ADAM design database, the 3DIS supports the recursive definition of VLSI components, as described in Section 3.2.3.1. This abstraction tool is applied to the recursive definition of the models and links of the four subspaces of components.

In this section we examine the dataflow subspace of our two-bit binary adder example to demonstrate the use of recursion abstraction primitives of the 3DIS for representation of its dataflow model. The other models are similar to the dataflow model. The differences are on minor properties. For example, the structural counterpart of a dataflow *value* is *carrier*. The carrier property *driver*, which describes hardware implementation properties such as **tri-state**, **open-drain**, etc., has no counterpart in the dataflow model. The interested reader is referred to Appendix B and [Knapp 85] for more detail.

---

[4] In Figures 6-3 and 6-8, the use of parentheses ( ) denotes objects whose details have been omitted for simplicity. Square brackets [ ] represent list delimiters.

6.4.2.1. The Dataflow Subspace and Dataflow Models

The subtype/supertype hierarchy for the *Dataflow-Model* is shown in Figure 6-4. Objects of type *Model* each have a name, a *Complete-Bit* similar to that of *Component*, and a *Designer*. There are four subtypes of *Model*, one for each subspace. Shown in Figure 6-4 is the subtype *Dataflow-Model*, also called *Node* for short. The other three subtypes of *Model* are *Structural-Model*, *Timing-Model*, and *Physical-Model*.

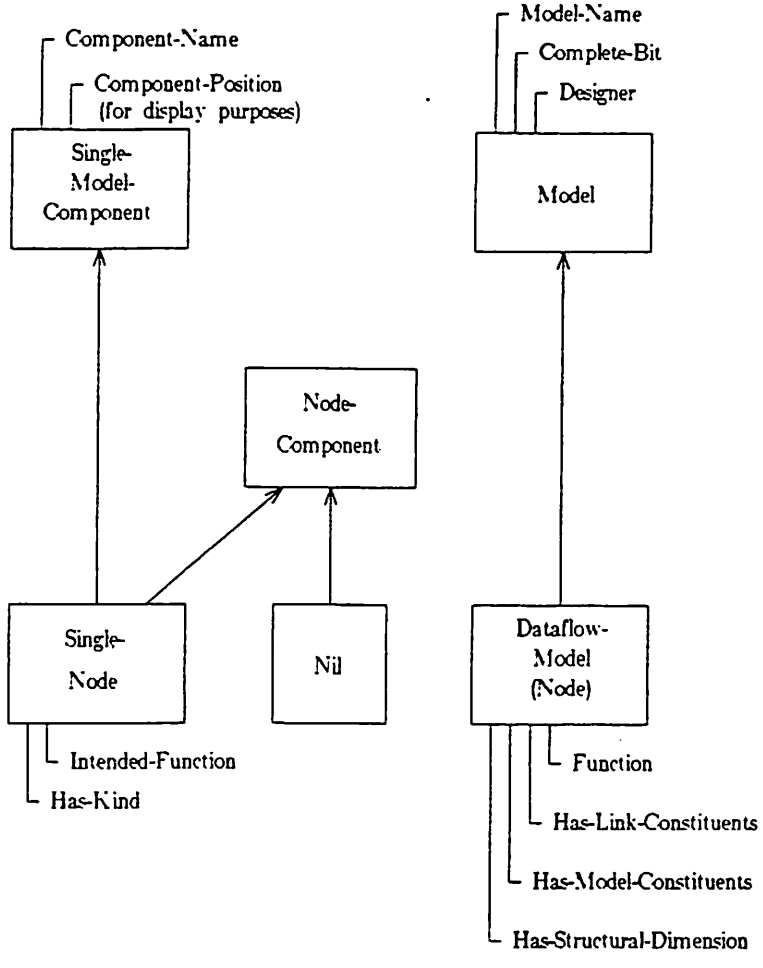


Figure 6-4: The subtype/supertype hierarchy of Dataflow Models

The generalization hierarchies for these three subspaces are shown in Figures B-1 and B-2 of Appendix B.

A *Dataflow-Model* has the following properties:

- The *Function* property indicates the overall function performed by the Node. For example, in Figure 6-3 the function of **H42padder-Dataflow** is "**Two Bit Adder**".
- The *Dimension* property specifies the bit width of the Node.
- The *Has-Link-Constituents* property indicates which links (for dataflow models, links are *Values*) are contained within the model.
- The *Has-Model-Constituents* property specifies which models (*Nodes*) are contained within the model.

The constituents of a model together express the application domain semantics of that model, thereby supporting its recursive definition. In the example of Figure 6-3, which corresponds to the two-bit adder dataflow graph of Figure 6-1, the link-constituents are the input, output and carry *Values*; and the model-constituents are the *Nodes* **FA-1** and **FA-2**. The constituents of a model are represented as lists of logical references.

The objects that are logically referred to in *Has-Model-Constituents* are of type *Node-Component*, which also designates that they are either of type *Single-Node* or *Nil*[5] If the reference is to *Nil*, then the constituent is not

---

[5]This is accomplished via the definition of the recursion abstraction described in Section 3.2.3.1.

further refined, i.e., the *Node* is either a primitive or its definition presently does not exist. In either case the recursive definition of the model ends at this point. If the reference is to a *Single-Node*, as is the case in the example, the recursive definition of the model continues through it. In the example, the *Single-Node*s are called **FA-1** and **FA-2**. **FA-1** has the *Intended-Function* **"Low Order Bit"**; presumably **FA-2** is the high order bit of the adder. Both **FA-1** and **FA-2** could have the value **"Full-Adder-Data-Flow"** in their *Has-Kind* properties; that means they are both one-bit full adder nodes. **"Full-Adder-Data-Flow"** is itself a *Node*, and is represented by a *Dataflow-Model*; hence it is further defined in terms of its model and link constituents. This is the recursion abstraction at work: *Models* are defined in terms of other *Models*.

Figure 6-5 illustrates a perspective view of the 3DIS geometric representation of a part of this example. In this figure, **FA-1** and **FA-2** are members of the type object *Single-Node*, while they are also the *Model-Constituents* of **H42padder-Dataflow**. Figure 6-6 illustrates the right view of the geometric representation for the **H42padder-Dataflow**. Both figures have been simplified to represent only the relevant part of the information in the database.

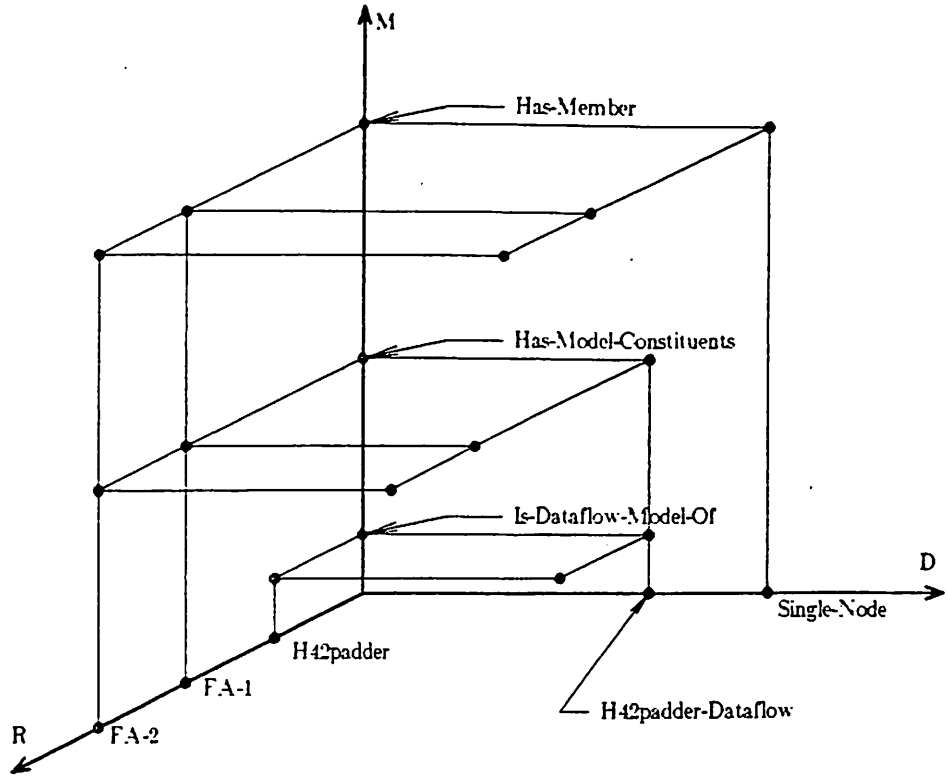


Figure 6-5: Perspective view of a 3DIS database

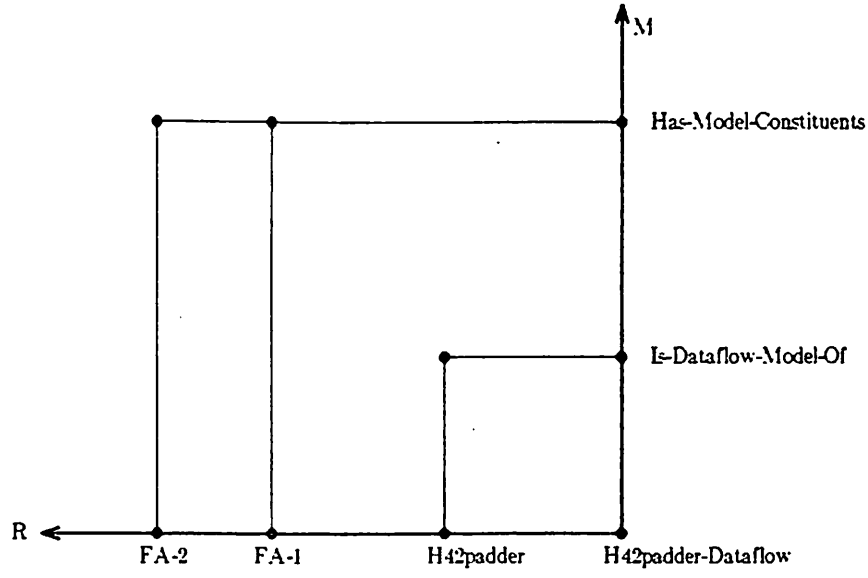


Figure 6-6: Right view of H42padder-Dataflow

6.4.2.2. Dataflow Links

Figure 6-7 shows the subtype/supertype hierarchy of *Links* for the dataflow subspace. *Links* are more complicated than *Models*, because they bear the burden of representing connections between *Models*. The generalization hierarchies for *Links* in all four subspaces are shown in Figures B-3 and B-4 in Appendix B. A Dataflow *Link* is called a *Value*. A *Value* has a *Name*, such as **Carry**, which is inherited from the supertype *Link*. It also inherits a *Complete-Bit* and a *Designer*, with meanings similar to those for the *Component*'s corresponding properties.

The reason a *Value* must have an explicit *Dimension* is that a *Value* is potentially a structured entity (for example a complex floating-point number). If a *Value* is a simple array, then the *Has-Structural-Dimension* property specifies the dimension of the array. If a *Value* is structured, then its *Has-Sublink-Constituents* property defines the structure. *Sublink-Constituents* are of type *Value-Component*, which also indicates that they are either of type *Nil*, or if they are of type *Single-Element*, it signifies that they are again either of type *Single-Value* or *Sub-Value* (Figure 6-7).

For example, a floating-point number **Flonum** is a structured value consisting of two fields **Mantissa** and **Exponent**. These are *Sub-Values*, which have *Has-Kind* properties of their own. The *Has-Kind* property of **Mantissa** might refer to a *Value* named **Long-Signed-Integer** and the *Has-Kind* property of **Exponent** might refer to **Excess-64-Integer**.

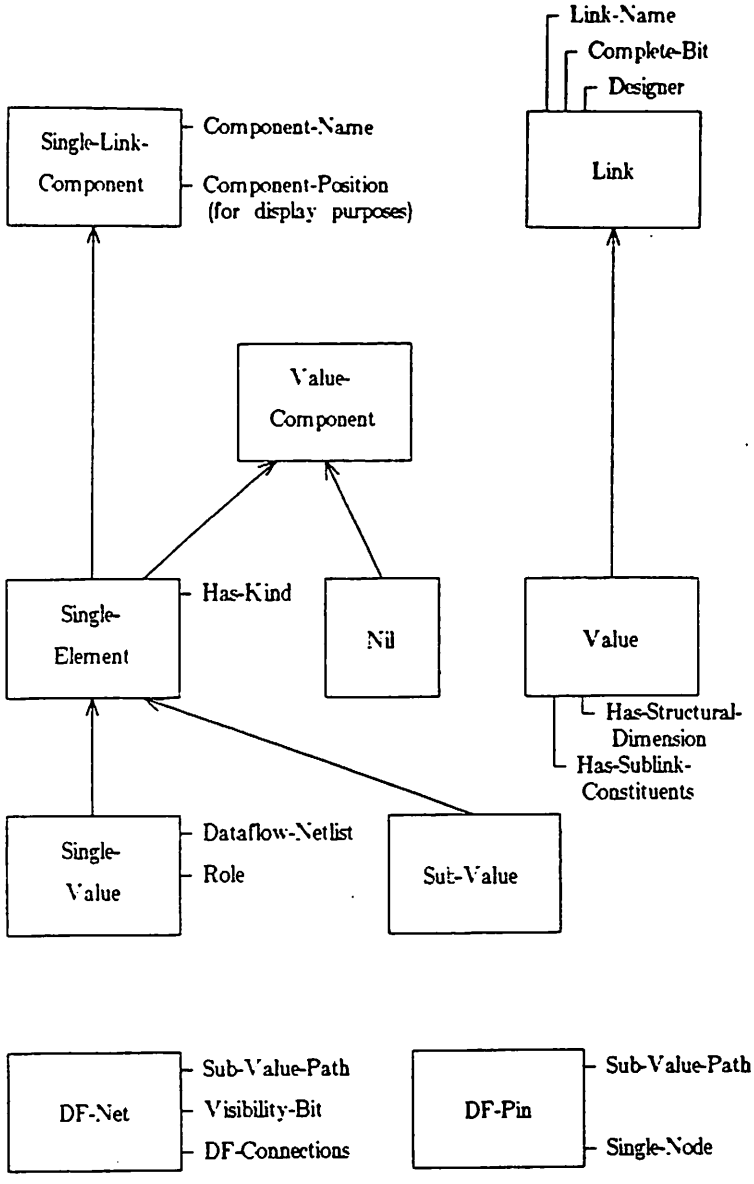


Figure 6-7: The subtype/supertype hierarchy of Dataflow Links

The input **X** of Figure 6-1 is a *Single-Value*. Figure 6-8 shows **X** in more detail. The *Has-Kind* property of **X** points to the *Value* **Two-Bit-Integer**. The *Value* **Two-Bit-Integer** has the *Structural-Dimension* **2**. **Two-Bit-Integer** also has *Sublink-Constituents* consisting of two *Sub-Values*,

named **High-Order-Bit** and **Low-Order-Bit** respectively. The *Has-Kind* properties of these bits have logical references to the primitive *Value* **Bit**. The *Has-Sublink-Constituents* of the *Value* **Bit** is *nil*, so the recursive definition of **Two-Bit-Integer** ends at this point.

In addition, **X** has a *Role* which is **second vector input**. Furthermore, it has connections, represented by a *Dataflow-Netlist*. The *Dataflow-Netlist* is a list of logical references to *DF-Nets*. In Figure 6-8, the two bits of the Two-Bit-Integer **X** are connected separately, only the "connections of the Low-Order-Bit" being shown as a *DF-Net*.

The *DF-Net* has a *Sub-Value-Path*. This is a path into the structure of the value being connected. For example, if the **high-order bit** of the **mantissa** of a complex floating-point number **A** was connected individually, the *Sub-Value-Path* would be "**A.Real.Mantissa.Bit63**". In Figure 6-8, the path "**X.Low-Order-Bit**" simply points to the low-order bit of **X**.

A *DF-Net* also has a *Visibility-Bit*; this determines whether the bit can be "seen" from outside **H42padder-Dataflow**. Since **X** is an input, this bit is **true** for all its *DF-Nets*. Other structured *Links* may have their visibilities determined on a field-by-field basis, which is why the visibility information is attached to the individual connections, e.g. *DF-Nets*, rather than to the *Single-Link*, e.g. *Single-Value*, itself.

DF-Connections are used to describe connections in the dataflow subspace. The *DF-Connections* of a *DF-Net* are references to *DF-Pins*. A

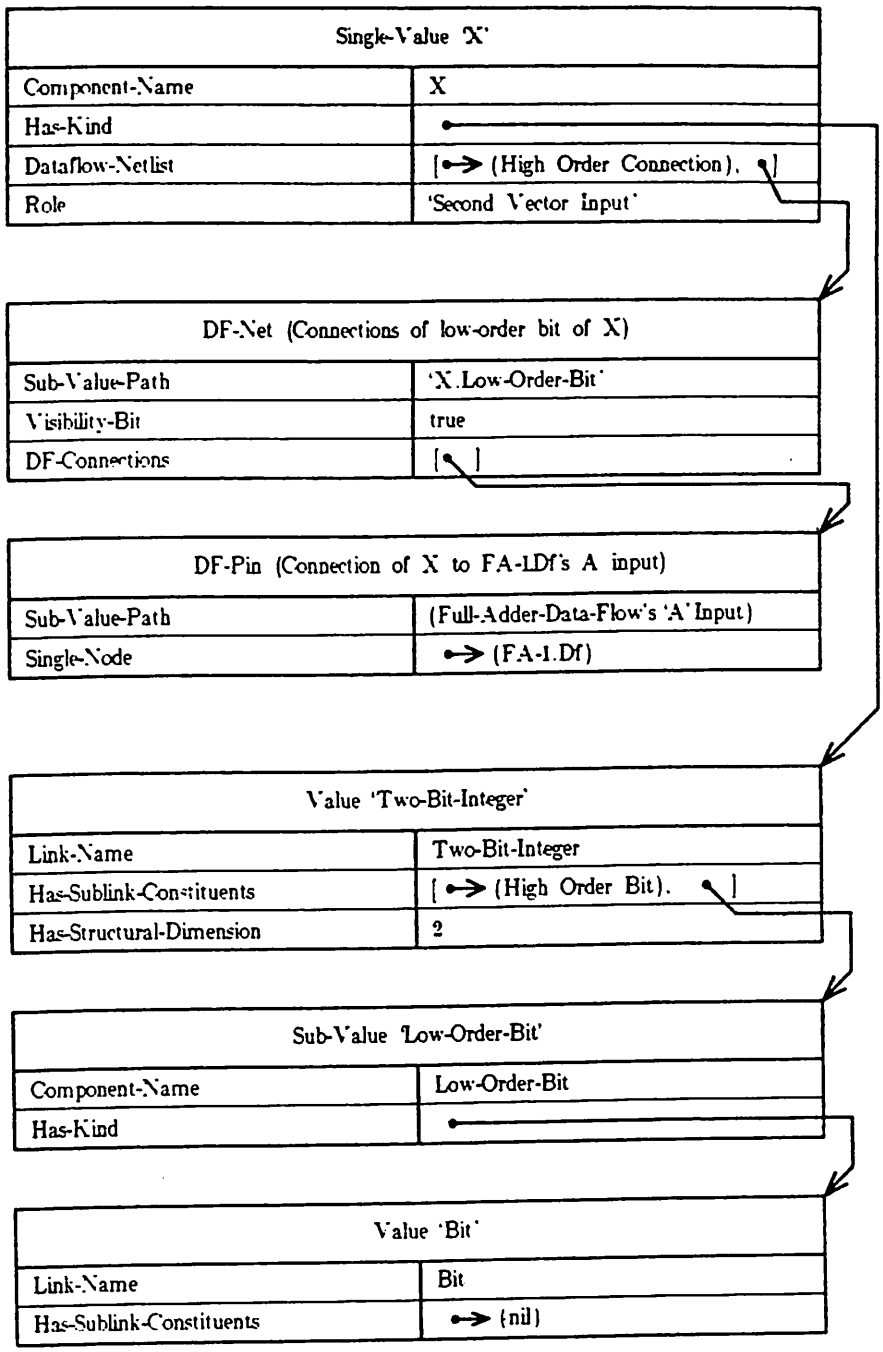


Figure 6-8: The definition and connections of the Value **X**

DF-Pin refers to a *Single-Node*, e.g. **FA-1.Df**, and a *Sub-Value-Path*, which represents a connection point on that *Single-Node*. In the example of Figure

6-8, the *Sub-Value-Path* of the **X** connection point is a path to the input bit **A** of the **Full-Adder-Data-Flow** model, which is given in parentheses in Figure 6-8 (recall that **Full-Adder-Data-Flow** is the *Has-Kind* property of **FA-1.Df**).

Using both the *Sub-Value-Path* of a link, as expressed in the *DF-Net*, and the *Sub-Value-Path* of a Single-Node connection point, as expressed in the *DF-Pin*, very general kinds of connections can be constructed. For example, using both paths in their full generality would allow us to make arbitrary permutations of structured array values at connection points. If a two-bit *Value* **P** was to be connected to the **X** input of **H42padder.Dataflow**, it would be possible to connect **P[1]** to **X[0]** and **P[0]** to **X[1]**, thus achieving a bitwise reversal at the point of connection.

6.4.3. Bindings

The two binding sets represent the interrelationships between the elements of the models. *Operation-Bindings* show the relationship between an operation (or value), a structure, and a time interval; (for example, an addition, an adder, and a microcycle). Similarly, a different Operation-Binding might represent the relationship between a value, a bus or register, and a microcycle. *Realization-Bindings* are used to represent the relationships between structural elements and physical realizations (for example, between an adder's schematic and its physical layout).

Both kinds of *Bindings* have properties that represent paths into the four hierarchies, e.g. **St-Path** as shown in Figure 6-2. The reason paths must be used is that *Bindings* refer to unique *Single-Model-Components*. Such a *Single-Model-Component* may be deep down in the recursion hierarchy, and the only way to uniquely specify it is by giving a complete path down into the hierarchy, starting at the root *Component*.

In *Operation-Bindings* the *Kind-of-Df-Path* property simply indicates whether the binding is to a *Node* or to a *Value*, by having the value of **type Node** or **type Value** respectively. similarly the *Kind-of-St-Path* property indicates whether the binding is to a *Carrier* or a *Module*. All combinations are permitted. These are examples of the use of the *generic interrelation abstraction* as described in Section 3.2.3.2. Similar considerations apply to properties of *Realization-Bindings*.

There is no *Kind-of-Range-Path* property for *Operation-Bindings* because the only valid timing element for a binding is a *Range* (a Timing Model). *Points* (Timing Links) have infinitesimal duration, and hence are never suitable for binding either operations or values to structural elements.

Chapter Seven

Conclusions and Future Research

This research presents an extensible object-oriented framework for database modeling, the 3DIS, integrated with a geometric representation of information, and a browsing, menu-oriented user interface, the ISL. The 3DIS provides simple constructs and tools appropriate for the specification and stepwise development of information systems with increasing complexity and different levels of abstraction. The 3DIS information management framework has been applied to a specific VLSI design environment to test its extensibility and modeling capabilities. Also, as a part of this research, an experimental prototype has been developed for the browsing-oriented portion of the ISL user interface.

7.1. Results and Contributions of this Research

3DIS databases are viewed as collections of objects and relationships among objects. Data, properties of data, descriptions/classifications of data (meta-data), operations on data (events), and constraints are all represented and treated uniformly and within a homogeneous object-oriented framework. Objects are the only fundamental constructs defined, where every identifiable

entity (real or conceptual) of an application environment corresponds to an object. Furthermore, the same operations are used to search and manipulate objects representing both data and meta-data. This enhances the dynamics of both structural and non-structural information. The primitive operations defined on the model are simple but functionally powerful to support object definition, manipulation, and retrieval.

The underlying approach in the 3DIS is to unify modeling, viewing, and treatment of database information. This is very different from the approach of most other semantic and conventional database models. Most such models clearly differentiate between modeling of data and representing the description of data (schema) [Tsichritzis 82], and contain several modeling constructs for different kinds of data. Therefore, even the most basic primitive operations may not apply to all such kinds of database information. This in turn, complicates design and manipulation of databases for the end-users.

This research has adopted the fundamental principles of semantic database modeling. In addition, a set of abstractions are defined that establish the basic relationships among objects and eliminate the need for prior knowledge about the structural information (meta-data) of databases in order to access them. The recursion-abstraction and generic-interrelation-abstraction introduced in the 3DIS are new tools that extend the modeling capabilities of this database framework. They support specific information management requirements of some recently growing class of design engineering applications.

The multi-purpose geometric representation for 3DIS databases described in this dissertation provides a wide range of information encapsulation and information organization capabilities. It also provides a semi-formal specification technique to formulate information encapsulation, and a formal definition for 3DIS modeling constructs in terms of their corresponding geometric components. The geometric representation is a suitable foundation for information browsing in a simple graphics-based user interface. Movements in this geometric framework serve as meaningful information access primitives.

The simple object-oriented user interface, ISL, described here gives a set of highly functional operations that support a browsing-oriented and a menu-oriented access environments. Since the 3DIS treats the meta-data as ordinary user data, the browsing-oriented environment is a uniform framework for browsing both data and meta-data. Most database browsers treat data and meta-data differently and many of them support browsing through data only.

The navigational operations of the browsing-oriented environment can be used to ask simple queries, such as checking the existence of a relationship between objects. The query operations of this environment have the simple format of ordered triples. These operations together with a small number of set-manipulation operations are powerful enough to formulate complex database queries. The editing environment of the user interface provides a small set of basic operations for object definition, manipulation, invokation, and display. The ISL user interface is simple enough for use by novice users and more efficient operations for experienced users are also supported.

A graphical interface for the browsing-oriented retrieving environment of ISL was implemented in PASCAL on the IBM PC/XT [Priddy 85]. This prototype is an experimental vehicle for evaluation and improvement of the browsing capabilities of the ISL user interface. This implementation includes the design of a suitable physical organization, a 3-Dimensional linked list, that supports both the object-oriented nature of the 3DIS data model and the browsing-oriented nature of its user interface.

The study of ADAM VLSI design environment resulted in an approach to specification and modeling of information needed for the design and evolution of VLSI components [Afsarmanesh 85c, Afsarmanesh 85a]. Some specific information management requirements of this application, such as the recursive definition of VLSI components, demanded introduction of certain new abstractions. These abstractions were added to the 3DIS information management framework. The example 3DIS database for the ADAM VLSI design environment described in this dissertation shows the 3DIS data modeling framework at work.

Significant benefits are expected from the presented approach in the development of the overall ADAM VLSI database system, some of which are listed below. Because the representation of design data is unified by the database, adding application packages with various kinds of data formats is greatly simplified. The database framework cleanly represents the data of interest. Important relationships between the design, specification, and the

target components are not obscured. Designers' freedom is limited to the degree permitted by the design specification. Finally, the design details are hidden from users until they are needed.

7.2. Directions for Future Research

Although a physical organization has been described for the 3DIS data model and a prototype has been developed for the browsing-oriented part of the ISL user interface, many features of this model and its interface are still un-implemented. The main concern of the prototype implementation has been the functionality rather than the speed or space. Issues of performance, concurrency, and access control are yet to be dealt with in an implementation. As a part of a joint project between the Computer Science and the Electrical Engineering departments, an implementation of the 3DIS for the ADAM VLSI design environment is underway at USC. The problems anticipated in the implementation of the 3DIS data model are largely efficiency considerations and the tradeoffs involved are the access versus the update speeds.

In order to improve the ISL user interface, future research can investigate the so-called "completeness" of its *query operations*. Another improvement to the ISL involves partial results of queries. Partial results to queries are obtained by the existing user interface prototype. But future investigation is necessary to determine the best way to return them to users to spare them the frustration of respecifying partially incorrect queries.

Further investigation to improve the spatial organization of the 3DIS geometric representation space may focus on how to use the *ordering* of and the *distance* between among database objects to convey certain semantic interdependencies. It may be possible to find such an arrangement technique for database objects that is generic enough to fit all applications. An investigation on the representation capabilities of the 3DIS geometric space might explore its relationship to the concept of universal relation.

There is a set of integrity constraints inherent in the 3DIS model that capture some of application semantics associated with objects and their interrelationships. Integrity constraints can also be defined as behavioral objects[1] of 3DIS databases. Users activate these objects whenever their related information (members of their related types) are inserted, deleted, or modified. But, the 3DIS model does not presently include a simple abstraction tool to formulate and specify generalized integrity constraints and daemons. Future investigation can provide a framework that allows defining integrity constraints in terms of the 3DIS modeling constructs.

Behavioral objects that model database dynamics are presently categorized as atomic objects. Users define behavioral objects in the form of procedures. These objects are treated as symbolic constants, so their information content is not interpreted by 3DIS databases. Further research on

---

[1]These are members of the predefined type CONSTRAINT-EVALUATORS as described in Sections 3.1.1, 3.2.1, and 3.2.2.

the specification, structure, and interpretation of behavioral objects can change their category from atomic objects to composite objects. As composite objects, behavioral objects will be defined in terms of other objects, allowing the 3DIS system to interpret their information contents meaningfully.

Finally, a future direction for research is to study the specific modeling concepts and tools required for distributed databases. Major issues include modeling and specification of object-scoping, object-equivalence, object-naming, and object-migration.

Appendix A

Sample
ISL Prototype Display Screens

This appendix contains several print outs of the display screen of a running version of the ISL prototype. There are a few minor differences between the format of the display screen in the running version and the version described in Section 5.3. One difference is that in the running version there are three control-boxes on the top of the screen for EDIT, QUERY, and NAVIGATION environments, instead of the two described in Section 5.3. Also, some of the names for menu-items used in the prototype are different than the description in Section 5.3. For details the reader is referred to [Priddy 85].

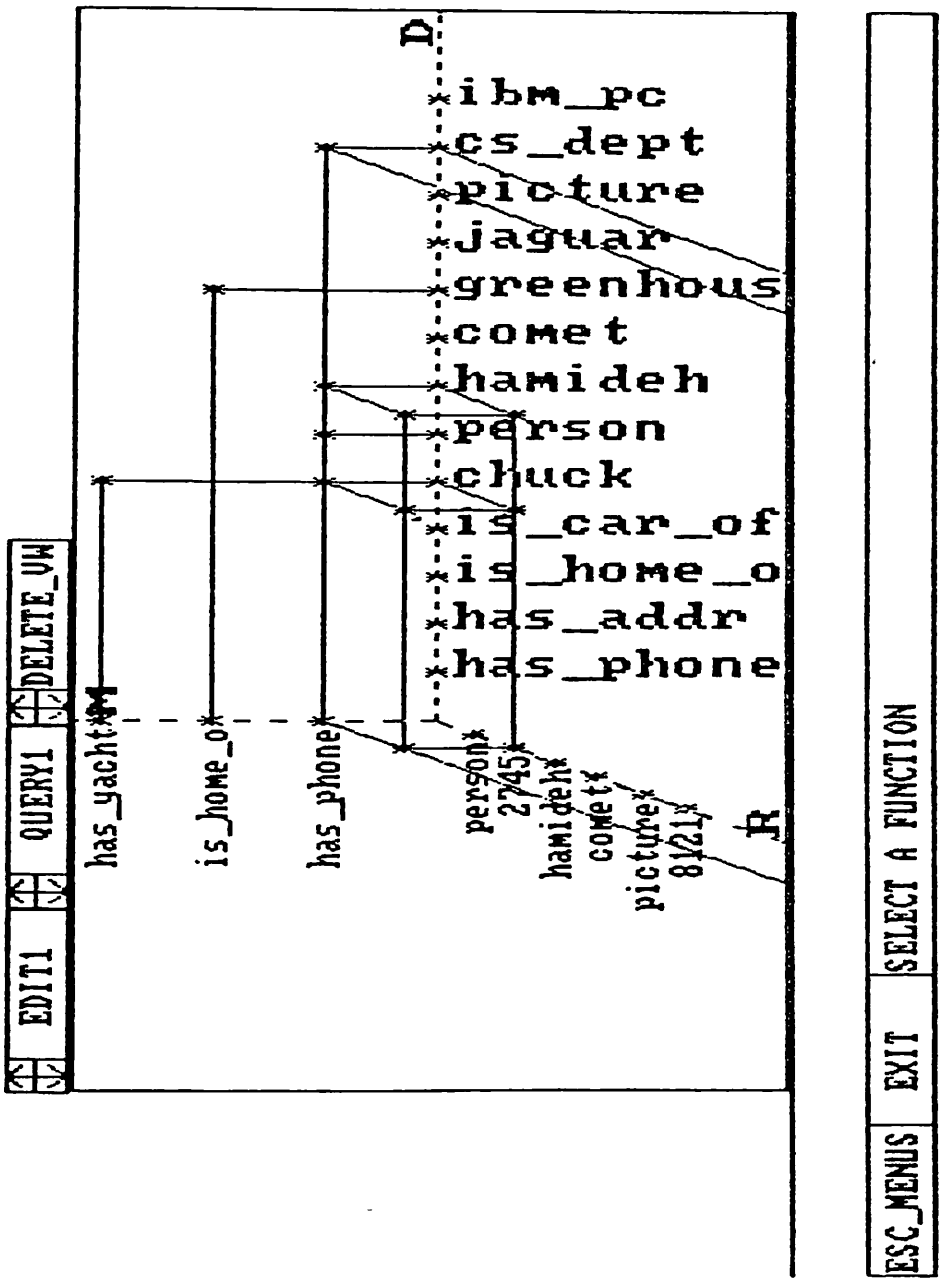


Figure A-1: A **P-view** of a personal database

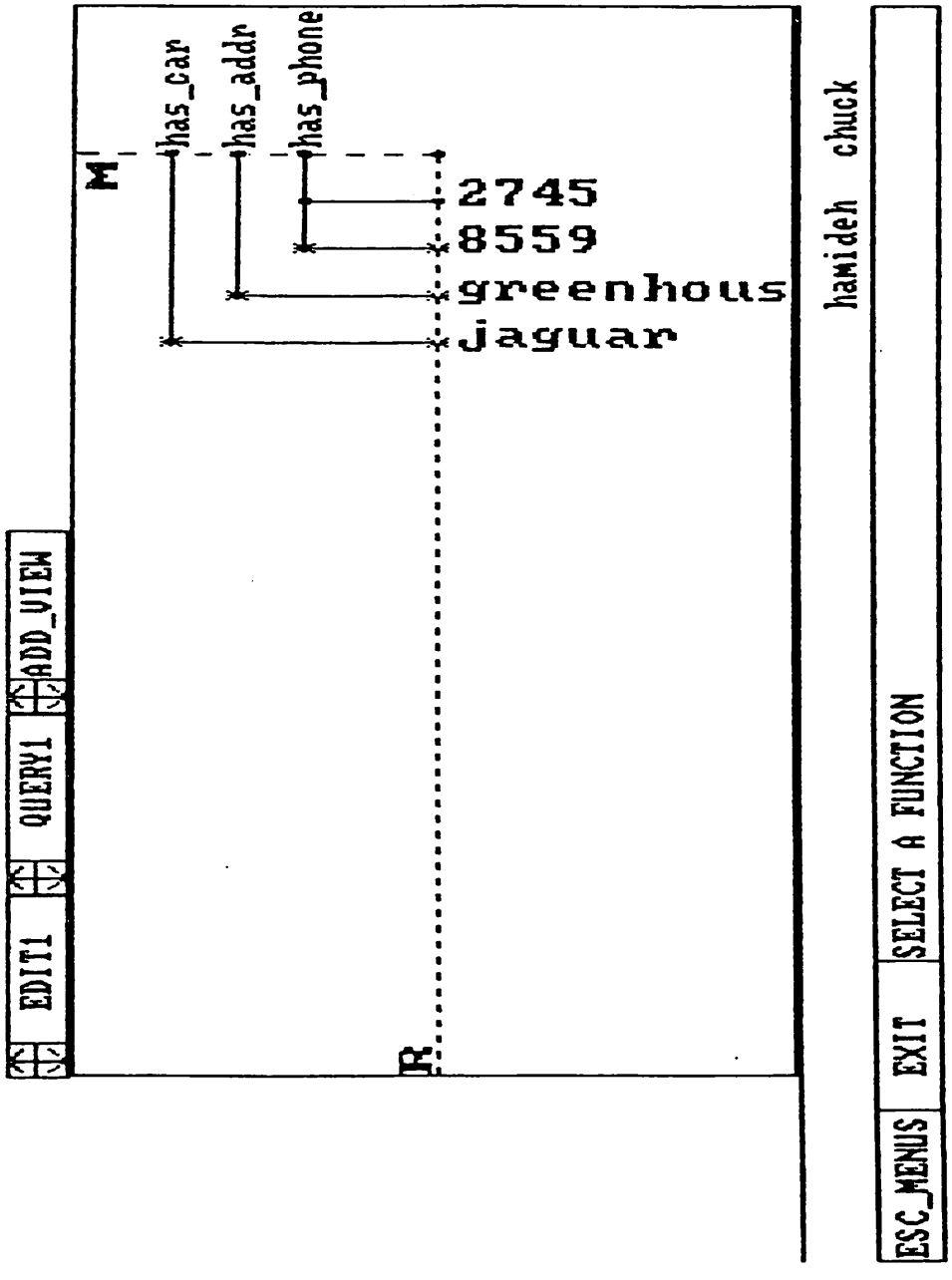


Figure A-2: Add-view of **Hamideh** to the **R-view** of **Chuck**

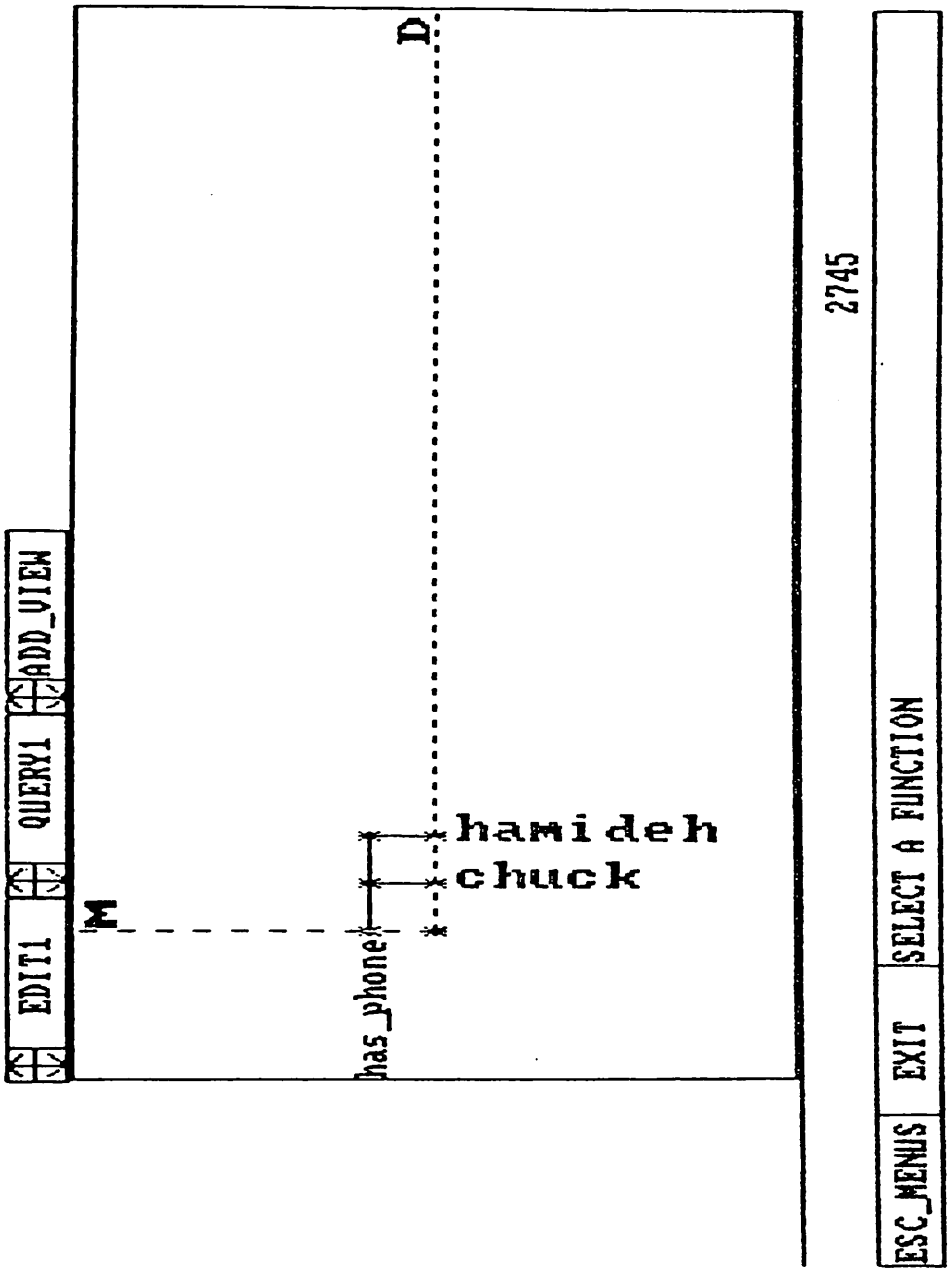


Figure A-3: L-view of 2745

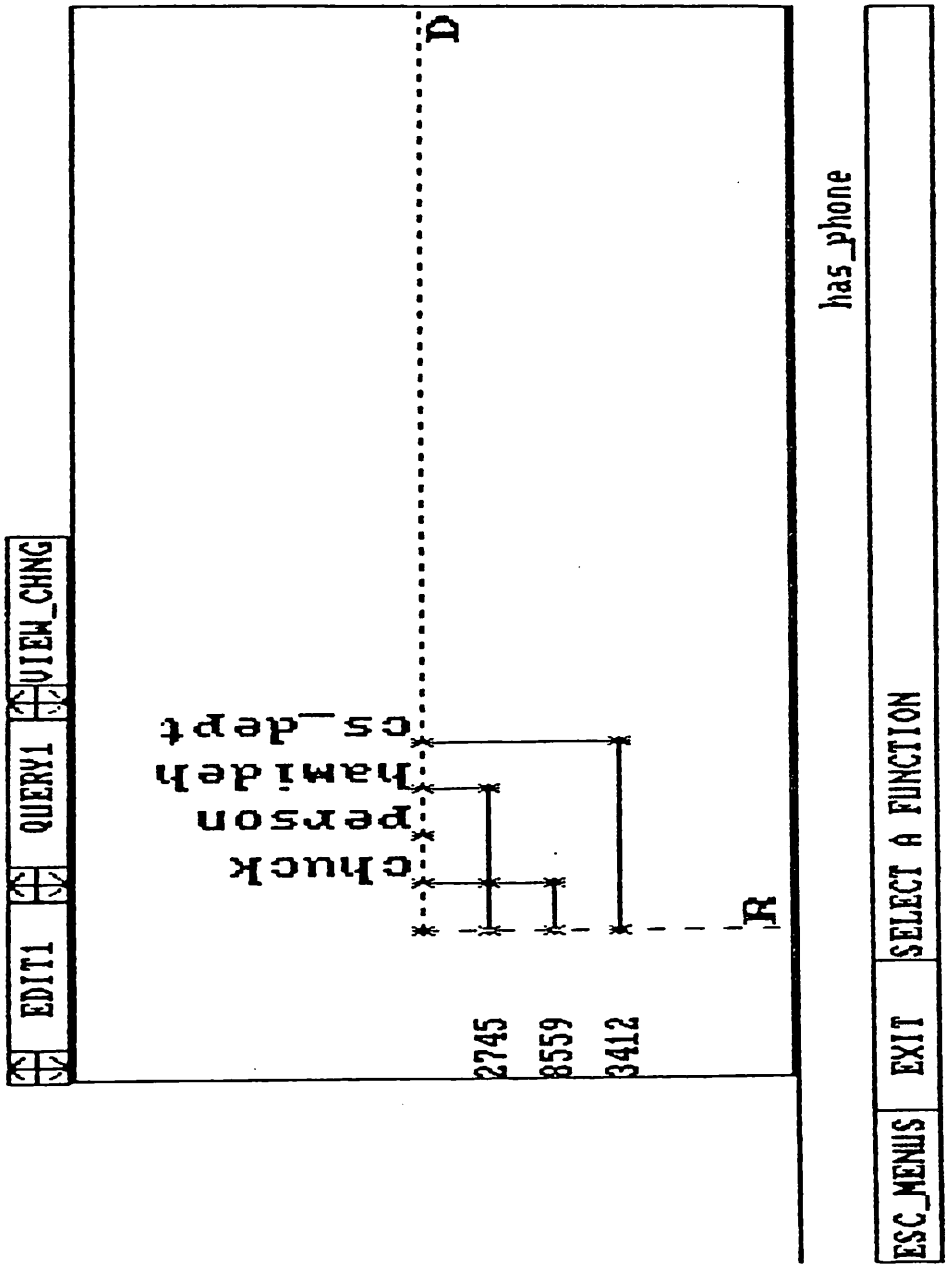


Figure A-4: T-view of Has-phone#

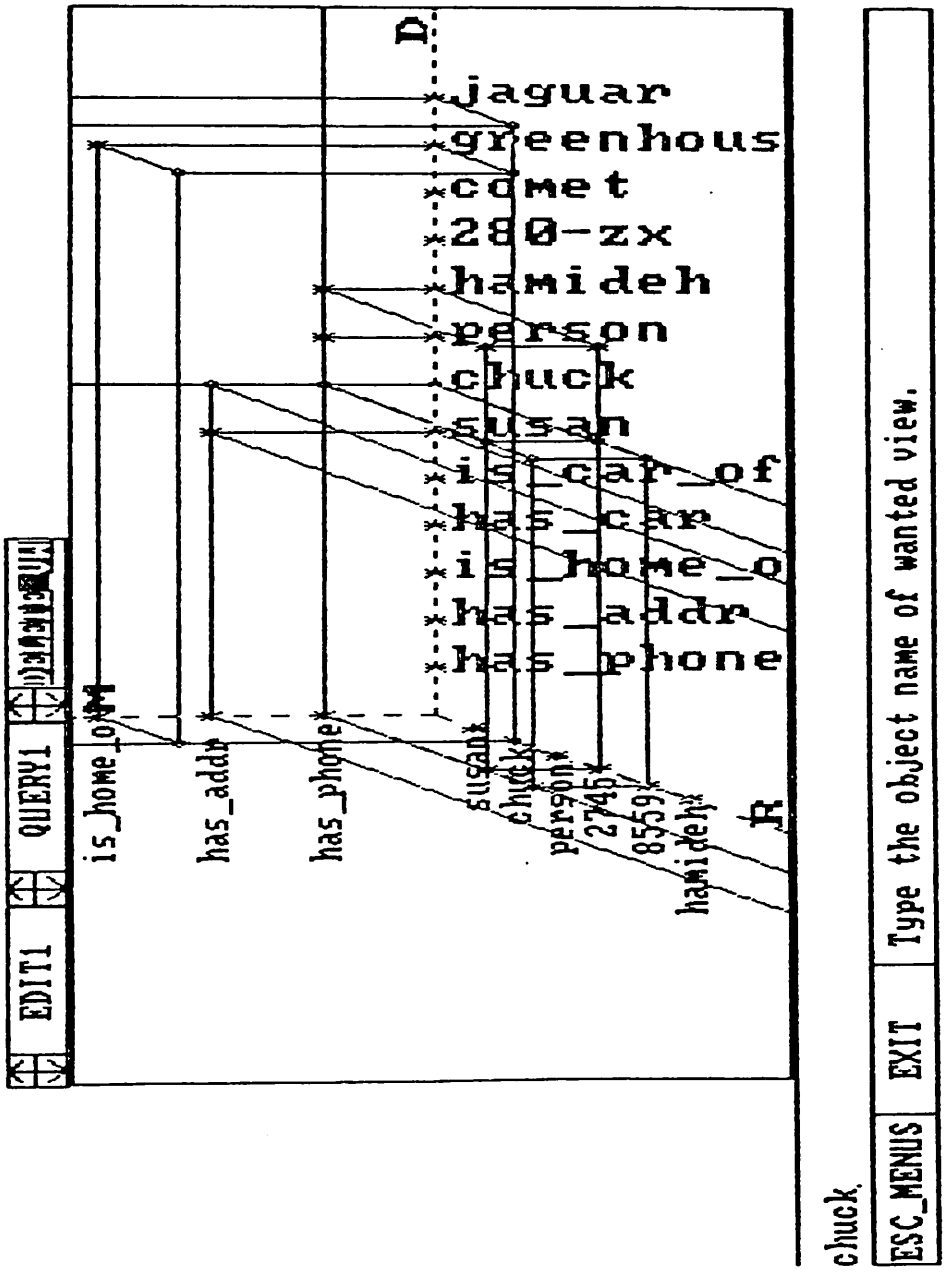


Figure A-5: A P-view of a personal database

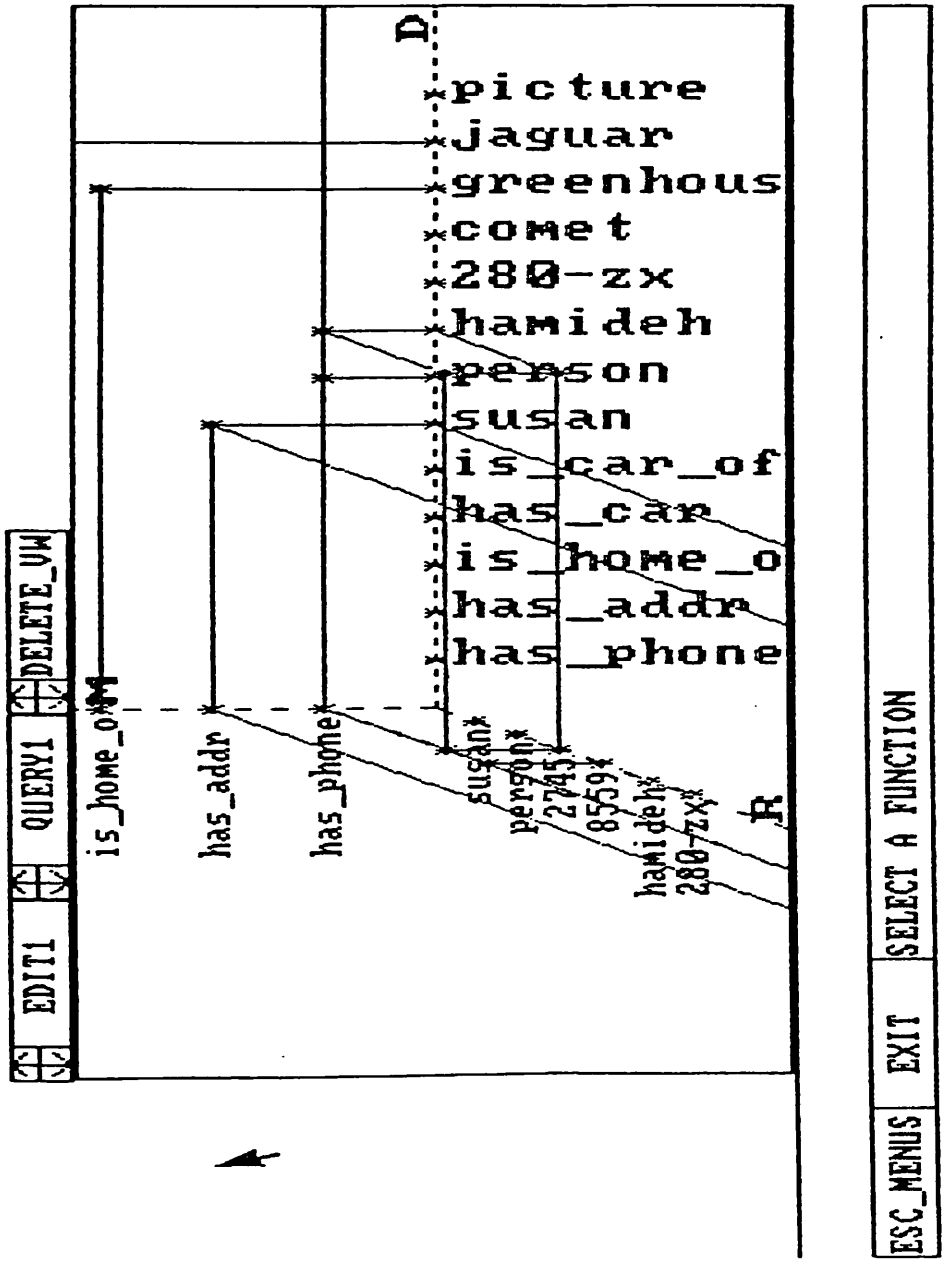


Figure A-6: The result of deleting **Chuck** from Figure A-5

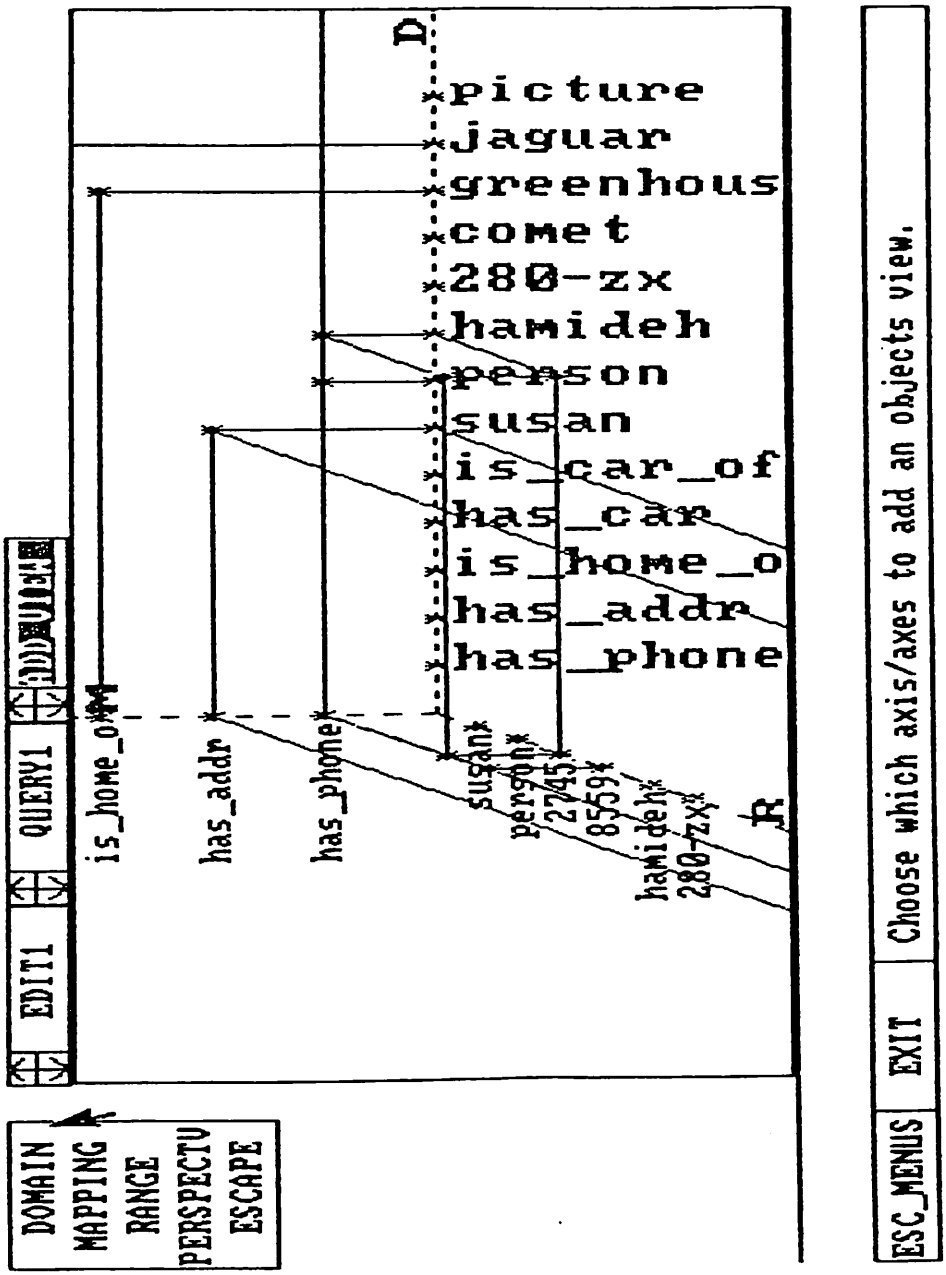


Figure A-7: A **P-view** of a personal database with **Addview** selected

Appendix B

Subtype/Supertype Hierarchies for the Four Subspaces of VLSI Components

This appendix gives a more global view of the subtype/supertype hierarchies. Some of these figures are repeated in Chapter 6; all figures are included here as a comprehensive reference. There are a few minor differences between the dataflow hierarchy, which was used as an example in Chapter 6, and the other hierarchies. Some major differences are discussed in this appendix, for more details the reader is referred to [Knapp 83b]. The following two conventions apply to the figures in this appendix. First, where range types for properties are not obvious, a colons (:) is used to indicate the type object over which a property is defined following the property name. Second, type objects with a star (*) superscript represent properties that define one-to-many relationships.

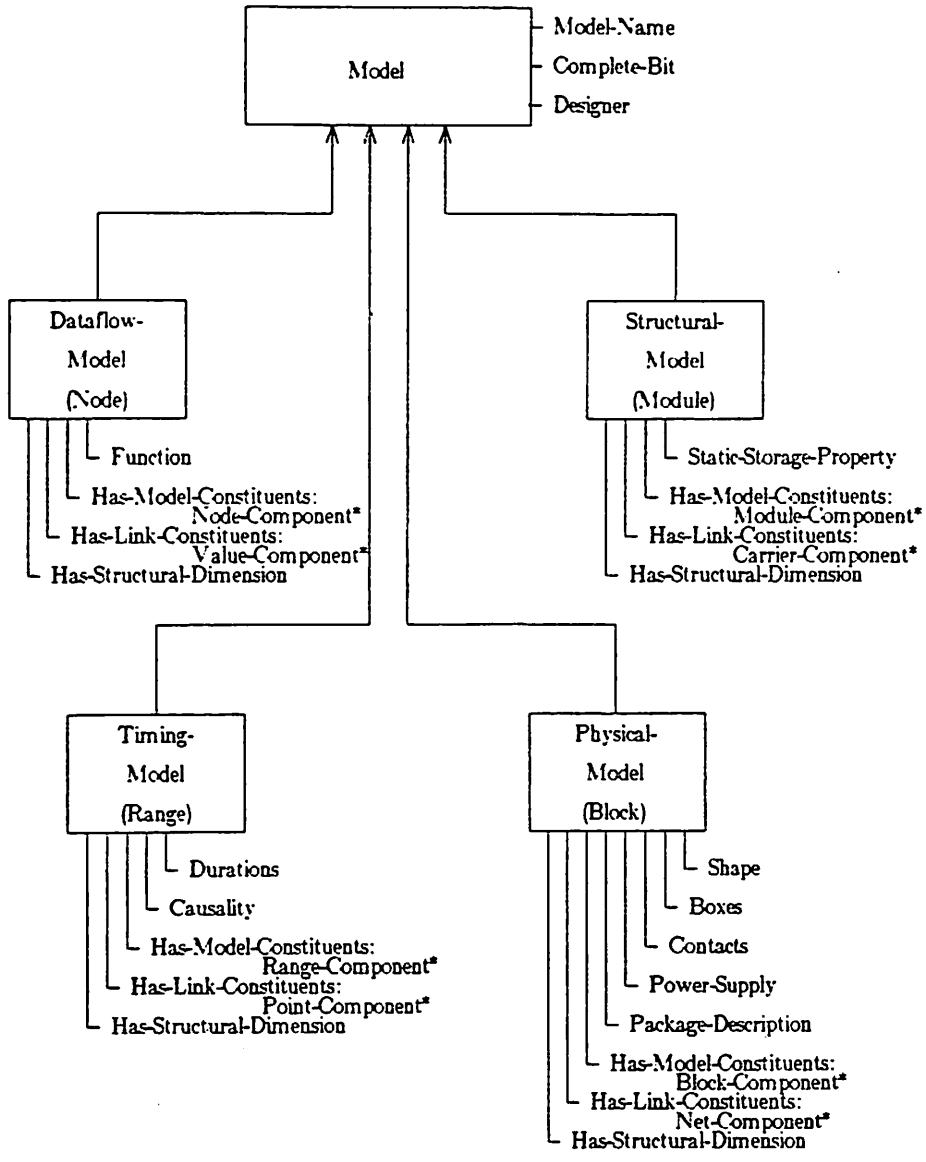


Figure B-1: The subtype/supertype hierarchy of Models

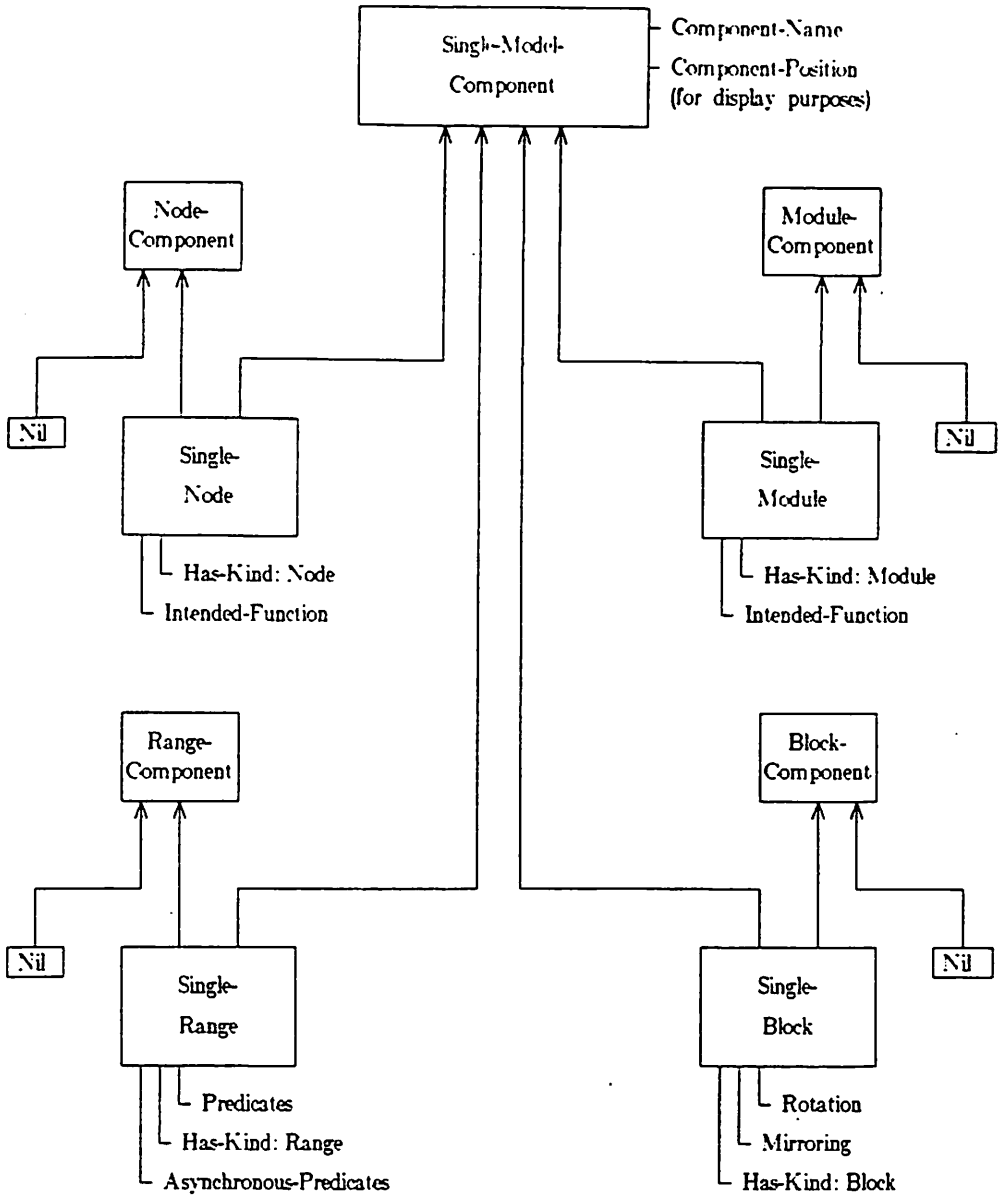


Figure B-2: The subtype/supertype hierarchy of Single-Models

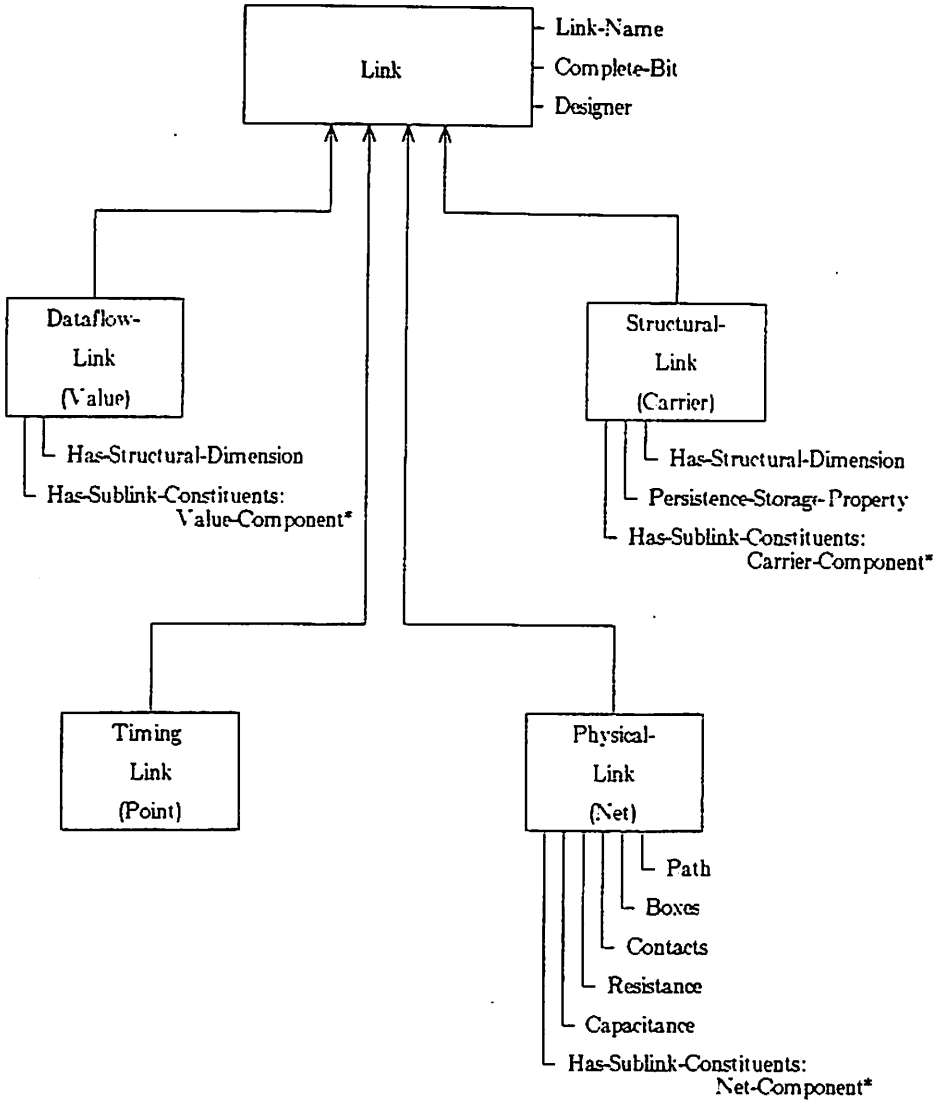


Figure B-3: The subtype/supertype hierarchy of Links

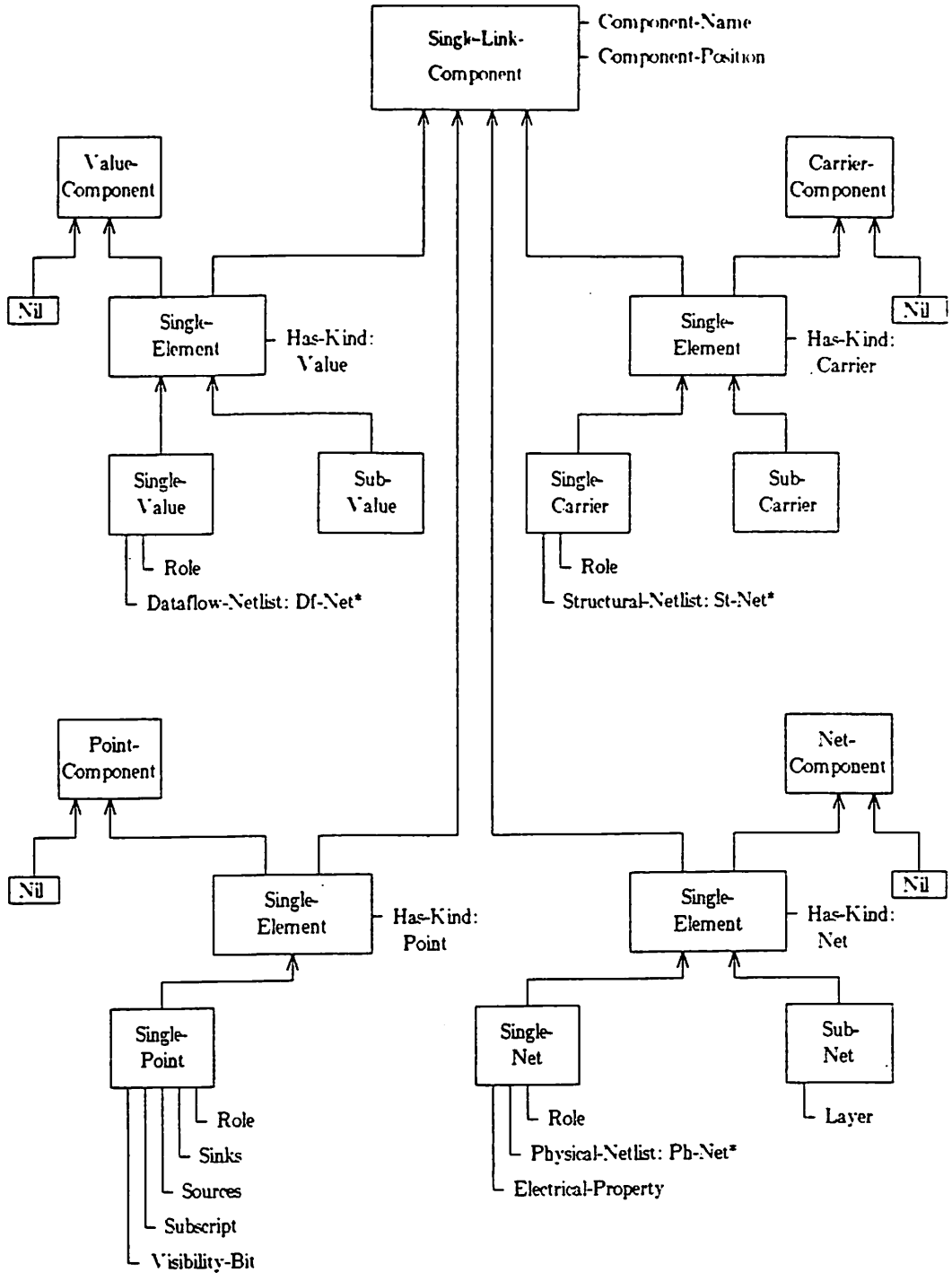


Figure B-4: The subtype/supertype hierarchy of Single-Links

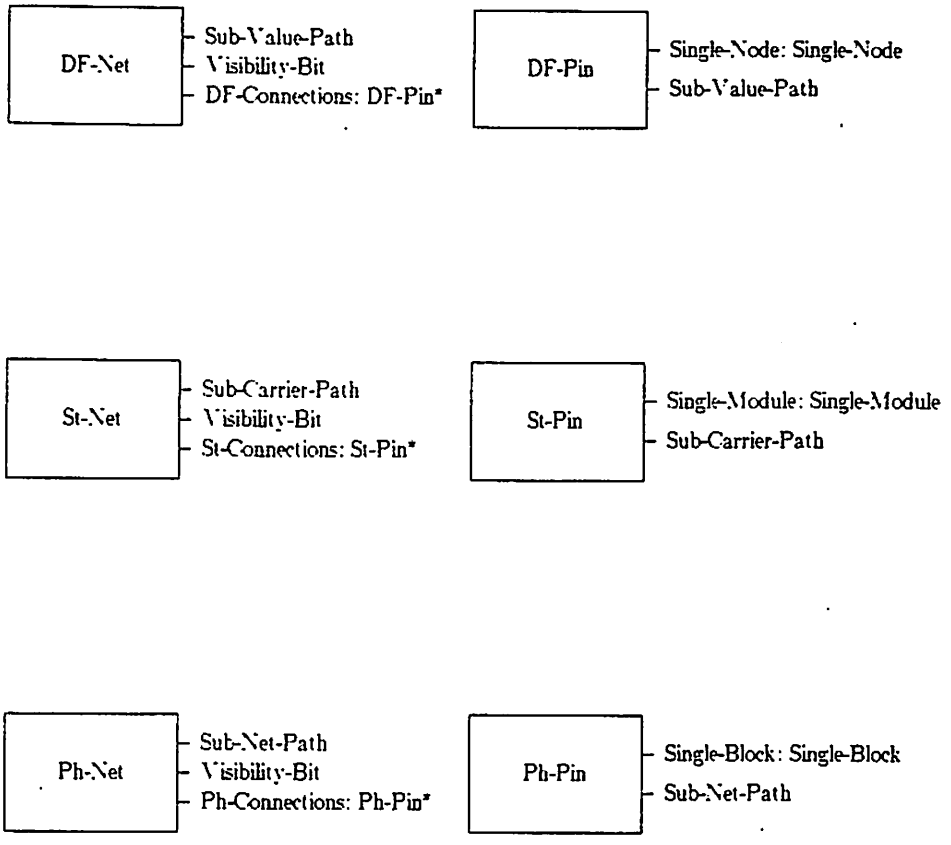


Figure B-5: Nets and Pins

B.1. Notes

In the example given in Chapter 6 we discussed the dataflow subspace in detail. The other subspaces are not structured in exactly the same way. Some major differences are described in the following notes:

- In the *Timing Subspace*:

1. *Timing-Models* have a set of *Durations*. These represent either constrained or achieved time intervals. Hence it would be possible to have both constraints and achievements listed for the same range.
2. *Timing-Models* also have a *Causality* property, which distinguishes constraining, measuring, and causative arcs from one another.
3. A *Single-Range* has the following predicate properties:
    a. *Predicates*: this property describes the conditions under which normal branching will occur.
    b. *Asynchronous-Predicates*: this property describes the conditions under which branching is not synchronized to a particular point in the timing graph, e.g. resets. The semantics of both kinds of predicates is described in detail in [Knapp 83a].
4. *Points* do not have any property. This is because they are of the following seven enumerated kinds, and their properties are implicit in these kinds:

a. *Simple* points have one in-arc and one out-arc. These points represent events.

b. *Alpha* points have one out-arc and no in-arcs. These points represent loop reentry points. The out-arc must have an indexing subscript, as the loop is considered to be a (possibly infinite) set of instantiations of the arc(s) between the alpha and the omega points.

c. *Omega* points have one in-arc and no out-arcs. These points represent loop backjump points.

d. *Or-fork* points have one in-arc and a number of out-arcs. They represent branch points. Each out-arc must have a predicate attached to it, describing the conditions under which the arc is taken.

e. *And-fork* points have one in-arc and a number of out-arcs. These represent "cobegin" constructs, i.e. points at which concurrent streams of events flow apart.

f. *Or-join* points have a number of in-arcs and a single out-arc. They represent points at which several disjoint execution paths merge.

g. *And-join* points have a number of in-arcs and a single out-arc. They represent coend points.

5. *Single-Points* have sink and source sets; these refer to *Single-Ranges* and express the connectivity of the timing graph.

• In the *Structural Subspace*:

1. The *Static-Storage-Property* of a *Module* represents the possibility that it might have static storage elements in it, i.e. memory. Hence this property would be **true** for a register and **false** for a gate.
2. *Single-Module*s and *Single-Nodes* have an *Intended-Function* property, which signifies the functions they perform in the target design, as opposed to the function(s) they may have as isolated entities. Hence, for example, a *Single-Module* **Add4** might have *Intended-Function* "**Address Indexing Adder**", whereas its intrinsic function (that of its *Has-Kind*) is just **Adder**.
3. A *Carrier* has a *Persistence-Storage-Property*, which describes its ability to store charge. Under some circumstances charge storage can be used as a memory mechanism.

• In the *Physical Subspace*:

1. *Physical-Model*s have a number of unique properties.
    a. *Shape*: this property expresses the bounding polygon of a piece of layout.
    b. *Boxes*: a block may not consist entirely of sub-blocks. In that case it has some primitive (layer, rectangle) boxes.
    c. *Contacts*: contacts are similar to boxes, but represent particular interlayer interconnection points.

d. *Power-Supply*: this property describes the power requirements (not the connections) of the block.

e. *Package-Description*: this property describes the package of an OEM or packaged component, as opposed to a layout.

2. A *Single-Block* has two properties, *Rotation* and *Mirroring*, that describe coordinate transformations applied to the block to determine its position and orientation.

3. A *Net* has the following properties that do not have direct counterparts in the other subspaces:

   a. *Boxes*: these are the constituent rectangles of the physical layout of the *NET*. They are colored rectangles, i.e. they have layer information attached.

   b. *Contacts*: these are the interlayer connections that form parts of the *Net*.

   c. *Path*: this is the abstract path along which the *Net*'s boxes are laid out.

   d. *Capacitance*: this represents the summed parasitic capacitance of the *Net*.

   e. *Resistance*: this represents the series parasitic resistance of the *Net*.

4. A *Sub-Net* has an additional property *Layer* that can be derived from the *Boxes* of its *Kind*.

Bibliography

[Abiteboul 84] Abiteboul, S., and Hull, R.
IFO: A Formal Semantic Database Model.
In *Proc. 3rd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*. April, 1984.

[Abrial 74] Abrial, J. R.
Data semantics.
In Klimbie, J. W., and Koffman, K. L. (editors), *Data Base Management*, pages 1-59. North-Holland, Amsterdam, 1974.

[Afsarmanesh 84] Afsarmanesh, H., and Mcleod, D.
A Framework for Semantic Database Models.
In *Proceedings of NYU Symposium on New Directions for Database Systems*. New York, NY, May, 1984.

[Afsarmanesh 85a]
Afsarmanesh, H., Knapp, D., McLeod, D., and Parker, A.
An Extensible Object-Oriented Approach to Databases for VLSI/CAD.
In *Proc. 11th Intl. Conf. on Very Large DataBases*. Stockholm, Sweden, August, 1985.

[Afsarmanesh 85b]
Afsarmanesh, H., Knapp, D., McLeod, D., and Parker, A.
An Approach to Engineering Design Databases with Applications to VLSI/CAD.
In *Proc. New Directions in Computing Systems*. Norway, August, 1985.

[Afsarmanesh 85c]
Afsarmanesh, H., Knapp, D., McLeod, D., and Parker, A.
An Object-Oriented Approach to Databases for VLSI/CAD.
Technical Report, Department of Computer Science, University of Southern California, April, 1985.

[Albano 83] Albano, A., Cardelli, L., and Orsini, R.
Galileo: a strongly typed, interactive conceptual language.
Technical Report 83-11271-2, Bell Laboratories, Murray Hill, NJ., July, 1983.

[ANSI 75] *Interim Report from the Study Group in Data Base Management Systems*
ANSI/X3/SPARC (Standard Planning and Requirements Committee), FTD (Bulletin of ACM SIGMOD), 1975.

[Batory 84] Batory, D.S. and Kim, W.
Modelling Concepts for VLSI CAD Objects.
technical report TR-84-35, Department of Computer Science, University of Texas at Austin, December, 1984.

[Birtwistle 73] Birtwistle, G. M., Dahl, O. J., Myhrhaug, B., Nygaard, K.
Simula Begin.
AUERBACH Publishers Inc., Philadelphia, Pa., 1973.

[Borgida 84] Borgida, A., Mylopoulos, J., and Wong, H.
Generalization/Specialization as a Basis for Software Specification.
On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages.
Springer-Verlag, 1984, pages 87-117, Chapter II.

[Bracchi 76] Bracchi, G., Paolini, P., and Pelagatti, G.
Binary Logical Associations in Data Modelling.
In *Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems*, pages 125-148. 1976.

[Brodie 81] Brodie, M. L.
On Modeling Behavioural Semantics of Data.
In *Proc. of Int. Conf. on Very Large Data Bases.* Cannes, France, September, 1981.

[Brodie 84] Brodie, M. L.
On the Development of Data Models.
In M. Brodie, J. Mylopoulos, and J. Smith (editors), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages*, pages 19-49. Springer-Verlag, 1984.

[Bryce 85] Bryce, D. and Hull, R.
SNAP: A Graphics-based Schema Manager.
Technical Report, University of Southern California, February, 1985.

[Buneman 79] Buneman, P., and Frankel, R. E.
A Functional Query Language.
In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 52-57. Boston, Massachusetts, 1979.

[Buneman 84] Buneman, p., and Nikhil, R.
The Functional Data Model and its Uses for Interaction with Databases.
In M. Brodie, J. Mylopoulos, and J. Smith (editors), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages*, pages 359-381. Springer-Verlag, 1984.

[Bushnell 83] Bushnell, M., Geiger, D., Kim, J., LaPotin, D., Nassif, S., Nestor, J. Rajan, J., Strojwas, A., and Walker, H.
DIF: The CMU-DA Intermediate Form.
Technical Report CMUCAD-83-11, CMU Center for Computer-Aided Design, 1983.

[Cattell 83] Cattell, R. G. G.
Design and Implementation of a Relationship-Entity-Datum Data Model.
Technical Report CSL 83-4, Xerox Corporation, Palo Alto Research Center, Palo Alto, California, May, 1983.

[Chen 76] Chen, P. P.
The entity-relationship model:Toward a unified view of data.
ACM Transactions on Database Systems 1:9-36, 1976.

[Codd 70] Codd, E. F.
A Relational Model of Data for Large Shared Data Banks.
Communications of the ACM 13(6):377-387, June, 1970.

[Codd 78] Codd, E. F., Arnold, R. S., Cadiou, J. M., Chang, C. L., and Roussopoulos, N.
RENDEZVOUS Version 1: An Experimental English-language Query Formulation System for Casual Users of Relational Data Bases..
Technical Report RJ2144, IBM Res. Lab., San Jose, California, 1978.

[Codd 79] Codd, E. F.
Extending the database relational model to capture more meaning.
ACM Transactions on Database Systems 4:397-434, 1979.

[Codd 81] Codd, E. F.
Data models in database management.
ACM SIGMOD Record 11(2):112-114, February, 1981.

[Date 81] Date, C. J.
An Introduction to Database Systems, Third Edition.
Addison-Wesley, 1981.
Consulting editors: IBM Editorial Board.

[Davis 82] Davis, R., Shrobe, H., Hamscher, W., Wieckert, K., Shirley, M., and Polit, S.
Diagnosis Based on Description of Structure and Function.
In *Proceedings of the National Conference on AI*, pages 137-142. AAAI, 1982.

[Director 81] Director, S.W., Parker, A.C., Siewiorek, D.P., and Thomas, D.E.
A Design Methodology and Computer Aids for Digital VLSI Systems.
IEEE Transactions on Circuits and Systems CAS-28:634-645, July, 1981.

[Dittrich 85] Dittrich, K.R., Kotz, A.M., and Mulle, J.M.
An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases.
Technical Report, Institut fuer Informatik II, Universitaet Karlsruhe, West Germany, 1985.

[Eastman 80] Eastman, C.M.
System Facilities for CAD Databases.
In *Proceedings of the 17th Design Automation Conference.* 1980.

[Ellis 80] Ellis, C. A., and Nutt, G. J.
Office Information Systems and Computer Science.
ACM Computing Surveys 12:27-60, 1980.

[Feldman 69] Feldman, J.A. and Rovner, P.D.
An Algol-Based Associative Language.
Communication of the ACM 12(8), August, 1969.

[Gibbs 83] Gibbs, S., and Tsichritzis, D.
A Data Modeling Approach for Office Information Systems.
ACM Transactions on Office Information Systems 1(4):299-319, October, 1983.

[Goldberg 83] Goldberg, A., and Robson, D.
SMALLTALK-80:The Language and its Implementation.
Addison-Wesley, 1983.

[Goldman 85] Goldman, K., Goldman, S., Kanellakis, P., Zdonik, S.
ISIS: Interface for a Semantic Information System.
In *ACM SIGMOD Record*, pages 328-342. Brown University, 1985.

[Granacki 85] Granacki, J., Knapp, D., and Parker, A.
The ADAM Advanced Design AutoMation System: Overview, Planner, and Natural Language Interface.
In *Proceedings of the 22nd Design Automation Conference.* 1985.

[Halo 84] *Halo Graphics Primitives Functional Description Manual*
2 edition, Lifeboat Associates, 1651 Third Ave. New York, N.Y. 10128, 1984.

[Hammer 78] Hammer, M., and McLeod, D.
The Semantic Data Model: A Modeling Mechanism for Database Applications.
In *Proceedings of ACM SIGMOD International Conference on the Management of Data.* Austin, Texas, 31 May-2 June, 1978.

[Hammer 81] Hammer, M., and McLeod, D.
Database Description with SDM: A Semantic Database Model.
ACM Transactions on Database Systems 6(3):351-386, September, 1981.

[Hendrix 77] Hendrix, G. G.
Some general comments on semantic networks.
In *Proc. 5th Int. Jiont Conf. on Artificial Intelligence*, pages 984-985. 1977.

[Hendrix 78] Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., and Slocum, J.
Developing a Natural Language Interface to Complex Data.
ACM TODS 3:105-147, 1978.

[Herot 80] Herot, C. F.
Spatial Management of Data.
ACM Transactions on Database Systems 5:493-513, 1980.

[Hull 85] Hull, R.
A survey of Research on Semantic Database Models.
Technical Report TR-85-329, University of Southern California, May, 1985.

[Katz 82] Katz, R: H.
A Database Approach for Managing VLSI Design Data.
In *Proceedings of the 19th Design Automation Conference.* 1982.

[Kent 79] Kent, W.
Limitations of record-oriented information models.
ACM Transactions on Database Systems 4:107-131, March, 1979.

[Kerschberg 75] Kerschberg, L., and Pacheco, J. E. S.
A Functional Data Base Model.
Monograph Series in Comput. Sci. and Comput. Appl., Dep. de In formatica, Pontificia Univ, Catolica do Rio de Janeiro, Brazil, 1975.

[King 82] King, R., and McLeod, D.
The Event Database Specification Model.
In *Proceedings of the Second International Conference on Databases: Improving Database Usability and Responsiveness*, pages 299-322. Jerusalem, Israel, June, 1982.

[King 84a] King, R., and McLeod, D.
Semantic Database Models.
In S.B. Yao (editor), *Principles of Database Design*. Prentice Hall, 1984.
to appear.

[King 84b] King, R., and McLeod, D.
A Unified Model and Methodology for Conceptual Database Design.
In M. Brodie, J. Mylopoulos, and J. Smith (editors), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages*, pages 313-327. Springer-Verlag, 1984.

[Knapp 83a] David Knapp, John Granacki, and Alice Parker.
An Expert Synthesis System.
In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 419-424. ACM-IEEE, September, 1983.

[Knapp 83b] Knapp, D. and Parker, A.
A Data Structure for VLSI Synthesis and Verification.
Technical Report DISC 83-6a, Digital Integrated Systems Center, Dept. of EE-Systems, University of Southern California, October, 1983.

[Knapp 85] Knapp, D. and Parker, A.
A Unified Representation for Design Information.
In *Proceedings of the Conference on Hardware Description Languages*. IFIP, 1985.

[Lochovsky 81] Lochovsky, F. H., and Tsichritzis, D. C.
Interactive Query Languages for External Data Bases.
Telidon Behavioural Reasearch 5,Technical Report DOC-TBR-DBRE-81-5-E, University of Toronto, March, 1981.

[Luo 81] Luo, D., and Yao, S. B.
Form Operations By Example - A Language for Office Information Processing.
Proc. ACM SIGMOD , 1981.

[Lyngbaek 84] Lyngbaek, P. and McLeod, D.
A Personal Data Manager.
In *Proceedings of the Tenth International Conference on Very Large Data Bases.* Singapore, August, 1984.

[McDonald 75] McDonald, N., and Stonebraker, M. R.
CUPID-The Friendly Query Language.
In *Proc. ACM Pacific 75 Regional Conf.*, pages 127-131. 1975.

[McLeod 80] McLeod, D., and Smith, J. M.
Abstraction in Databases.
In *Proceedings of Workshop on Data Abstraction, Databases and Conceptual Modelling.* Pingree Park, Colorado, 23-26 June, 1980.

[McLeod 83] McLeod, D., Bapa Rao, K. V., and Narayanaswamy, K.
An Approach to Information Management for CAD/VLSI Applications.
In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* San Jose, California, May, 1983.

[Microsoft 83] *Microsoft Mouse for the IBM Personal Computer*
Microsoft Corporation, 10700 Northup Way, Bellevue, WA 98004, 1983.
Document No. 8808-100-00.

[Motro 84a] Motro, A.
Browsing in a Loosly Structured Database.
In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* 1984.

[Motro 84b] Motro, A.
BAROQUE: a Browsing Interface to Relational Databases.
Technical Report TR-84-310, University of Southern California, August, 1984.

[Motro 85] Motro, A.
Assuring Retrievability from Unstructured Databases by Contexts.
Technical report, University of Southern California, February, 1985.

[Mylopoulos 75] Mylopoulos, J., Borgida, A., Cohen, P., Roussopoulos, N.,Tsotsos,J., and Wong, H.
TORUS - Natural Language Understanding System for Data Management.
In *Proc. 4th Intl. Joint Conf. Artificial Intelligence*, pages 414-421. 1975.

[Mylopoulos 80a] Mylopoulos, J., and Wong, H. K. T.
Some features of the TAXIS data model.
In *Proc. 6th Int. Conf. Very Large Data Bases*, pages 399-410. 1980.

[Mylopoulos 80b] Mylopoulos, J.
An Overview of Knowledge Representation.
In *Proceedings of Workshop on Data Abstraction, Database, and Conceptual Modeling.* Pingree Park, Colorado, June, 1980.

[Parker 84] Parker, A.
Automated Synthesis of Digital Systems.
IEEE Design and Test , November, 1984.

[Priddy 85] Priddy, C., Afsarmanesh, H.
ISL User Interface Manual.
Technical Report, University of Southern California, August, 1985.
in preparation.

[Roussopoulos 75]
Roussopolous, N., and Mylopoulos, J.
Using semantic networks for data base management.
In *Proc. 1st Int. Conf. Very Large Data Bases*, pages 144-172. 1975.

[Roussopoulos 77]
Roussopoulos, N.
ADD: Algebraic Data Definition.
In *Proc. 6th Texas Conf. on Computing Systems*. Austin, Texas, November, 1977.

[Schrefl 84] Schrefl, M., Tjoa, A. M., Wagner, R. R.
Comparison- Criteria for Semantic Data Models.
In *International Conference on Data Engineering*, pages 120-125. IEEE, Los Angeles, California, April, 1984.

[Shipman 81] Shipman, D.
The Functional Data Model and the Data Language DAPLEX.
ACM Transactions on Database Systems 2(3):140-173, March, 1981.

[Smith 77a] Smith, J. M., and Smith, D. C. P.
Database Abstractions: Aggregation and Generalization.
ACM Transactions on Database Systems 2(2):105-133, June, 1977.

[Smith 77b] Smith, J. M., and Smith, D. C. P.
Database Abstractions: Aggregation.
Communications of the ACM 20(6):405-413, June, 1977.

[Stonebraker 82] Stonebraker, M., and Kalash, J.
TIMBER: A Sophisticated Relation Browser.
In *Proc. 8th Int. Conf. Very Large Data Bases*, pages 1-10. Mexico City, Mexico, Sept. 8-10, 1982.

[Stonebraker 84] Stonebraker, M.
Adding Semantic Knowledge to a Relational Database.
In M. Brodie, J. Mylopoulos, and J. Smith (editors), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages*, pages 333-354. Springer-Verlag, 1984.

[Su 79] Su, S. Y. W., and Lo, D. H.
A Semantic Association Model for Conceptual Database Design.
In *Proceedings of International Conference on the Entity-Relationship Approach to Systems Analysis and Design.* Los Angeles, California, December, 1979.

[Tsichritzis 82] Tsichritzis, D. C., and Lochovsky, F. H.
Data Models.
Prentice-Hall, Englewood Cliffs, N.J., 1982.

[Ullman 82] Ullman, J. D.
Principles of Database Systems, Second Edition.
Computer Science Press, Rockville, Maryland, 1982.

[Wiederhold 79] Wiederhold, G., and El-Masri, R.
The Structural Model for Database Design.
In *Proceedings of International Conference on the Entity-Relationship Approach to Systems Analysis and Design.*
Los Angeles, California, December, 1979.

[Wong 77] Wong, H. K. T., and Mylopoulos, J.
Two views of data semantics: A survey of data models in artificial intelligence and database management.
INFOR 15:344-383, 1977.

[Wong 79] Wong, S. and Bristol, W.A.
A CAD Database.
In *Proceedings of the 16th Design Automation Conference.*
1979.

[Wong 82] Wong, H. K. T., and Kuo, I.
GUIDE: Graphical User Interface for Database Exploration.
In *Proc. 8th Int. Conf. Very Large Data Bases*, pages 22-32.
Mexico City, Mexico, Sept. 8-10, 1982.

[Zaniolo 85] Zaniolo, C., Ait-Kaci, H., Beech, D., Cammarata, S., Kerschberg, L., and Maier, D.
Object Oriented Database Systems and Knowledge Systems.
Technical report, MCC, 1985.

[Zloof 75] Zloof, M. M.
Query by example: The invocation and definition of tables and forms.
In *Proc. 1st Int. Conf. Very Large Data Bases*, pages 1-24.
1975.

[Zloof 80] Zloof, M. M.
A language for Office and Business Automation.
Tech. rep. RC8091, IBM, 1980.
Yorktown Heights, New York.