Development of a Computer-Understandable Representation of Design Rationale to Support Value Engineering

by
Primo T. Alcantara, Jr.

Dissertation Submitted to the Faculty of
Virginia Polytechnic Institute and State University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
in
Civil Engineering

APPROVED

Dr. J. M. de la Garza
(Chairman)

Mr. M. Bandy
OVEST

Dr. Y. J. Beliveau

Prof. D. B. Jones

Dr. W. E. Showalter

Dr. M. C. Vorster

July 1996
Blacksburg, Virginia

DEVELOPMENT OF A COMPUTER-UNDERSTANDABLE REPRESENTATION OF DESIGN RATIONALE TO SUPPORT VALUE ENGINEERING

by

Primo T. Alcantara, Jr.

Dr. Jesus M. de la Garza, Committee Chairman

Department of Civil Engineering

(ABSTRACT)

The life span of facilities produced by the Architecture-Engineering-Construction industry is typically 25 years or more. Several distinct phases characterize the life span of a facility. Each of these phases involve numerous participants from different professional disciplines. These participants generate and use a lot of information about the facility. Current methods used by the industry to convey this information are drawings and specifications. However, these drawings and specifications reflect only a summary of the information generated and used by the project participants. This summarized information only describes the product. Information about the process of generating these information becomes implicit in the drawings and specifications. Rationale is the collective term for this set of implicit process information.

The main issue addressed by this dissertation is the need to communicate design rationale information. Design rationale is a subset of the entire rationale generated for a facility.

Design rationale refers to information about the design process. Explicitly stating design rationale information reduces the chance of misinterpreting design drawings and specifications.

The primary objective of this dissertation is to define a data structure capable of representing design rationale information. This data structure also allows a computer system to perform analytical tasks on the design rationale data. Examples of analytical tasks a computer system can perform on design rationale data include: generating a parameter dependency network and resolving data conflicts. This dissertation defines this data structure as two separate but complementary modules. The Knowledge Representation Module assists in gathering project-specific *product* information. The Rationale Storage Module assists in capturing project-specific *process* information. This dissertation discusses each of these two modules in detail.

The secondary objectives of this dissertation include: (1) defining a computer program architecture, (2) creating a computer program interface, and (3) verifying the appropriateness of the data structure in representing design rationale. A proof-of-concept computer program, DRIVE, applied to an actual value engineering study project accomplishes these objectives.

For

Maria Angelica,

Paulo Miguel

and

David Gabriel

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Some products of the Architecture-Engineering-Construction (AEC) industry are buildings, bridges, roads, dams, and wharves. These structures typically have life spans of 25 years or more. Concept definition, design, construction, operation, maintenance, retrofit, and demolition are the various phases in the life cycle of constructed facilities. Although these facilities have such a long life span, the AEC industry concentrates on minimizing the design and construction cost of these facilities, frequently without regard for the life cycle cost implications. Often, design and construction costs are not the major cost items when considering the life cycle cost of a facility.

One factor that results in unnecessary life cycle cost expenditures is the lack of information flowing from one life cycle phase to another. A common practice in the AEC industry is the use of different firms for the design, construction, operation, and maintenance of a facility. This practice results in useful information being "trapped" in one phase and not communicated on through subsequent phases. If the personnel

involved in the latter phases had access to this "trapped" information, they might have not incurred unnecessary cost expenditures. This "trapped" information is usually the reasoning or rationale behind the decisions made in an earlier phase.

Figure 1.1 shows a graph on the total expenditures made for a typical constructed facility over its life span. The two dashed lines represent two speculated consequences of capturing design rationale on the total facility life cycle expenditures. These dashed lines show that capturing design rationale theoretically leads to lower total expenditures over the entire life of a facility. However, the author also speculates that these life cycle savings is only possible if owners are willing to spend more and compensate designers for capturing design rationale during the concept definition and design stages.

Architectural programming is the term used in the AEC industry to refer to the concept definition stage for building projects. In practice, the effort spent on architectural programming is becoming smaller as designers are under a lot of pressure to produce the design drawings as quick and as cheap as possible. If the owners are willing to spend more on initial costs, designers can properly perform the architectural programming task and capture rationale for these initial stages. These activities are very beneficial to the life cycle cost effectiveness of a facility.

One dashed line shows that design rationale improves the construction cost effectiveness of a facility. This shows that savings resulting from the higher design costs can manifest

itself as early as the construction phase of the project. The other dashed line shows that capturing design rationale results in higher design and construction expenditures, breaking even only during the operation and maintenance stage. Both dashed lines ultimately lead to a lower facility life cycle cost.

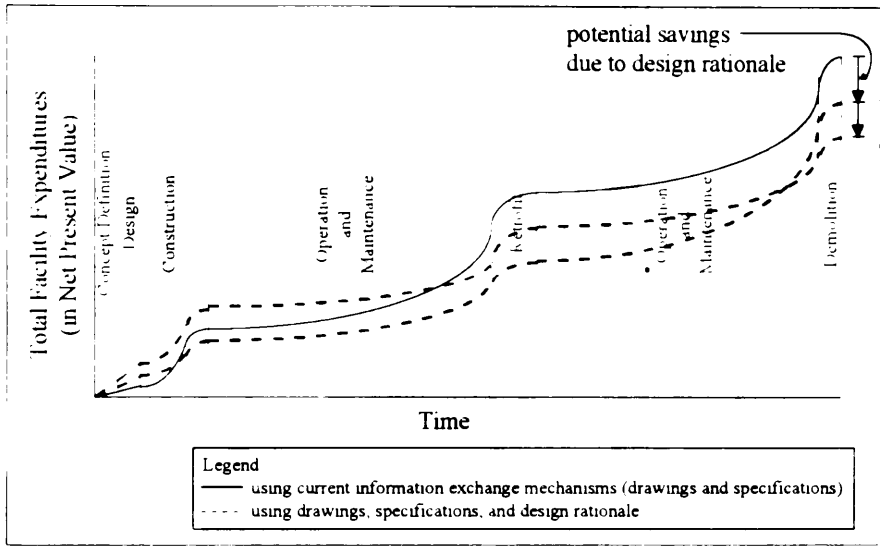


Figure 1.1 Speculated Effects of Capturing Design Rationale

Design rationale is the collection of all the data, knowledge, and reasoning about the evolution of a design product. Capturing and storing design rationale is synonymous with explicitly expressing the requirements, preferences, and reasoning implicitly embedded within the design drawings and specifications. The design drawings and specifications only represent the final results of the design process. Design is only one phase in the life cycle of constructed facilities. Access to design rationale information is especially important to the people involved in the latter life cycle stages such as construction,

operation, maintenance, rehabilitation, and demolition. The ability to retrieve rationale allows the players in these downstream stages to perform their tasks more efficiently. Access to design rationale information avoids the need to speculate about or request for information implicitly stored in the drawings and specifications.

Value Engineering (VE) specialists benefit greatly from the explicit representation of design rationale information. VE is a systematic method of analyzing the functions of systems to satisfy user needs and to maximize the level of service at lowest life cycle cost. The same individuals who performed the design of a system can, by themselves, perform a VE analysis on their design. However, the usual case is that other people, VE specialists, work together with the designers in conducting VE studies. Currently, VE specialists infer design rationale information from the drawings and specifications. These rationale information include the functions, basis, and restrictions of the various facility systems.

The AEC industry benefits greatly from the ability to access design rationale information. Facility projects designed with the aid of design rationale information will, theoretically, have fewer conflicts occurring during the construction phase. Designers tend to optimize their narrow scope of the total design. For example, in building projects, structural engineers optimize the structural design, sometimes without regard to the mechanical design. The explicit statement of design rationale allows designers to better coordinate their design efforts and make trade-offs with other designers, specially those in other areas

of specialty. Thus, design rationale helps in achieving an optimum design for the entire facility leading to a more efficient utilization of the owners' resources.

Rationale is not exclusive to the design stage. The other life cycle stages, such as construction and operation, also have their own rationale for the various decisions made in each of these stages. A "life cycle" rationale system captures rationale for all stages. This allows owners to have a computer model of their facility complete with rationale taken from designers, builders, and operators. Such a system will definitely help the owners during the entire life cycle of their facilities. Construction personnel can understand the reasons behind the design thereby allowing them to construct more efficiently the facility. Facility operators can maintain the facility more effectively because of their ability to access the rationale behind the design and construction of the facility. Finally, people responsible for retrofitting and demolishing a facility do not need to speculate about the decisions made by the designers, builders, and facility operators during its life cycle.

The AEC industry is widely using computers in most of its day-to-day operations (Choi and Ibbs 1990). However, with a few exceptions, the industry is still using "paper" drawings and specifications to exchange project information. Computers merely changed and expedited the paper generation process (de la Garza et al. 1994). Using computers as a paper generator is an inefficient way of using this powerful technology. Computer technologies further enhance the quality of a design by capturing both the design product and the design rationale. This research aims to develop a data structure allowing the use

of computers to capture, store, retrieve, and process information about the rationale behind a design of a facility.

1.2 RESEARCH OBJECTIVES

The primary objective of this research effort is the creation of a generic data structure capable of representing the reasoning behind a design. This generic data structure organizes design rationale data into a useful information source for designers, value engineers, cost estimators, construction schedulers and other persons involved in activities within and following the design phase of a project. Asides from assisting people by providing organized, meaningful information, the data structure also allows a computer software program to perform analysis on the design rationale data.

Other secondary objectives of this dissertation include:

- defining a software architecture for a generic design rationale system.
- verifying the appropriateness of the data structure in representing design rationale
- creating a software interface showing the possibility of capturing and presenting rationale information stored in the above-mentioned data structure.

1.3 RESEARCH LIMITATIONS

The AEC industry has five domain areas — building construction, highway construction, heavy construction, industrial construction, and marine construction. This research

focuses on developing a rationale representation scheme for the design of a facility project. The building construction domain serves as an example for the generic design rationale system described in this research. The building construction domain model contains objects (for example, rooms, columns), as well as their associated properties (for example, dimensions, load carrying capacity). Other domain areas (for example, highway construction) contain different objects (for example, roadways). Development of models for domain areas other than building construction is beyond the scope of this research.

Rationale about a facility project starts accumulating from the concept definition stage and terminates only after the demolition stage. Each of the life cycle stages has its own particular set of rationale information. For example, rationale during the design stage involves relating design parameters, owner requirements, and code requirements. During the construction stage, relationships among labor, equipment, methods, material, and time constitute rationale. This dissertation limits the rationale representation structure to support the capture of rationale only during the design stage. However, this structure serves as a starting point for supporting rationale capture for the other life cycle stages.

The creation of a robust building model capable of representing all possible building design entities (for example, rooms, walls, door frames, HVAC equipment, plumbing) is not within the extent of this research. Rather, the dissertation creates sufficient model entities to illustrate the design rationale concept for building construction.

The computer program by-product of this research assists in the capture, storage, retrieval, and processing of design rationale information. Its intended users are designers and value engineers. Many others (for example, construction planners, facility operators) benefit from design rationale. However, it is outside the bounds of this research to provide support for their specific tasks (for example, estimating, scheduling).

A VE study is a systematic method divided into several phases: Information Gathering, Speculation, Analysis, Development, Presentation, and Post-Occupancy Evaluation. Each of these phases has different uses of captured design rationale. For example, a design rationale system supports the Information Gathering phase by organizing and presenting rationale information. During the Speculation phase, it may analyze the stored rationale information for any design items suitable for VE. The computer program by-product of this research actively supports only the Information Gathering phase of VE.

1.4 RESEARCH METHODOLOGY

1.4.1 Literature Review

The first task comprising the methodology of this research is a review of the available literature on various subject areas relevant to the development of a data structure for representing design rationale. The first subject area focuses on the fragmented nature of the AEC industry. Various researchers have advocated the use of computerized representation techniques to address this fragmentation problem (de la Garza and Oralkan

1995, Howard 1991, Ito et al. 1989, Peña-Mora et al. 1995). One of these representation techniques involves the capture of design rationale information.

The second subject area for the literature review concentrates on design rationale — its definition, the various types of information comprising rationale, and the different implementations of design rationale systems by other researchers. The broad applicability of the design rationale concept to the AEC industry has necessitated the selection of one specific AEC domain area, namely, building construction. Capturing design rationale for a building facility enhances the efficiency of performing various tasks such as cost estimating, value engineering, and construction planning. The varied nature of these tasks has required the selection of only one task, namely, value engineering, for the purposes of illustrating the benefits of capturing design rationale. These limitations have further expanded the literature review to include the concepts of architectural programming, building modeling, computer-aided design, and value engineering.

1.4.2 Field Studies

The field studies have involved observing and participating in two actual VE studies conducted by the **O**ffice of the Chief of Engineers' **V**alue **E**ngineering **S**tudy **T**eam (OVEST). OVEST conducts VE studies for the U. S. Army Corps of Engineers, which constructs various projects totaling billions of dollars annually. The two field studies have focused on a building project for a military hospital complex. These field studies have

helped get valuable insight into current VE practices and problems as well as define the functionalities needed by a computer application supporting the VE process.

1.4.3 Data Structure Development

Concepts investigated in the literature review and issues identified during the field studies form the foundation for the definition of a data structure to represent design rationale information. The development of a theoretical data structure representing design rationale is the main objective of this research.

A secondary research objective is the definition of a computer program architecture capable of supporting an implementation of this data structure. Although this research has placed various limitations (Section 1.3), a major requirement in the definition of the computer program architecture is the easy extensibility of the data structure. This allows other researchers to address the limitations of this dissertation.

1.4.4 Computer Implementation

A secondary research objective is the verification of the appropriateness of the data structure in representing design rationale. The development of a computer implementation using the theoretical data structure accomplishes this objective. Similar to the data structure development, the literature review and field studies form the background materials helping define the functionalities of the computer program by-product of this research. **D**esign **R**ationale for the **I**nformation phase of **V**alue **E**ngineering (DRIVE) is

the name given to this computer implementation DRIVE assists in the capture, storage, retrieval, and processing of design rationale information.

The computer implementation process also involved selecting the computer platform and prototyping software. The platform selected was the Microsoft Windows operating system and the prototyping software selected were Intellicorp's Kappa-PC and Autodesk's AutoCAD. Kappa-PC is an object-oriented expert system prototyping environment. AutoCAD is a popular graphical modeling application used by the building design and construction industry.

1.4.5 Industry Validation

The industry validation studies applied the computer program by-product of this research to two actual building projects. The first project is the New River Valley Regional Center for Economic Development building located in Dublin, Virginia. The architect of this project is a Blacksburg architectural design firm, Mills, Oliver, and Webb, Inc. The author conducted an interview with the project architect to gather design rationale information. The second project is the McAlpine Locks Support Buildings Project. The author has joined the VE study conducted by OVEST for the McAlpine project. In both test projects, the designers have given the author access to their electronic two-dimensional design drawings. The author has used these drawings to create the three-dimensional DRIVE models for these projects. The author has also entered into the DRIVE system the design rationale information gathered from the various rationale elicitation activities.

1.5 DISSERTATION CHAPTERS

The dissertation consists of seven chapters. Chapter one serves as an introduction of the research. It briefly discusses the problems faced by the AEC industry, design rationale, and VE. It also states the research objectives, limitations, and methodology.

Chapter two provides a summary of the available literature on the fragmentation of the AEC industry, object-oriented programming, design rationale, architectural programming, VE, building modeling, and computer aided design. The discussion focuses on the relevance of these concepts to this research.

Chapter three provides an overview of a computer program architecture for a generic design rationale system. Several domain-dependent knowledge representation modules coupled with a single domain-independent rationale storage module comprise this rationale capture system. It also includes descriptions of specific computer programs utilized in the development of the computer program by-product (DRIVE) of this research. This chapter contains a discussion on the two possible methods of using DRIVE to capture design rationale. It also describes several usability guidelines implemented on the DRIVE user interface. It also includes a discussion of the program structure of DRIVE. This chapter also describes an actual building project that serves as an example illustrating the various aspects of the theoretical data structure

Chapters four and five present the main objective of this dissertation, a data structure capable of representing design rationale information. The concepts detailed in these two chapters form the theoretical underpinnings for the DRIVE computer program. All the sections in both these chapters are each divided into the following four sub-sections — theoretical discussion, practical example, DRIVE interface, and DRIVE implementation. Chapters four and five describe, respectively, the structure of the domain-dependent **K**nowledge **R**epresentation **M**odule (KRM) and the domain-independent **R**ationale **S**torage **M**odule (RSM). The KRM consists of objects representing the design. The RSM contains the rationale behind decisions made about those design objects. Chapter five also shows how a computer program can perform operations on rationale data.

Chapter six focuses on specific DRIVE capabilities supporting the VE Information phase. This chapter also has the four sub-section breakdown of chapters four and five.

Chapter seven summarizes the whole research effort. Findings derived from this research and recommendations for future research work conclude this dissertation.

The dissertation also has four appendices. Appendix A shows the BNF grammar representation of the theoretical data structures. Appendix B contains a tutorial manual on using DRIVE. Appendix C presents all the capabilities of DRIVE using a set of images taken from the DRIVE application. Appendix D is a PC-formatted diskette containing the source code for the DRIVE application.

CHAPTER 2

LITERATURE REVIEW

2.1 AEC INDUSTRY

Concept definition, design, construction, operation, maintenance, retrofit, and demolition are the various phases in the life cycle of constructed facilities. Owners, design firms, construction firms, material suppliers, bankers, lawyers, government agencies, end users, facility operators, maintenance teams, and demolition firms are some of the participants involved in these life cycle phases. Each life cycle phase involves only several of these participants. For example, the design phase traditionally involves only the owners and the design team with negligible participation from the construction team.

The emphasis on specialization is one of the major cause of the fragmentation prevalent in the AEC industry today (Howard et al. 1989). Specialization promotes flexibility and technological innovation within an industry. During the 1960s, specialization increased productivity within the AEC industry. However, as the degree of specialization increased, the problems associated with fragmentation began to negate some of the benefits of specialization.

Horizontal and vertical are the two types of fragmentation in the AEC industry (Howard et al. 1989). Horizontal fragmentation occurs between the numerous participants involved during one particular life cycle phase. For example, the architectural, structural, mechanical, electrical, and civil engineering specialists during the design phase seek to optimize their design sometimes adversely affecting the design in another field. Vertical fragmentation refers to the separation of each of the life cycle phases. For example, the construction team does not receive input from the facility operators. Facility operators can give valuable advice to construction personnel, specifically on operability and maintainability issues of constructed facilities.

The breakdown of the communication infrastructure between the numerous project participants is one of the negative effects of fragmentation. Project information passed between participants typically reflects only the results of their highly specialized work. This occurs both horizontally within a life cycle phase (for example, the architectural designer providing the final layout and specifications of the rooms to the mechanical designer), or vertically across life cycle phases (for example, the general contractor submitting the complete as-built drawings to the owner). By conveying only the end results of specialized work, much of the process knowledge becomes implicit in the description of the product knowledge. Product knowledge refers to information about the product itself (for example, dimensions, materials used). Process knowledge refers to information about the process leading to the product knowledge description (for example, construction means and methods). One of the aspects of process knowledge is rationale.

2.2 OBJECT-ORIENTED PROGRAMMING

Object-**O**riented **P**rogramming (OOP) is one of the numerous styles of computer programming available today. The ever increasing complexity of computer program applications is one of the reasons behind the creation of the OOP style. Modules allow programmers to develop and maintain larger and larger computer programs. OOP supports modularity through the concepts of objects, message passing, and inheritance. The OOP discussion in this section is a synthesis of the OOP concepts detailed in the works of Dym and Levitt (1991), Stefik and Bobrow (1985), Fikes and Kehler (1985), and Schildt (1991), as well as the Kappa-PC Reference Manual (1994).

2.2.1 Objects

The basic building block in OOP is the object. An object (also called frame, schema, or unit) in OOP is an entity containing properties and behavior. The concept of an object in OOP is similar to the concept of an object in the real world. For example, a reinforced concrete beam is a real world object. It has properties like length, width, height, weight, concrete strength and steel reinforcement. It also has defined behavior like the amount of deflection it makes under the action of a particular load distribution. This behavior is dependent upon the properties of the concrete beam and the load distribution. Figure 2.1 shows the OOP representation of a concrete beam object. It has attributes corresponding to the various properties of the concrete beam. It also contains methods describing its behavior. OOP uses the terms attributes, slots, variables, and parameters to refer to object properties. It also uses the terms methods and procedures to refer to object behaviors.

Concrete Beam		
Attributes		Methods
Length	4000 mm	Deflection Calculation
Cardinality	1	dependent upon
Value Type	Numeric	Length
Width	250 mm	Width
Cardinality	1	Height
Value Type	Numeric	Concrete Strength
Height	500 mm	Steel Reinforcement
Cardinality	1	Steel Strength
Value Type	Numeric	Load Distribution
...		...

Figure 2.1 Example of an Object

Properties or attributes in OOP contain facets. Attributes are characteristics of an object while facets are characteristic of the attribute itself. Some examples of facets include *cardinality*, *value type*, *permissible values*, and *demons*. *Cardinality* facets limit the number of separate values for an attribute. *Cardinality* facets do not constrain the actual values in a slot. For example, the attribute `Length` of the object `Concrete Beam` has a cardinality equal to 1, meaning that there is only one value for the length of the specific concrete beam. *Value type* facets limit the type of values for an attribute. Common value types include integer (a numeric integer), numeric (a numeric value), character (a single letter or number), string (a sequence of characters), object (an object name), and boolean (true or false). For the attribute `Length` of the object `Concrete Beam`, the *value type* facet is numeric. The *permissible values* facet constrains the actual values valid for a particular attribute. This facet is dependent upon the *value type* facet. For example, if the *value type* facet for an attribute is numeric, then the *permissible values* facet is usually a range of numeric values.

Demons are another facet of object-attributes. *Demons*, also called active values, are special methods or procedures attached to a particular attribute. The computer program invokes these *demons* when it requires, changes, or accesses the value of an attribute. These *demons* perform various procedural computations on the attribute-values depending on the current action of the program. For example, a construction cost estimating system requires the cross-sectional area of a beam for the calculation of the construction cost. If the value of the attribute `Cross-Sectional Area` for the object `Beam` is unknown, the program invokes the "when required" *demon* to calculate the value of the unknown area. This "when required" *demon* contains procedural code setting the value of the attribute `Cross-Sectional Area` to the product of the `Width` and `Height` attributes of the `Beam` object. *Demons* can also monitor any changes made to the attribute-values. Used in conjunction with other attribute-facets, such as *permissible values*, these *demons* can perform error checking, ensuring that the new attribute-values conform to the *permissible values* facet.

2.2.2 Message Passing

Message passing, also called sending messages, is the primary means of executing code in OOP. Methods contain the procedural code representing the behavior of an object. For example, the maximum deflection of a simple beam under a uniformly distributed load depends on the intensity of the load and the length, modulus of elasticity, and moment of inertia of the beam. Using OOP representation, the object `Simple Beam` contains a method `Maximum Deflection Uniform Load` that performs the actual calculation.

Methods can also contain instructions to send messages to other object-methods. Asides from explicit message passing, methods can also trigger demons associated with an object-attribute. These demons can send messages to other methods. Demons can also trigger other demons. Thus, in OOP, a seemingly innocent change in the value of an object-attribute, or the sending of a single message, can have far reaching consequences.

2.2.3 Inheritance

Grouping related objects together leads to the formation of object networks. Object networks, also known as frame hierarchies, represent a taxonomic breakdown structure. The concept of inheritance arises from the generalization-specialization relationships existing between the various objects in a taxonomy. Figure 2.2 shows an example of an object network. It shows two kinds of objects, namely, classes and instances. Classes, represented by ovals, are objects containing a description valid for a set of similar objects. In Figure 2.2, `Office Buildings`, `Housing Buildings`, and `Apartment Buildings` are examples of classes. Instances, denoted by rectangles, are objects representing a single specific entity, such as `Office Building A`, `Office Building B`, and `Hotel Building D` in Figure 2.2.

Instances are always a specialization of a class and are never a generalization of another object. In other words, instances are the end nodes of an object network. Classes, on the other hand, can simultaneously exist as a generalization and a specialization of different objects. For example in Figure 2.2, `Housing Buildings` is simultaneously a specialization

of **Buildings** and a generalization of **Apartment Buildings**. In an object network, higher order objects have specializations consisting of lower level objects. Conversely, lower level objects have a generalization consisting of a higher level object.

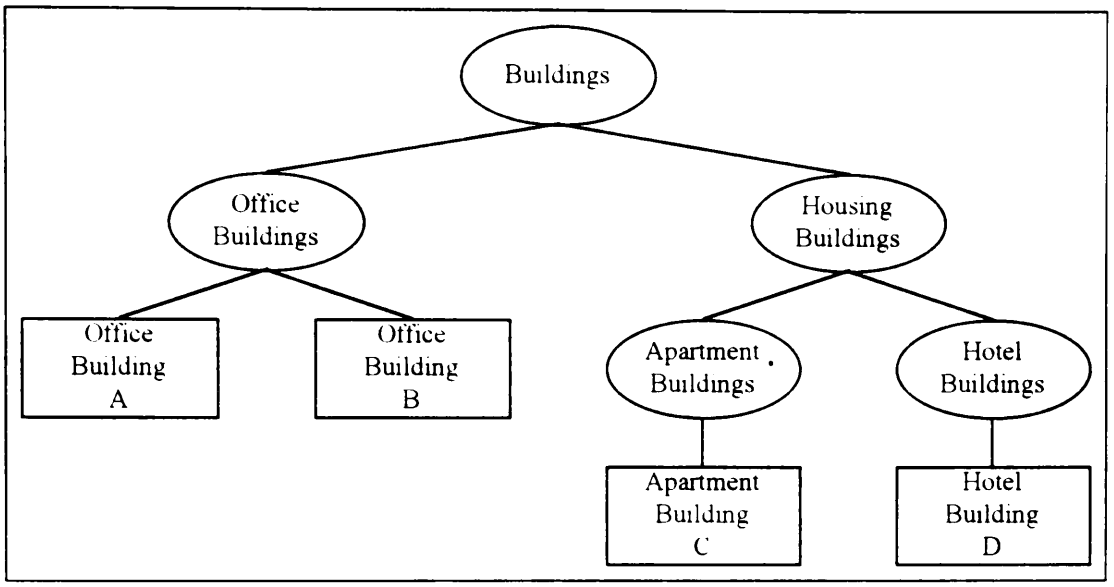


Figure 2.2 Object Network Example

Lower level objects inherit the attributes (properties) and methods (behaviors) of their respective higher level objects. Inheritance allows for modularity of program data. Modularity provides easy extensibility to the data of an existing program. The creation of new classes that are almost similar to an existing class involves the use of the generalization-specialization relationship. For example in Figure 2.2, when the need arises to define a new object to represent **Residential Houses**, programmers can search the existing object network for a similar object to act as the generalization for the new object. The existing object **Housing Buildings** is the class most similar to **Residential Houses**. Programmers can create a new specialization class from **Housing Buildings**. They can

then modify the attributes of this new class to reflect information specific to **Residential Houses** but not generally applicable to all **Housing Buildings**. Figure 2.3 shows in detail the relationship between these two objects.

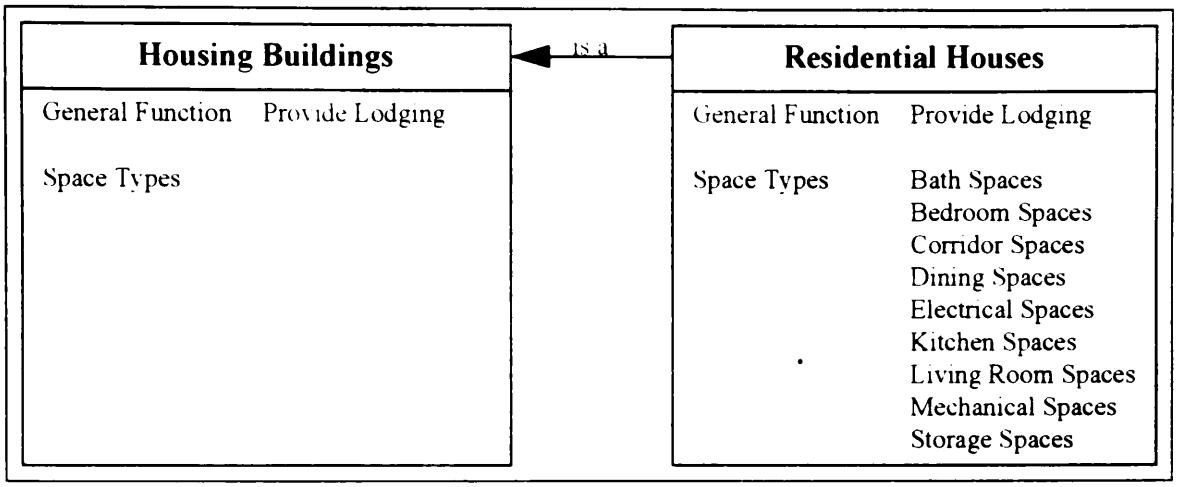


Figure 2.3 Object Inheritance Example

Inheritance also helps in efficiently storing and organizing data contained in a program. It may seem from Figure 2.3 that OOP stores a lot of redundant data. For example, the value **Provide Lodging** of the attribute **General Function** of the object **Housing Buildings** is the same as that of the object **Residential Houses**. However, the internal storage mechanism of OOP makes use of pointers. Pointers refer to the actual physical location of a particular value in computer memory. Rather than store the value **Provide Lodging** twice — once for the object **Housing Buildings** and the other for the object **Residential Houses**, OOP stores the value **Provide Lodging** once at the **Housing Buildings** level and all its specializations or subclasses make use of pointers to the value stored at the **Housing Buildings** level. Attributes and values are not the only things

inherited from classes by subclasses and instances. Lower level objects also inherit methods. As with attributes and values, subclasses and instances do not store copies of the procedural code embedded in methods defined at a higher level class. OOP removes the inefficiency of storing copies of the same procedural code by using pointers.

2.3 DESIGN RATIONALE

2.3.1 Definition of Design Rationale

Rationale is an explanation of the underlying reason regarding a particular decision. For constructed facilities, rationale starts accumulating from the initial definition of the project concept up to the final demolition of the project. An example of rationale for a decision made during the design phase is "the reason for providing additional ceiling crawl space is to allow for the easy installation of additional mechanical and electrical ducts in the future." Another example, this time for the construction phase, is "the reason for scheduling the installation of the mechanical equipment before constructing the roof slab is that the equipment requires the use of a hoisting crane, thereby creating the need for allowing access from above for the crane." Each life cycle phase from concept definition to final demolition has its own set of rationale information. However, this dissertation focuses only on representing rationale during the design phase.

Design Rationale (DR) refers to the collection of explicitly stated information about the reasons explaining, justifying, and disproving decisions about a design (Chandrasekaran et al. 1993, de la Garza and Oralkan, 1992, Fischer et al. 1991, Garcia et al. 1994, Ganeshan

et al. 1991, Lee and Lai 1991, Mostow 1985, Peña-Mora et al. 1995, Ullman 1994). The artificial intelligence scientific community also uses the term design history (Mostow 1985) to refer to DR. The STEP standardization committees use the term design intent while the commercial-industrial business complex uses corporate memory (Ullman 1994).

2.3.2 Information Comprising Design Rationale

Design documents currently used by the AEC industry are design drawings and product specifications. These documents efficiently communicate information answering the question "WHAT does the final constructed product look like?" The information explicitly conveyed by these documents are:

- design object (for example, conference room walls)
- graphical object representation (for example, detail A9-1 on sheet A9)
- object specification (for example, brand X or equal)

A DR document answers "how", "why", and "why not" questions about information implicitly embedded in design drawings and specifications. The following list shows some of the explicit information presented by a DR document (de la Garza and Oralkan 1995, Garcia et al. 1994, Lee and Lai 1991, Peña-Mora et al. 1995, Ullman 1994):

- decision problem (for example, material specification)
- object functions (for example, filter sound, enclose space)
- object performances (for example, acoustic, economic)
- performance parameters (for example, sound transmission coefficient [STC], construction cost)
- object alternatives (for example, brand X, brand Y, brand Z)
- parameter assumptions (for example, construction cost values for all brands assumed material available locally)
- original parameter values (for example, brand X has STC 46 and costs $19/unit, brand Y has STC 52 and costs $21/unit, brand Z has STC 38 and costs $14/unit)
- parameter constraint type (for example, minimum value of STC)
- parameter constraint value (for example, STC 45)
- justification about constraint (for example, section 3.4.5.6 of code ABC)
- justification for alternatives (for example, brand X satisfies STC requirement, brand Y satisfies STC requirement, brand Z has cheapest construction cost)
- justification against alternatives (for example, brand X has higher construction cost, brand Y has higher construction cost, brand Z violates STC requirement)
- revised parameter values (for example, brand X has STC 46 and costs $25/unit, brand Y has STC 52 and costs $21/unit, brand Z has STC 38 and costs $14/unit)
- justification about revisions (for example, brand X not available locally)
- design object version history (for example, substitute brand Y for brand X)

Designers solve decision problems (material specification) by determining required functions (filter sound) and transforming these requirements into specific performance parameters (minimum STC of 45). These performance parameters then form the basis for the evaluation of alternatives. The selection of the most suitable alternative leads to an object specification (brand X or equal) and a graphical object representation (detail A9-1).

2.3.3 Uses of Design Rationale

DR information helps the original designer by storing details about their current design task. As designers concentrate on one particular aspect of a design project, they typically forget the details on another particular aspect (Ullman 1988). A DR system allows designers to retrieve specific details about their previous design decisions.

Another use of DR systems is to assist designers in generating a design. DR systems can contain specific domain knowledge such as **H**eating, **V**entilation, and **A**ir **C**onditioning (HVAC) design (Garcia et al. 1994) and kitchen design (Fischer et al. 1991). DR systems can act as intelligent design assistants — notifying the designer when a design decision conflicts with a specific rule embedded within the domain model.

DR also assists in coordinating the design efforts of the numerous designers (for example, architectural, structural, mechanical, electrical, and civil engineers) working concurrently on various aspects of a single building project. DR helps designers understand the reasoning behind design decisions made by other designers. This increased understanding

assists in negotiating and resolving the typical conflicts arising from the partial designs of the various designers from different fields (Peña-Mora et al. 1995, Klein 1993).

Access to DR information also allows people involved in other stages in the project life cycle to perform their tasks more efficiently (de la Garza and Oralkan 1995). For example, a construction planning engineer in charge of formulating a construction contract bid notes that partition walls comprise one of the major cost items for the building. The engineer notes that the specifications state the use of brand X or equal. Through the availability of DR information, the engineer understands the designers' assumptions and reasons for specifying brand X as the material type for the partition walls. The designers specified brand X because of its STC value of 46, which satisfies the minimum code requirements (STC 45). The designers also assumed that brand X is available locally for a unit cost of $19/unit. However, brand X is not locally available and costs $25/unit to import. Given this information, the engineer decides to use brand Y (STC 52, $21/unit) in place of brand X (STC 46, $25/unit). The engineer also avoids making the mistake of using the cheapest alternative brand Z ($14/unit) because of its low STC value (STC 38), which violates the minimum value code requirement.

2.3.4 Classification of Design Rationale Systems

Peña-Mora et al. (1995) outlines three classification schemes for DR systems. These schemes are: (1) Single vs. Multiple Participants, (2) Passive vs. Active Capture of Design Rationale, and (3) User-Driven vs. Computer-Supported Conflict Mitigation. The first

classification scheme focuses on how different DR systems handle cooperative work among numerous designers. Single refers to a DR system allowing only one designer to work on one project at any one time. Multiple allows several designers to work concurrently on a single project. The second classification scheme considers whether designers are solely responsible for storing DR in a computerized data structure (Passive Capture) or the computer system itself generates or provides DR (Active Capture). Finally, the third classification scheme concentrates on the level of support given by the computer system in resolving design conflicts. User-Driven indicates that designers have to resolve design conflicts manually. Computer-Supported indicates that the computer system assists in conflict resolution.

This research uses classification scheme focusing on the level of computer usage of the captured rationale. In essence, this is a synthesis of Peña-Mora's second and third classification schemes. On one extreme, a passive user-driven DR system only assists designers in storing and retrieving the rationale. Designers are responsible for the interpretation of the captured rationale and the computer merely acts as a database search engine. At the other end of the spectrum, an active computer-supported DR system performs computations on captured rationale. Designers still need to express their rationale to an active system. However, an active system uses these captured rationale to assist designers in certain design tasks, such as design verification and conflict resolution.

2.3.5 Examples of Design Rationale Systems

The purpose of this section is to present the some of various DR systems done by other researchers. Conducting this literature review ensures that the dissertation makes a contribution to the body of knowledge. All the DR systems described in this section has had an impact, to some extent, on this dissertation. Of these systems, SALT (Section 2.3.5.6) and SOS (Section 2.3.5.8) have had significant influence on this research work.

2.3.5.1 Active Design Documents (ADD)

ADD (Garcia et al. 1994) seeks to capture DR by using the computer as a "designer's apprentice". Its domain is the preliminary design of HVAC systems for commercial office buildings. It contains a pre-defined set of relationships between the various design parameters (for example, `Heating Capacity`, `Primary Heating System`, and `Winter Design Temperature`). These relationships allow the system to expect certain values for the design parameters. If the designer proposes a value different from the expected value, the apprentice asks the designer for justification regarding these differences. However, ADD's rationale transfer flow is not one way from the human designer to the computer system. The designers can also ask the apprentice to provide a possible design parameter value and show the rationale behind that particular value.

ADD presents DR by using the following four computer display output areas: Dependency Graph, Design History, Design Impacts, and Local Evaluation. The Dependency Graph shows graphically the network of design parameter relationships relevant to the current

rationale query. The Design History screen shows linearly the sequence of actions the designers took while creating their design using ADD. Design Impacts is a scrollable text box showing a summarized description of the effects of a change in the current design state. Local Evaluation allows designers to explore in detail how a specific design decision came into being.

2.3.5.2 ADD+

ADD+ (Garcia and de Souza 1995) is a modification of the rationale retrieval module of ADD. ADD+ improves on the rationale presentation capabilities of ADD by generating natural language text descriptions. ADD+ takes information from ADD's four output screen areas, ADD's knowledge base, and the dialogue flow between the computer system and the user. ADD+ maps these information sources into **R**hetorical **S**tructure **T**heory (RST) Schemes. The RST Schemes are a set of rules controlling how, what and when to present the natural language text description of the design rationale.

2.3.5.3 Collaborative Axiomatic Design Support (CADS)

CADS (Favela et al. 1993) is a DR system supporting the communication and coordination between multiple building designers. CADS contains a graphical interface to generate DR based on the **A**xiomatic **D**esign **L**anguage (ADL). ADL uses the following constructs: **F**unctional **R**equirements (FR), **D**esign **P**arameters (DP), Relationships, Claims, and Groups. The relationships Satisfied-by and Requires relate FR and DP. The relationships Sub-requirement and Sub-design-parameter create hierarchies for FR and

DP. The relationship Version-of associate different alternatives of DP. Claims relate to other constructs by the relationships Supports and Denies. A Group is a collection of constructs to associate a relationship to it.

2.3.5.4 Design Rationale Authoring and Retrieval System (DRARS)

DRARS (de la Garza and Ramakrishnan 1995) focuses on the building construction domain. DRARS is a passive DR system that relies on the human user to make use and interpret the captured rationale. It uses the computer as a database search engine to present DR information. It uses a variation of the Lee and Lai (1991) model of DR. DRARS uses objects called Views, Goals, Alternatives, Claims, Questions, Answers, and Versions. The human user is responsible for giving descriptive and meaningful names for these objects. It is also up to the human user to store the appropriate information in these objects and to create the proper relationships between these pieces of information.

2.3.5.5 Reviewer's Assistant

The Reviewer's Assistant (East et al. 1995) assists in the design review process. Although this system does not capture rationale during the actual design process, it captures rationale during a design review. It uses a case-based reasoning approach. It is essentially a growing database of comments about the various building components of several different projects. It uses classification scheme similar to the UNIFORMAT system to group these comments for easier searching of the database. If a reviewer has a specific design review comment on a particular building project, he or she can search the

comments database for a similar one. This removes the need to retype a review comment. Also, if the stored comment has a solution implemented in a prior project, the system can assist in the review process by providing a possible solution to the current problem.

2.3.5.6 Knowledge Acquistion Language (NaCl = SALT)

SALT (Marcus and McDermott 1989) is a knowledge acquisition program for creating rules tailored for expert systems that check the validity of a parameter against a set of constraints. SALT allows users to create and maintain expert systems by checking the consistency of a new piece of knowledge with the existing knowledge base. Users therefore need not worry if the new bit of knowledge they are currently entering into the system is 100% compatible with the knowledge base. SALT keeps track of how the various pieces of knowledge fit together. SALT warns the user of any inconsistencies that might occur because of the new knowledge addition. SALT then provides mechanisms allowing users to resolve the inconsistency, thereby ensuring the integrity of the knowledge base. One example of an expert system generated and maintained with SALT is VT, an elevator design expert system for Westinghouse Elevator Company.

2.3.5.7 SHARED-Design Recommendation and Intent Management System (SHARED-DRIMS)

SHARED-DRIMS (Peña-Mora et al. 1995) is a part of the Distributed and Integrated environment for Computer-aided Engineering (DICE) project. DICE uses a network of computers having a global object-oriented database and a distributed control mechanism

SHARED-DRIMS captures DR by using a design product model, a comprehensive knowledge base about the design product, and a user-interaction module. It checks design decisions for consistency with the stored DR information in the computer database It immediately notifies all interested participants of a design inconsistency, thereby mitigating the negative effects of a design conflict.

2.3.5.8 Skull Object Space (SOS)

SOS (de la Garza and Oralkan 1995, 1992) is a DR capture system geared towards enhancing building construction cost estimating. It captures DR by linking functions and requirements with the physical characteristics of a design element. SOS consists of six object networks, called libraries (Figure 2.4). The SOS libraries store both design and DR information The Building Typology Library, Space Typology Library, Building Element Typology Library, and Building Component Library contain classes that are generalizations or parents of the various objects representing the design. For example, in Figure 2.5 the instance `XYZ.Building` is an instance of the Building Typology Library object `Office.Building`. The Specifications Library has objects containing information about the required parameters for a particular design object. These parameters form the basis for selecting the most suitable object from the Material and Equipment Library. The Material and Equipment Library contains objects representing a wide variety of commercially available building materials and equipment.

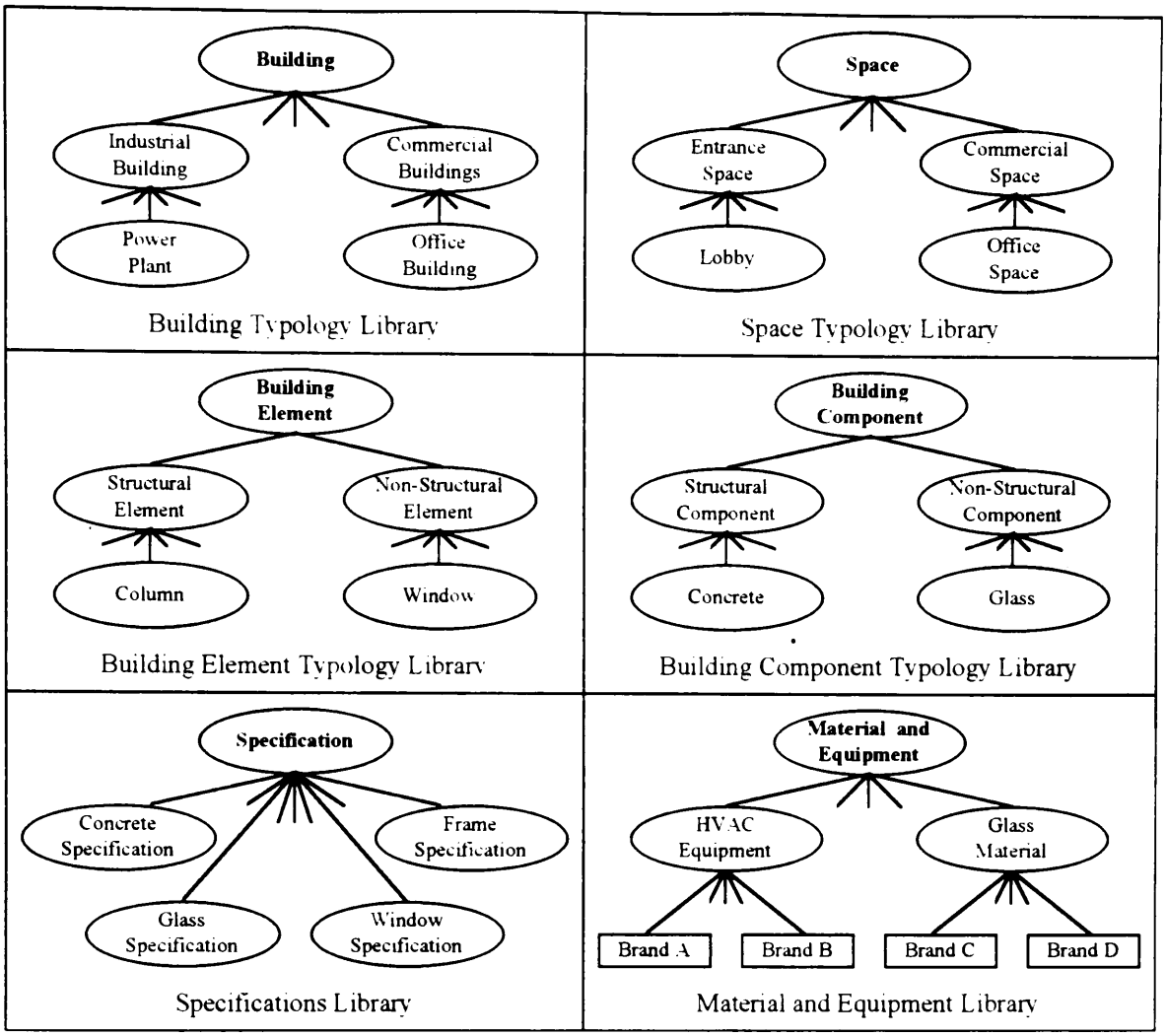


Figure 2.4 SOS Object Libraries (adapted from de la Garza and Oralkan 1992)

SOS uses a rule base to perform inferences on this object-oriented data structure. The rule base and its inferences constitute DR within SOS. Figure 2.5 indirectly shows how SOS externalizes DR information. It uses a four level approach in capturing DR. The following list describes briefly the DR information captured in each of these levels.

1. Global Building Level - functions and user requirements of the building.

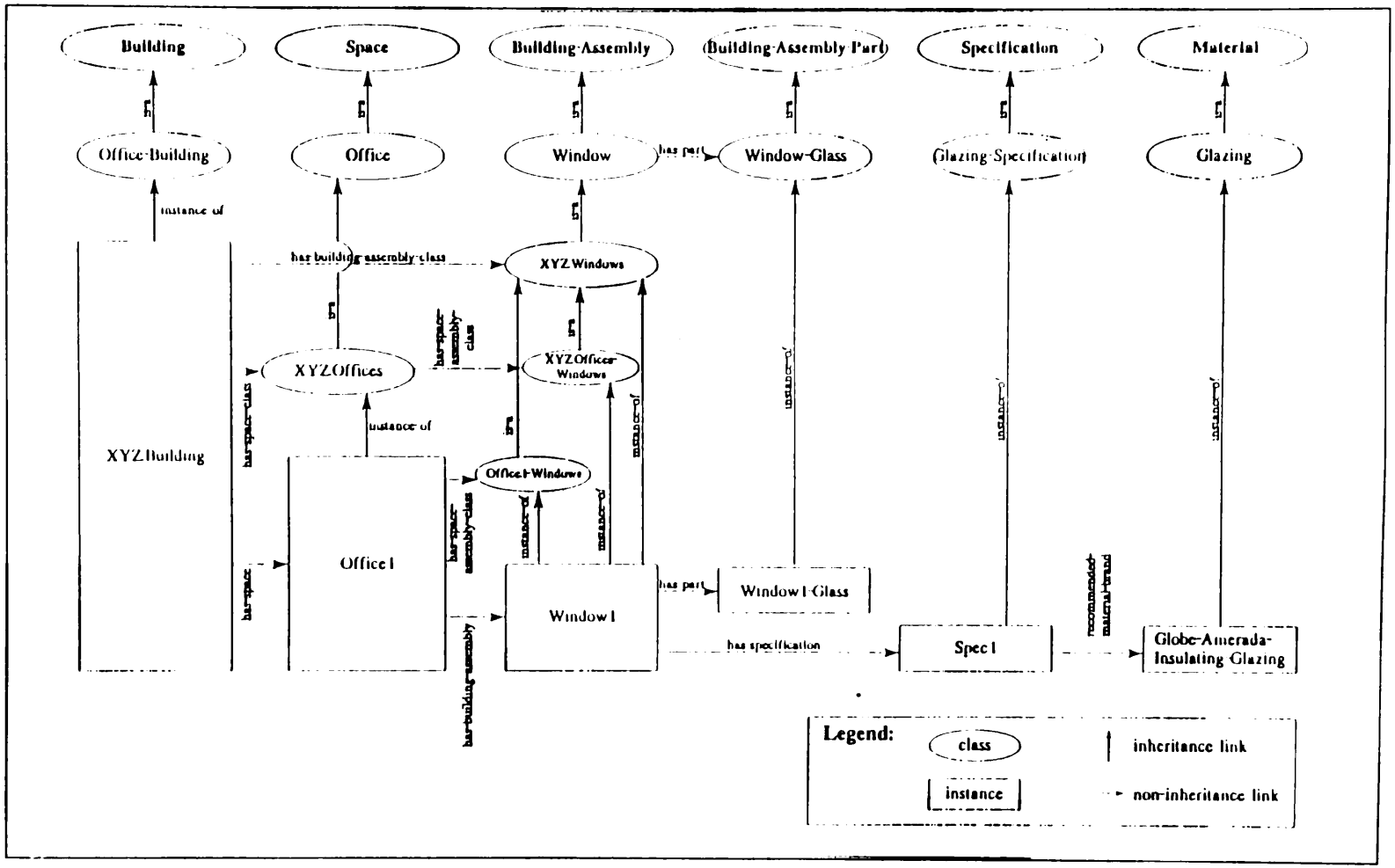


Figure 2.5 SOS Design Rationale Mechanism (taken from de la Garza and Oralkan 1995)

2. Space Level - functions and user requirements of building spaces and their relationships to the functions and user requirements of the building.

3. Building Assembly Level - performance information about the various building assemblies and their relationship to the functions and user requirements of the spaces containing these assemblies.

4. Building Component Level - physical property information about the various building components and their relationship to the performance information of the assemblies containing these components

SOS maintains links relating lower level DR information with higher level DR information. This allows users to trace DR information from the lower level material specification all the way up to the global building level function definitions. A companion application to SOS, the Material Application Module assists the contractor in specification interpretation, material selection, and material substitution.

2.4 ARCHITECTURAL PROGRAMMING

2.4.1 Definition of Architectural Programming

Design is the second stage in the life cycle of a facility, as Figure 1.1 shows. The concept definition stage precedes the design stage. For building projects, architectural programming is the term given to the concept definition stage. Architectural programming is the process of defining the architectural problem, while design is the process of creating a solution for this problem (Evans and Wheeler 1969, Peña 1977,

White 1972). Design (and consequently, DR) depends a great deal on the architectural program document.

An architectural program document is an organized presentation of the background information, fact analysis, evaluations, and conclusions pertinent to the building project. Various owners and architects have varying views on what types of information goes into an architectural program. Asides from the traditional information specific to a particular building (for example, building type, building size, location), an architectural program can include information such as a master plan (for example, assessing present conditions, projecting current trends, and outlining future potentials regarding the owners' building development), a feasibility study (for example, analyzing the economic timing of the building project), and an operations analysis (for example, staffing projections, equipment purchases). Thus, it is up to the owners and architectural programmers to agree upon the contents of an architectural program (Evans and Wheeler 1969, White 1972).

Architectural programming deals with the owners' goals, aspirations, wants, needs, ideas, and resources for the building. As such, a major chunk of the decision making responsibility lies with the owners. Architectural programmers act as stimulants in forcing owners to make decisions. These decisions greatly reduce the number of design solutions, thereby saving time and money. Architectural programmers process the large amount of raw data on the owners' goals, aspirations, ideas, and resources into useful information on the owners' wants and needs. Often, owners have a difficult time differentiating between

wants and needs. It is the job of the architectural programmer to assist owners in the differentiation process. Owners typically have a large amount of wants while having a limited amount of resources (for example, money, time, land). Needs are a smaller subset of the owners' wants given the owners' available resources. Design seeks to transform a set of owner wants and needs into a set of drawings and specifications that allow builders to construct a structure (Evans and Wheeler 1969, Peña 1977, White 1972).

2.4.2 Architectural Programming Information

Architectural programming is an organized process composed of the following five steps (Peña 1977 p 25):

1. Establish Goals - What do the owners want to achieve? Why?
2. Collect Facts - What is it all about?
3. Uncover Concepts - How do the owners want to achieve the goals?
4. Determine Needs - How much money, space, and quality?
5. State the Problem - What are the significant conditions and the general directions the design of the building should take?

Theoretically, architectural programmers should follow these five steps in numerical order. However, in actual practice, architectural programmers sometimes perform the first four steps in a different sequence or even concurrently. The fifth step, stating the problem, is always the last step. It is only after collecting as much information as possible about the owners' goals, ideas, and resources can architectural programmers begin the process of

defining the design problem. Thus, it is the problem statement that links architectural programming stage with the design stage (Peña 1977).

Architectural programmers define the design problem using four considerations: Function (People, Activities, Relationships), Form (Site, Environment, Quality), Economy (Initial Budget, Operating Costs, Life Cycle Costs), and Time (Past, Present, Future). Figure 2.6 shows the relative densities of information contained in each step-consideration combination. The first three steps contain the largest amount of information but in a widely scattered, totally unstructured manner. It is in these first three steps that the architectural programmer extracts information from the owners. The architectural programmer also organizes, classifies, and displays this large body of information in a manner suitable for discussion and decision making. The fourth step shows a denser circle reflecting the growing number of decisions made by the owners. These decisions provide structure to this set of information. The fifth phase contains a very dense circle representing an architectural programming document containing a clear statement of the design problem representing a very organized collection of information free of irrelevance (Peña 1977). The amount of information continues to accumulate from the first step to the last step. Information density increases due to the increasingly concise manner of information presentation.

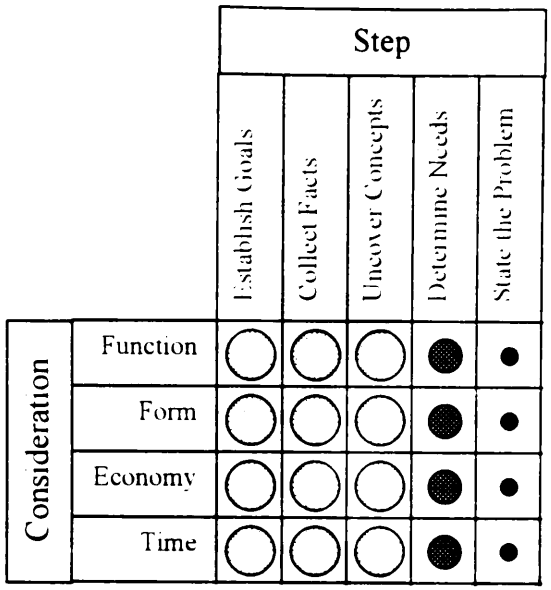


Figure 2.6 Architectural Programming Information Densities

2.4.3 Comparison of Design Rationale and Architectural Programming

Design rationale, as discussed in Section 2.3, is a collection of information behind the design drawings and specifications of a facility. These design drawings and specifications are the end products of the schematic design and design development stage of architectural design. These design documents are to design development as an architectural program document is to the architectural programming stage. Thus, design drawings and specifications are highly dense information modules similar to the architectural program document shown on the last column of Figure 2.6. The process of capturing and storing DR is similar to the extraction and organization of information during architectural programming.

The architectural program, being the link from the architectural programming stage to the design stage, is one of the foundations of a DR document. The architectural program lists all the requirements of the owners for the building project. The design team (architectural, structural, mechanical, electrical, and civil engineers) seek to transform these requirements into design drawings and specifications for use by builders. During this transformation process, the source behind some of the numerous design decisions is the architectural program. However, the architectural program, in itself, contains a great deal of implicit rationale. An architectural program *rationale* document allows the possibility of tracing the rationale behind design drawings and specifications even further than the architectural program document, to the very basic premises (for example, owners' goals, aspirations, ideas, and resources). However, capturing rationale during the architectural programming stage is beyond the scope of this research effort.

One final difference between architectural programming and DR lies in the focus of these concepts. Architectural programming focuses on the building as a whole. It shows the relationship of the building description to the goals, ideas, and resources of the owners. On the other hand, DR focuses on the various parts (for example, spaces, rooms, walls, equipment) of a building. DR shows the relationships existing among the various parts of a building, as well as the relationships existing between some of these parts to the description of the building (architectural program).

2.5 VALUE ENGINEERING

2.5.1 History of Value Engineering

Lawrence Miles, while working as an engineer for General Electric, developed the concept of Value Engineering (VE) during World War II. The critical material and labor shortages due to the war forced the company to resort to using alternative materials and manufacturing methods. General Electric's management noted that these measures often reduced costs and/or increased product quality. They gave Miles the task of developing a technique to systematically achieve these significant improvements. Miles called this system Value Analysis (VA). In 1954, the U. S. Navy Bureau of Ships started applying the VA process to its procurement activities, calling their process VE. Today, the terms, value analysis, value engineering, value management, and value control are synonymous with each other (Dell'Isola 1982, Value Engineering 1991, Zimmerman and Hart 1982).

2.5.2 Definition of Value Engineering

VE is an organized method of analyzing the functions of a system to generate alternatives maximizing the ratio of the provided level of service to its associated life cycle costs. Level of service refers to the performance, reliability, quality, safety, and aesthetic requirements of the owners and/or end users. Life cycle costs refer to the total estimated expenditures on the system from its conception up to its demolition. There are two ways of maximizing this ratio. One method is to lower the life cycle cost while maintaining the same level of system service. The other method seeks to maintain the cost level while increasing the level of service.

2.5.3 VE Job Plan

A VE study is a systematic method divided into several phases. There are several variations of this formal VE job plan. The Environmental Protection Agency uses a six-phase job plan composed of Information, Creative, Analytical, Investigation, Recommendation and Implementation. The General Services Administration uses an eight-phase plan composed of Information, Functional Analysis, Creative, Judgment, Development, Presentation, Implementation, and Follow-up (Zimmerman and Hart 1982). These other variations of the VE job plan either group together or break apart into greater detail the following six-phase job plan used for this research:

1. Information Gathering - gathering the details of the design as well as the reasoning leading to the creation of this design
2. Speculation - brain-storming for VE suggestions
3. Analysis - selection of more feasible VE suggestions for further development
4. Development - further examination and design development of VE suggestions
5. Presentation - presentation of feasible VE suggestions to the owner for final approval regarding implementation
6. Post-Occupancy Evaluation - checking whether the implemented VE suggestions resulted in life cycle cost savings.

The first five phases are the same as the five-phase VE job plan (Value Engineering 1991) used by OVEST. The sixth phase comes from Kirk (1989).

The most important part of the VE job plan is the Presentation phase. It is in the Presentation phase where the VE team submits their VE proposals to the owners for possible implementation. Even if the VE team performs conscientiously the other VE phases, these efforts are all useless if the owners do not accept any of the submitted proposals. Thus, the Information, Speculation, Analysis, and Development phases must focus towards a successful implementation of proposals. Finally, there is obviously no Post-Occupancy Evaluation phase if the owner does not implement the VE proposals.

2.5.4 FAST Diagrams

Central to the VE concept is the analysis of functions. As such, in 1965, Charles Bytheway developed a technique called **F**unctional **A**nalysis **S**ystems **T**echnique (FAST). FAST seeks to stimulate the creative thinking process by focusing on the core functions of a system or product (Bytheway 1992). Figure 2.7 shows a diagram relating the various functions of the McAlpine Locks Support Facilities (OVEST 1995). A FAST diagram is the name given to these types of diagrams.

All elements of a FAST diagram have a verb-noun pattern (for example, verb = Maintain and noun = Lock Operations). This verb-noun pattern helps in creating a functional breakdown. The left-to-right relationships of the FAST diagram answer the "How do you *verb noun* " question, while the right-to-left relationships answer the "Why do you *verb noun* " question. For example, asking the question, "How do you *Maintain Lock Operations*?", enables value engineers to formulate more specific system functions

like *Test Miter Gates*, *Repair Miter Gates*, *Store Miter Gates*, and *House Operations Personnel and Equipment*. To check the consistency of the FAST diagram, value engineers often ask the "Why" question on a more specific system function. For example, the answer to the question "Why do you *Test Miter Gates*?" is *Maintain Lock Operations*. The generation of a functional breakdown of a system directs the value engineering team's creative thinking efforts to the formulation of function-based alternatives as opposed to a simple cost reduction alternative that may or may not be suitable.

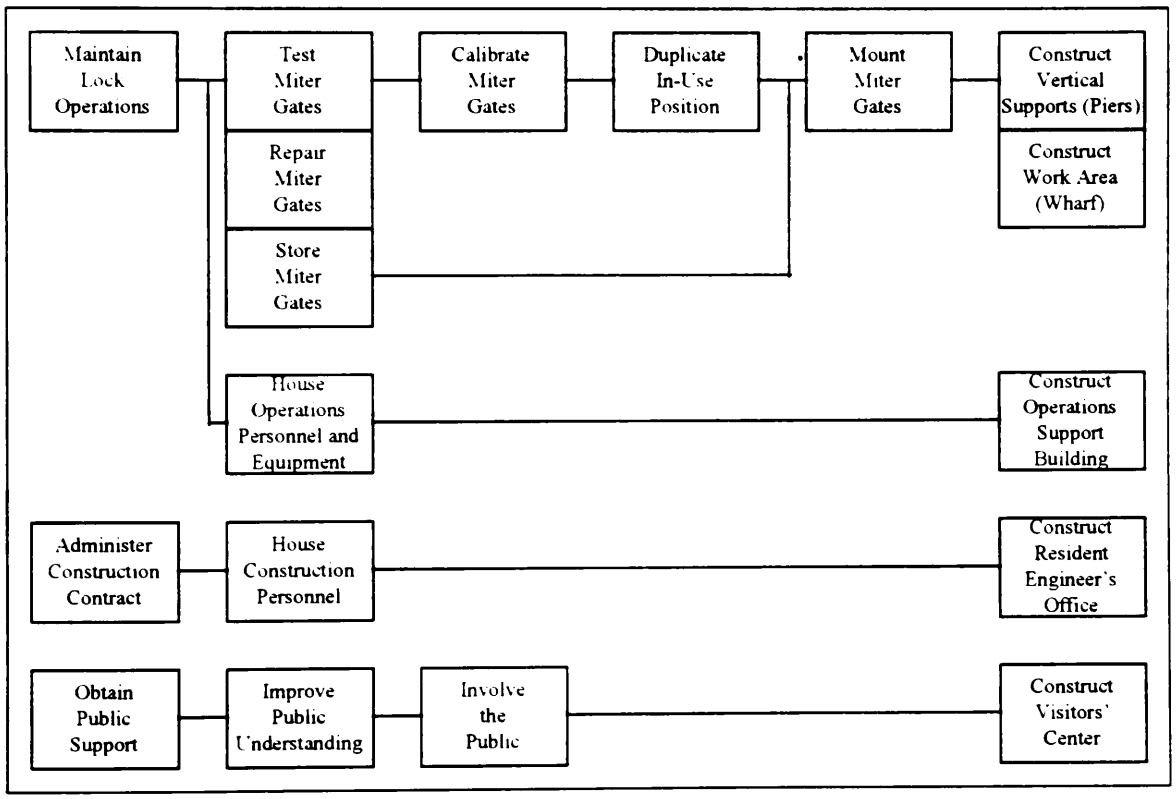


Figure 2.7 FAST Diagram Example (from OVEST 1995, Appendix D)

2.5.5 VE and Constructability

There is a close relationship between VE and constructability. The Construction Management Committee of the ASCE Construction Division defines constructability as "the application of a disciplined, systematic optimization of the construction-related aspects of a project during the planning, design, procurement, construction, test, and start-up phases by knowledgeable, experienced construction personnel" (Construction Management Committee 1989). Both VE and constructability seek to improve the economic performance of facilities (Russell et al. 1994).

The use of the **H**ierarchy of **O**bjective **T**echnique (HOT) diagrams (O'Connor et al. 1991) is another proof of the overlap between VE and constructability. HOT diagrams are very much similar to the VE FAST diagrams. Their main difference is that HOT diagrams concentrate on objectives while functions are the focus of FAST diagrams. The use of HOT diagrams improved highway construction specifications by structuring objectives, strategies, tactics, and solutions.

The main difference between VE and constructability lies in their primary objective. VE seeks to optimize the total life cycle cost of a facility, while constructability seeks to optimize the facility's construction cost. Another difference between VE and constructability is the timing of their studies. VE studies occur only at discrete, specific points during the life cycle of a project (for example, 30% design completion, 60% design completion). Constructability, on the other hand, ideally starts during the conceptual

planning stage and continues throughout the detailed design, construction, and start-up stages of a project (Russell et al. 1994).

2.5.6 Current Industry Practices

The construction industry typically applies VE studies during the design and/or construction phases of a project. There exists a greater potential for life cycle cost savings, the earlier the VE study takes place in the life cycle. Owners hire VE specialists to conduct VE studies during the design phase. Owners typically pay the VE specialists a fixed fee for their work. During the construction phase, owners sometimes encourage contractors to submit **VE** **C**hange **P**roposals (VECP). If the owners agree to implement a VECP, the owners and contractors share any construction cost savings resulting from the VECP (Code of Federal Regulations 1991, Dell'Isola 1982, Zimmerman and Hart 1982).

2.5.6.1 OVEST VE Process

The remainder of this section describes the VE process as practiced by OVEST. A group of fifteen to twenty people representing owners, designers, and value engineers comprise the VE team responsible for the Information Gathering, Speculation, Analysis, and Development phases of the VE study. This team completes these four phases in a period ranging from one to five days depending on the complexity of the study project.

The VE study begins with the introduction of the team members. Project planning personnel, regulatory personnel, and end users represent the owner. Architects, civil,

structural, mechanical, and electrical engineers involved in the design development comprise the designers' group. OVEST is a group of full-time architects and engineers experienced in their respective fields and VE. After the introductions, the VE study coordinator gives a brief description of VE. This overview helps clarify misconceptions and remove apprehensions about VE for team members having no VE experience.

2.5.6.2 OVEST Information Gathering Phase

The VE Information Gathering phase starts with the coordinator asking an owners' representative to present an overview of the project, its requirements and special considerations. The coordinator then presents the FAST diagram and the cost models for the project. This identifies the functional breakdown and high cost areas in the design, thereby allowing the VE team to focus their efforts on these areas.

The VE team then discusses the design. OVEST selects a subset from the current design drawings before the VE study. This subset representatively shows the major components of the project. The team processes this selected set in the following manner:

- the study coordinator first gives an overview of the contents of the drawing.
- the most knowledgeable person regarding the drawing from the designers (for example, the mechanical designer and the mechanical layout drawing) explains further the development and the rationale behind the design.
- the entire group then discusses the design. The owners' representatives further clarify various issues such as project requirements, site conditions, and material

availability. Other designers also explain the interaction between the drawing being discussed and their design (for example, how the mechanical layout affects the structural design). OVEST tries to get the owners and the designers to explain their requirements and design development more thoroughly by asking them specific questions. OVEST uses a strategy in this explanation gathering task. The OVEST members asking the most questions are not the most knowledgeable OVEST member on that topic (for example, while the mechanical drawings are being discussed, OVEST's architects, civil, structural, and electrical engineers ask the most questions). This allows the OVEST's most knowledgeable member on the current topic to evaluate the designers' explanation and take notes. Further, this makes implicit design reasoning to become explicit because the other members can ask questions that seem "very simple" to the experts. Clarification and explanation, not criticism, of the design is the main purpose of these discussions.

2.5.6.3 OVEST Speculation Phase

The study coordinator initiates the VE Speculation phase by presenting these guidelines:

- use creative thinking in speculating alternatives,
- there will be no idea to be ruled out, and
- there will be no analysis nor criticism of ideas in this phase.

The entire team then starts suggesting alternatives and explaining briefly its rationale. The coordinator writes down each these speculations for further analysis in the next phase. This process goes on until the team runs out of ideas.

2.5.6.4 OVEST Analysis Phase

The VE team evaluates each of the speculated alternatives in the Analysis phase. The team discusses the advantages and disadvantages of each of these alternatives. The person suggesting the alternative starts the discussion by explaining the alternative further. The other team members then contribute their thoughts on the ideas' merits or demerits. Although the focus of the entire team is on providing life cycle cost savings, different groups take different perspectives. The owners' representatives emphasize project requirements, the design group, design evolution and constraints, and value engineers, project functionality. The length of the discussion varies from five to thirty minutes per proposal. The team then decides whether or not to pursue the idea in the development phase. In some cases where the idea needs further research, the team defers giving judgment on the alternatives' fate.

2.5.6.5 OVEST Development Phase

The VE team splits up into smaller teams in the VE Development phase. These sub-teams process any alternatives deferred for further study. If a deferred alternative seems promising, it joins the list of other alternatives slated for further development. These sub-teams also begin the detailed design process on the alternatives to quantify any possible cost savings. The VE team then return to their respective home offices and complete the development of alternatives. Many telephone conversations and correspondence between the team members characterize this phase.

2.5.6.6 OVEST Presentation Phase

OVEST presents their VE recommendations to the owners' representatives (project planners, regulatory personnel, and end users) in the VE Presentation phase. This phase begins with the coordinator giving a brief overview of the project and a summary of the savings possible through the implementation of the feasible VE proposals. OVEST then presents each one of the feasible VE proposals, its rationale, advantages, disadvantages, and cost savings. The entire group then discusses salient points of the various proposals. As the group debates the feasibility of accepting a particular proposal, modifications sometimes surface. The debate for each proposal lasts for five to thirty minutes depending on its controversy. An owners' representative with sufficient authority decides whether a particular proposal is (a) accepted as is, (b) accepted with some modifications, (c) not accepted, or (d) deferred for further study.

2.5.7 Computer Technologies Supporting VE

The purpose of this section is to present some of the work done by other researchers in using computers to support VE. Conducting a review of their research is a necessary step in ensuring that this dissertation research makes a contribution to the body of knowledge.

2.5.7.1 P/VEX

P/VEX (Gibbs 1989) is a prototype expert system supporting the information, speculation, and analysis phases of a VE study. Its domain area is on the design of missile systems. It

is a repository of successful VE alternatives applied to particular missile systems. It acts as a database search engine suggesting VE alternatives to other similar missile systems.

2.5.7.2 Expert System for Value Management in the Design of Office Buildings (ESVMDOB)

ESVMDOB (Shen and Brandon 1991) employs a blackboard type structure with two main knowledge sources, the **C**oncept **D**esign **A**nalysis **M**odule (CDAM) and the **S**chematic **D**esign **A**nalysis **M**odule (SDAM). CDAM deals with the conceptual definition of an office building. It assists in the generation of a FAST diagram. It also provides a checklist to stimulate the creative thinking process of users. SDAM seeks to optimize the schematic design of buildings by using cost estimating expertise to continually monitor life cycle costs as the design work progresses. It presents design items with a high savings potential and assists in the generation and evaluation of alternatives.

2.5.7.3 ValuExpert

The domain area of ValuExpert (Taher and Diekmann 1992) is school roofing systems. It gathers information about the current roof design through a set of question and answer dialogs, compares the current design with its database of roofing knowledge, and finally, presents and ranks possible design alternatives.

2.5.7.4 ValuLink

The World Wide Web is another form of computer technology, asides from expert systems, that can support VE. ValuLink (1995) is a database on the Internet developed to support businesses in using VE. It contains a growing library of successful studies, federal regulations, legal cases, and product information.

2.5.7.5 Sturges' Function/Allocation/Component Model

Sturges (1992) proposes a three-tiered computer representation structure to represent the relationship among the various functions of a particular design. The structure consists of (1) function blocks, (2) allocation list, and (3) component stack. The function blocks are similar to the nodes in a FAST diagram. However, the links between these blocks are not only the simple how/why relationships in FAST diagrams. New link types include information flow, temporal relationships, causal connections, alternatives, and revisions. The allocation list linked to a specific function contains design specifications, performance requirements, resources, and component data. The component stack contains a list of possible components (such as pipes, wires) that could satisfy a particular function. Although this system allows for a computer system to perform calculations on the data, at present it is still up to the human designer to interpret and make use of these relationships.

2.5.7.6 Three-Dimensional Modeling

The creation of object-oriented three-dimensional computer models of a project (Reinschmidt et al. 1991) is an example of current research on computer applications on

constructability. Columbia University, Stone & Webster Engineering Corporation and the National Science Foundation are conducting this research. The results from this research show that three-dimensional solid models:

- help eliminate design inconsistencies,
- are easier to construct as compared to two-dimensional drawings,
- generate more accurate construction schedules,
- provide help in construction planning and monitoring, and
- increase construction productivity

2.5.7.7 Graphical Computer Constructability Analysis (GCCA)

GCCA (Fisher and O'Connor 1991) uses an animation software to analyze a construction procedure showing the graphical relationships between the components, equipments, workers and tools. Semi-automated pipeline construction is an actual example of an application of GCCA.

2.6 BUILDING MODELING

The OOP style is suitable for creating a building model using design typologies (Coyne et al. 1990, de la Garza and Oralkan 1992, Gero 1990, Gielingh1988, Turner 1990, Wade 1992). Design typologies are hierarchical classification schemes of physical objects or abstract concepts. Object networks easily represent these hierarchical classification schemes. Three classification schemes, namely, the Skull Object Space (SOS) model, the AEC Building Systems Model, and the General AEC Reference Model have had some

impact on this dissertation work. Section 2.3.5.8 details the SOS model while the following two sub-sections describe the other two models.

2.6.1 AEC Building Systems Model

The AEC Building Systems Model (Turner 1990) organizes and models AEC building product data into a conceptual object network transferrable across different computer systems. It accomplishes this computer platform portability by using a neutral file format. This model is part of the **P**roduct **D**ata **E**xchange **S**tandard / **ST**andard for the **E**xchange of **P**roduct model data (PDES/STEP) standardization efforts of the **I**nternational **S**tandards **O**rganization (ISO).

The AEC Building Systems Model decomposes a building project into a collection of systems and system components. Figure 2.8 shows the AEC Building Systems Model enumerating all building systems. Each of the objects in this model contains attributes representing their various properties. An objective of the AEC Building Systems Model is to provide the framework for future **C**omputer-**A**ided **D**esign (CAD) systems. PDES/STEP envisions an object-based foundation for future CAD systems, as opposed to the drawing- or geometry-based foundation of current CAD (Computer-Aided Drafting) systems. By having objects as the foundation of CAD systems, users can view both spatial and non-spatial properties on an equal level. Current CAD systems emphasize spatial properties — "hanging" or "attaching" non-spatial properties on drawings or geometries.

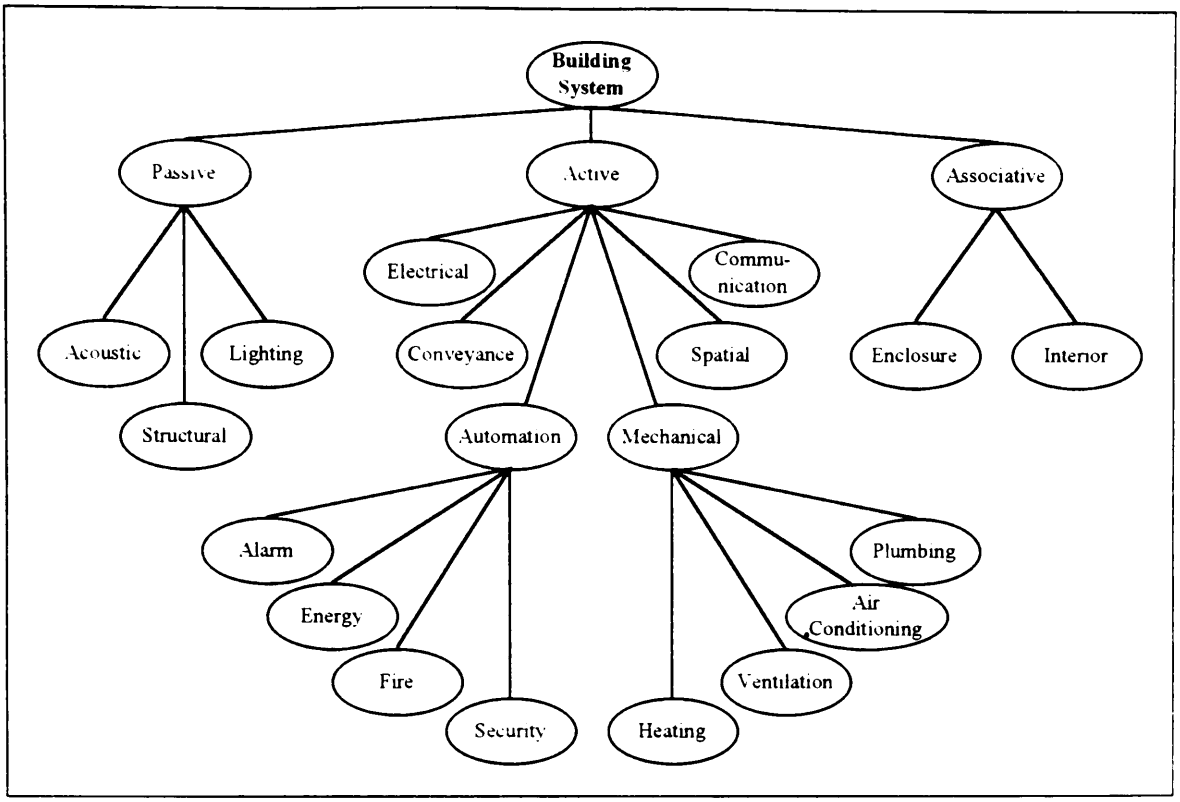


Figure 2.8 AEC Building Systems Model

2.6.2 General AEC Reference Model (GARM)

GARM (Gielingh 1988) is another model under development within the PDES/STEP standardization efforts. PDES/STEP defines the four layers of specifying product definition data (Figure 2.9). GARM provides a model for combining and relating the various product-type models under development for the AEC industry-type layer. GARM serves the needs of the whole AEC industry (second layer). The AEC Building Systems Model (Section 2.6.1) serves the needs of the AEC buildings industry (third layer). The fourth non-STEP layer contains entity descriptions defined outside STEP, such as national standards or company standards.

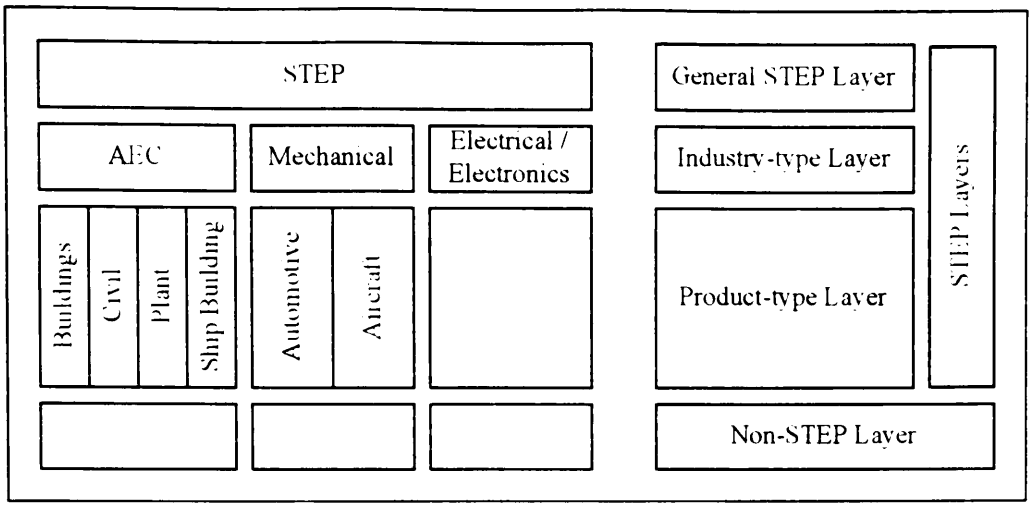


Figure 2.9 STEP Product Definition Data Layers

GARM uses a generic entity, called a **P**roduct **D**efinition **U**nit (PDU), to represent an entire product or a decomposition of the product. GARM uses three levels of PDUs, as shown in Figure 2.10. Generic PDUs have only parametric definitions (width w, height h, location l). Specific PDUs have values for some of these parameters (width 120 cm, height 200 cm). The PDU Occurrences are actual entities having all the parameters defined (width 120 cm, height 200 cm, location Room 101).

Each PDU has several characteristics. Each of these characteristics relates to an aspect. For example, two of the numerous characteristics of a `Building` PDU are `Illuminance Level` and `Reverberation Time`. `Illuminance Level` relates to the `Visual` aspect while `Reverberation Time` relates to the `Acoustical` aspect. In OOP parlance, characteristics are attributes, while aspects are groups of related attributes.

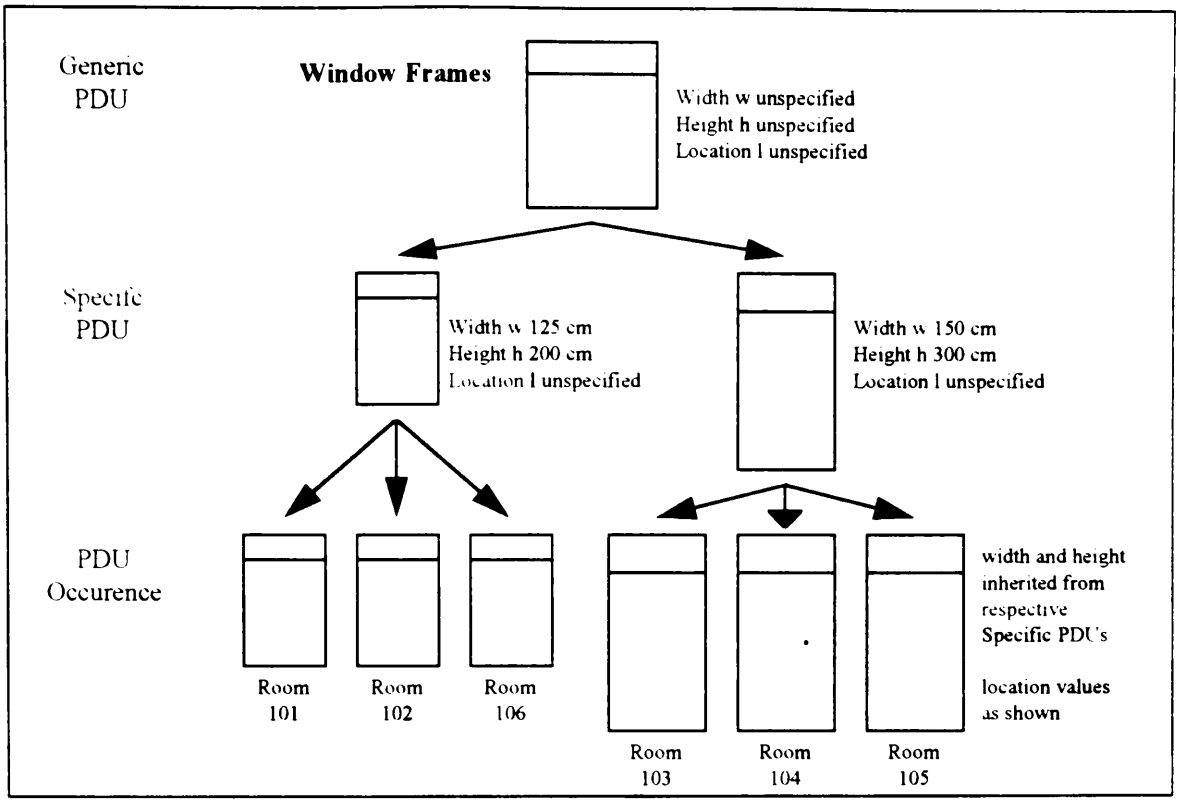


Figure 2 10 GARM PDU Example: Window Frames

GARM also makes use of life cycle phase distinctions for the various PDU's. Table 2 1 shows the relationship between the various life cycle phases, PDU sub-types, and types of characteristics. A Functional Unit describes the requirements imposed on the PDU. A Technical Solution represents a design satisfying Functional Unit requirements. Several Technical Solutions, or design alternatives can exist for a particular Functional Unit. In cases where there is no suitable Technical Solution, modification of the Functional Unit is necessary. A Planned Unit describes how to construct a Technical Solution. A Physical Unit represents how a PDU actually exists. In building systems, a Physical Unit PDU represents an as-built object. An Operational Unit describes the PDU in use. An Alteration Unit details the maintenance, renovations, and modifications made on a PDU.

A Demolition Unit stores information about the PDU during and after demolition. There are three types of characteristics, namely, Required, Expected, and Measured. The distinctions among these three types are evident from their names.

Table 2.1 Life Cycle Phases and PDU Sub-types

Life Cycle Stage	PDU Sub-type	Type of Characteristics
as required	Functional Unit	Required Characteristics
as designed	Technical Solution	Expected Characteristics
as planned	Planned Unit	Expected Characteristics
as built	Physical Unit	Measured Characteristics
as used	Operational Unit	Measured Characteristics
as altered	Alteration Unit	Measured Characteristics
as demolished	Demolition Unit	Measured Characteristics

2.7 COMPUTER AIDED DESIGN

The purpose of this section is to present the various directions of research in Computer Aided Design (CAD). This section presents three systems representing three directions of CAD research. JSpace integrates existing computer tools to store project specific information across the project life cycle. CIFECAD customizes a commercially available computer aided drafting package into a CAD package. ICADS uses artificial intelligence modules to support the graphical CAD process. These three systems have had significant influence on this research work.

2.7.1 JSpace

JSpace (Jacobus Technology 1996c) is an object-oriented development environment for manipulating project-specific data within the AEC industry. JSpace uses data from

various electronic sources to create an object-oriented representation of an AEC project. It has an interface module allowing users to make use of existing electronic data without excessive manual re-input of electronic data. JSpace has several interface modules for popular data formats within the AEC industry, such as, MicroStation, AutoCAD, dBase, Oracle, ASCII, and several others (Jacobus Technology 1996b, 1994). To illustrate, JSpace examines the graphical properties (for example, dimensions, location, level, color, symbol name, line weight) of the various elements (for example, lines, groups of lines, surfaces, solids) stored in a CAD data file to create an instance of a JSpace Class Library object (Jacobus Technology 1996e, 1995).

The JSpace Class Libraries (Jacobus Technology 1996b) contain descriptions of the properties (attributes) and behaviors (methods) of JSpace objects. The objects in these libraries serve as parents to the various instances representing an AEC project. These instances inherit the properties and behaviors of their respective parents. The JSpace Class Editor (Jacobus Technology 1996a) allows users to easily modify these class libraries without having to modify anything on the underlying JSpace program. One of the behaviors of class library objects defines how a newly created instance acquires values for most of its properties. This "instance creation" behavior may perform mathematical calculations, database queries, or knowledge base inferences to define the values of the various properties (Jacobus Technology 1996a). Other behaviors embedded in class library objects relate, group, and decompose objects, perform various operations when a

property value changes, and deliver information in other formats such as CAD drawings or database tables (Jacobus Technology 1996b, 1994).

Graphical representation is much more effective than textual representation of information contained in JSpace objects. This is especially important when conveying information to downstream life cycle phases, such as, construction and operations. The JSpace Viewer (Jacobus Technology 1996d) displays JSpace information using three-dimensional visualization and real time animation techniques. It also has the capability of accessing and presenting non-graphical information related to the graphical objects.

2.7.2 CIFECAD

CIFECAD (Ito et al. 1989) is an object-oriented CAD tool under development by Stanford University's **C**enter for **I**ntegrated **F**acility **E**ngineering (CIFE). It customizes the standard off-the-shelf AutoCAD package through additional menu items and functions. AutoCAD is essentially a drafting package — it draws lines and arcs. The custom CIFECAD menu items and functions transform AutoCAD into a design package. Instead of drawing lines and arcs, CIFECAD draws objects relevant to building projects, such as, beams, columns, walls, and doors. For example, using standard AutoCAD, designers draw two parallel lines to denote the plan view of a beam and enter its cross-sectional dimensions in a table of values. Using CIFECAD, designers create a beam by specifying its cross-sectional dimensions and its end points. The custom functions in CIFECAD create the three-dimensional graphical representation of the beam within

AutoCAD. CIFECAD uses AutoCAD's 3DPolyline function to create the graphical representations. CIFECAD also attaches non-geometric information, such as, material type and construction cost, to the graphical object representations. CIFECAD uses AutoCAD's extended entity data capabilities to attach text descriptions representing non-geometric information. These non-geometric information provide enhanced descriptions of the design objects.

The most notable conceptual limitation of CIFECAD is its inability to create sub-classes. In CIFECAD, all graphical design objects are direct instances of CIFECAD's pre-defined object types. This results in a lot of redundant data stored in the model. For example, a CIFECAD model containing three identical beams (Beam_1, Beam_2, and Beam_3) stores each of their respective geometric and non-geometric parameters even though they have similar values. Sub-classes provide a more efficient method of storing data. For example, the creation of a sub-class (Beam_Type_A) provides a storage space for the parameters inherited by design instances (Beam_1, Beam_2, and Beam_3).

2.7.3 Intelligent Computer-Aided Design System (ICADS)

ICADS (Pohl et al. 1989) leverages a CAD system with artificial intelligence tools to provide interactive feedback to the designer. ICADS consists of four components: a computer-aided drafting system, a Geometry Interpreter, a set of Design Knowledge Bases, and an Expert Design Advisor. The computer-aided drafting system is Accugraph Corporation's Mountain Top CAD package. It stores graphical information through

points and lines. The need to access directly the data structure of the CAD drawing has necessitated several modifications to this commercially available CAD package.

The Geometry Interpreter transforms the CAD drawing information (points and lines) into a set of design objects such as rooms, walls, and windows, among others. This module runs in the background, such that the designer does not notice the transformation of drawing information into design objects. To illustrate, when a designer draws a line dividing a space into two rooms, the Geometry Interpreter analyzes this drawing change, infers the purpose of the line, and modifies the object base (reduce the dimensions of the existing room and create a new room) to reflect the effect of the drawing change.

The Design Knowledge Bases assist designers in creating a comprehensive description about the evolving design solution. ICADS currently has only two knowledge bases — Building Type and Site Neighborhood. These knowledge bases, working together with the Geometry Interpreter defines parameter values for the various design objects. These parameter values are the driving mechanisms for the Expert Design Advisor.

The Expert Design Advisor consists of a set of **I**ntelligent **D**esign **T**ools (IDT) and a Blackboard Control System. The four IDTs currently available in ICADS are: Acoustic IDT, Thermal IDT, Lighting IDT, and Structural IDT. These IDTs are expert systems continuously monitoring and evaluating the evolving design solution. These IDTs suggest parameter values for the various design objects, check these values for consistency with its

design knowledge, and alert the designer whenever a design conflict occurs. The Blackboard Control System coordinates the concurrent activities of the various IDTs and controls the overall operation of the ICADS system.

CHAPTER 3

DATA STRUCTURES 0: OVERVIEW

The purpose of this chapter is to provide the reader with a frame of reference for the discussion in the next two chapters. This chapter presents a brief summary of a computer program architecture for a generic design rationale capture system. Chapters 4 and 5 further detail the concepts outlined in this overview. This chapter describes the capabilities and the reasons for selecting the specific computer programs used in the development of DRIVE. It also discusses two rationale capture styles. This chapter also describes several usability guidelines implemented on the DRIVE user interface. It also includes a discussion of the program structure of DRIVE. A description of an actual building project used as an example in subsequent chapters completes this set of introductory information.

3.1 DESIGN RATIONALE CAPTURE SYSTEM ARCHITECTURE

One of the advantages of OOP (Section 2.2) is modularity. OOP isolates different portions of a computer program from one another except at their connection points. Modularity allows programmers to maintain large computer programs. In modular

programs, modifying or upgrading a program module does not require modifications to the other modules. Applying the modularity concept to a DR capture system involves separating the portion the portion of the system concerned with the product description or design, and the portion concerned with the reasoning or rationale behind the design decisions. A generic DR capture system is another term for such a modular system. It is generic in the sense that users need not modify the rationale capture module for use with different product types or design domains.

Figure 3.1 displays graphically the architecture of a generic DR system. This dissertation uses the terms **K**nowledge **R**epresentation **M**odule (KRM) and **R**ationale **S**torage **M**odule (RSM) to refer to the product description and rationale capture modules, respectively. The three basic components of a generic DR system are a current context, a KRM pool, and a single RSM. The current context provides a working area allowing a single KRM and the RSM to interact with one another. Each KRM consists of a set of related objects representing a particular design domain. For example, a building construction KRM contains spaces, walls, and windows, while a highway construction KRM has roadways, medians, and guardrails. Although each KRM consists of different objects, these KRMs are still similar to each other because of their basic building block, namely, objects. This allows a single RSM to interact with different KRMs. The RSM stores the rationale behind the design decisions made about the various KRM objects. The RSM stores rationale as objects. The attributes of the RSM objects represent the various properties

constituting rationale, such as, who made a design decision, the owner requirements affecting a decision, and the relationship between a decision and other design objects.

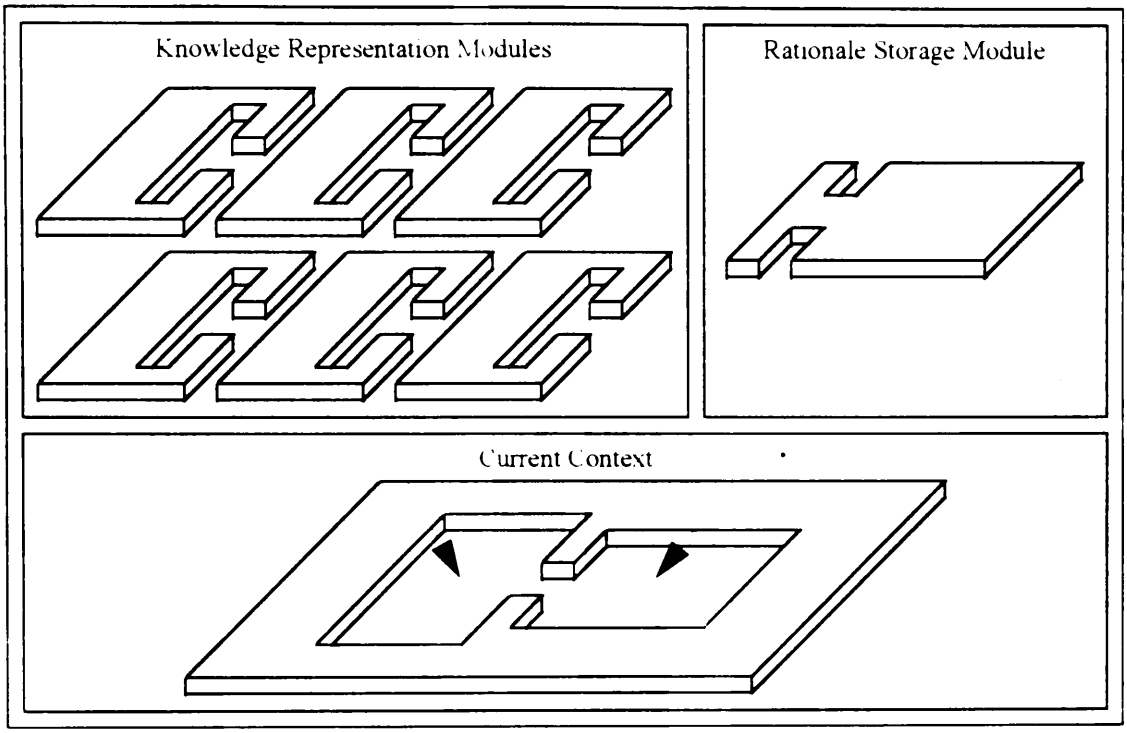


Figure 3.1 Generic Design Rationale System Architecture

3.2 PROGRAM DEVELOPMENT TOOLS

Figure 3.2 shows this dissertation's implementation of the generic DR system architecture shown in Figure 3.1. For current context portion of Figure 3.1, this dissertation uses DRIVE, which integrates two Microsoft Windows applications, namely, Kappa-PC (version 2.3.2) and AutoCAD (Release 12).

3.2.1 Kappa-PC

Kappa-PC (1994) is an object-oriented, graphical development environment. It stores all its data as objects in a hierarchical network. It supports the OOP concepts of objects, attributes (slots), methods, demons (monitors), inheritance, and message passing. The items in parenthesis are the Kappa-PC terms equivalent to the OOP terms used in Section 2.2. Kappa-PC also provides graphical development tools. These tools allow programmers to manage the hierarchical network of objects, create user interfaces, and link specific objects to particular interface objects. Kappa-PC supports a variety of customizable interface objects such as, windows, menus, buttons, list boxes, and scroll bars. Kappa-PC uses an interpreted language. This allows programmers to rapidly develop and test Kappa-PC code without having to compile the code into low-level machine instructions. Finally, Kappa-PC, being a Microsoft Windows application, can communicate with other Microsoft Windows applications through **D**ynamic **D**ata **E**xchange (DDE). Through DDE, applications can send data to, receive data from, issue commands to, and accept commands from other applications.

The various capabilities of Kappa-PC mentioned in the previous paragraph makes it suitable for developing the DRIVE application. Object networks are the foundation of DRIVE's data structures, thereby requiring an object-oriented development environment. DRIVE also requires the development of an interface for users to interact with the program. Kappa-PC's library of interface objects makes it appropriate for this task.

DRIVE is a proof-of-concept application. Proof-of-concept applications place more emphasis on program coding speed rather than program execution speed. An interpreter allows much faster program coding than a compiler. Interpreters allow programmers to type a piece of code and immediately evaluate its performance. Compilers, on the other hand, require programmers to recompile the program code into machine level instructions before executing the program to evaluate the effects of the modified program code. However, when considering program execution speed, the interpreters are at a disadvantage against compilers. Compiled programs, because of their low-level machine level instructions, achieve much higher program execution speeds as compared to similar interpreted programs. Finally, DRIVE needs to integrate an object-oriented development tool with a CAD package. Kappa-PC's DDE capability accomplishes this requirement.

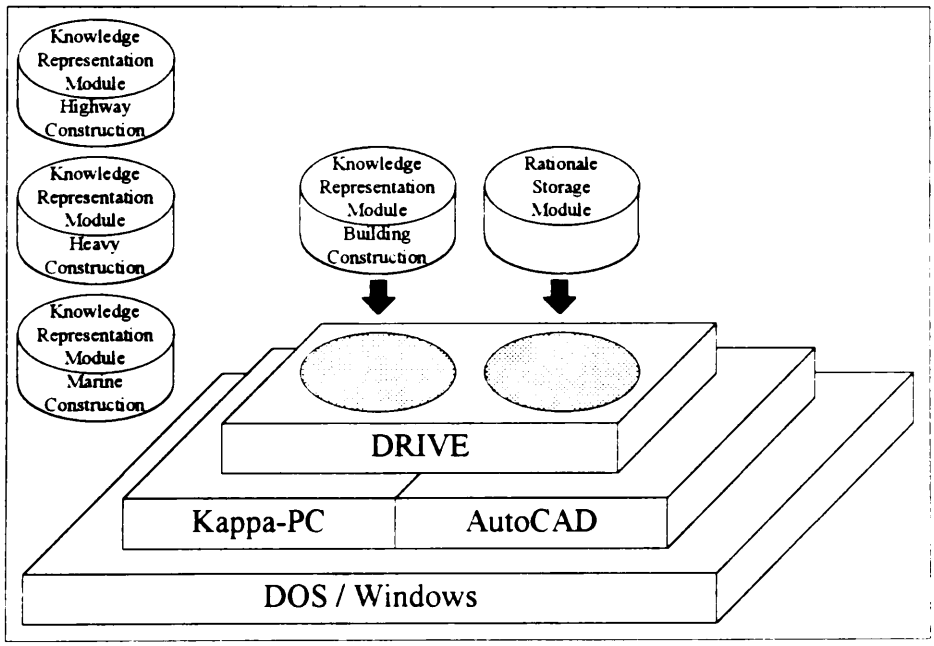


Figure 3.2 DRIVE Implementation

JSpace (Section 2.7.1) is also a logical choice for developing the DRIVE application. However, the capabilities described in Section 2.7.1 represent the 1996 version of JSpace. During the initial stages (i.e., 1992) of this research effort, JSpace was still in its infancy stage. This made JSpace less suitable then as a development environment when compared to Kappa-PC. However, as of the present (i.e., 1996), JSpace is a very capable development environment and is much better than Kappa-PC.

3.2.2 AutoCAD

AutoCAD (1992) is a general-purpose computer-aided drafting tool popularly used by the AEC industry. It allows users to draw two-dimensional elements such as lines, arcs, circles, rectangles, and planar surfaces. It also supports three-dimensional elements such as cubes, cylinders, spheres, wedges, and three-dimensional surfaces. As with the majority of the so-called CAD tools used by the AEC industry today, people are using AutoCAD as a drafting tool. Several researchers have already proposed extending the basic capabilities of a computer-aided *drafting* package to transform it into a true computer-aided *design* package (Ito et al. 1989, Jacobus Technology 1996c, Pohl et al. 1989).

AutoCAD Release 12 (1992) has two application interface languages, AutoLISP and **A**utoCAD **D**evelopment **S**ystem (ADS). AutoLISP is an implementation of the LISP programming language, while the foundation for ADS is the C programming language. Due to the nature of their underlying languages, AutoLISP is an interpreted language

while ADS is a compiled language. Both AutoLISP and ADS allow users to customize the AutoCAD environment. The customized functions created using AutoLISP or ADS transform AutoCAD from a *drafting* package to a true computer aided *design* package.

The two primary reasons for selecting AutoCAD are (1) it runs on the Microsoft Windows environment and (2) the availability of the source code for CIFECAD (Section 2.7.2). AutoCAD's DDE capability allows it to communicate with the selected object-oriented development tool, Kappa-PC. CIFECAD uses the AutoLISP programming language to provide building design capabilities to the basic AutoCAD package (Ito et al. 1989). For example, designers using the basic AutoCAD package draws two parallel lines to represent a beam element in plan view. With CIFECAD, designers type in the beam's cross-sectional properties (as well as other non-geometric properties) and select the CIFECAD-created column objects supporting the beam. CIFECAD then uses AutoLISP functions to create a three-dimensional representation of the beam. The AutoLISP source code for CIFECAD serves as the model for the AutoLISP code of the DRIVE application.

3.3 RATIONALE CAPTURE STYLES

The DRIVE prototype system captures the owners' and designers' rationale behind the creation of a design. The DRIVE prototype allows users to create a three-dimensional building model. It is also a highly graphical application assisting in the documentation of design decisions. DRIVE can capture design rationale in real-time or in a retrospective

basis. Figure 3.3 illustrates these two rationale capture styles. DRIVE allows real-time rationale capture by helping users document their designs concurrent with the design development. Retrospective rationale capture involves taking a completely designed project, recalling the rationale behind the design decisions, and using DRIVE to attach these rationale onto the design objects. This allows users to attach design rationale to existing two-dimensional or three-dimensional design drawings.

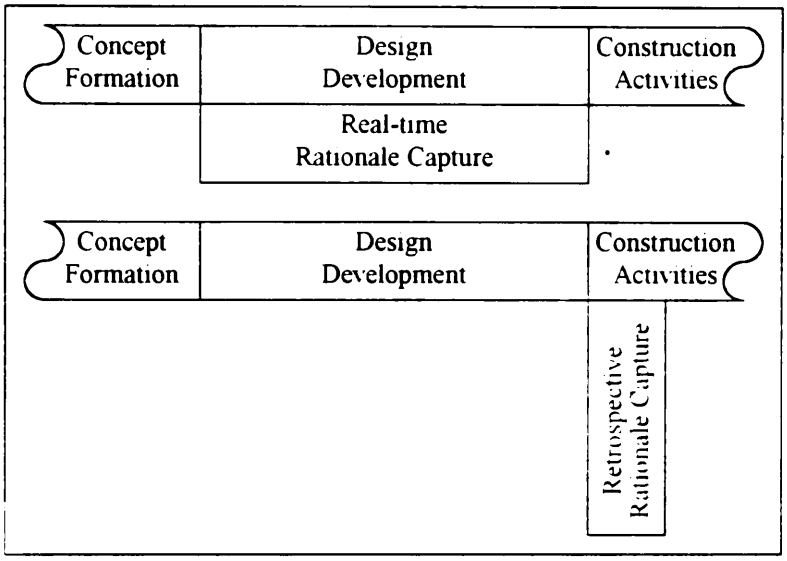


Figure 3.3 Rationale Capture Styles

3.4 DRIVE USABILITY GUIDELINES

There is a basis for the particular characteristics of the DRIVE user interface. DRIVE implements several usability guidelines outlined in Hix and Harston (1993). Some of the implemented guidelines are

- Maintain display inertia. Display inertia means that interface objects must appear at the same position and not move from one position to another.
- Make user actions easily reversible.
- Keep locus of control with the user. The computer system should not dictate the sequence of actions a user must perform to accomplish a certain task.
- Prevent user errors. This involves disabling invalid choices, thereby minimizing the use of error messages.
- Use user-centered wording. Using user-supplied names like `Partition Wall 1` instead of `Object_32` enhances the usability of the system.
- Be consistent. The actions of the various interface objects should not vary from one screen to another.
- Support recognition rather than recollection. This involves providing visual cues to assist the user perform a task.

3.5 DRIVE PROGRAM STRUCTURE

The purpose of this section is to present an overview of the internal module structure of DRIVE. Readers not interested in the implementation details of DRIVE can proceed directly to the next section without adversely affecting their ability of following the discussion in the succeeding chapters. Figure 3.4 shows the various internal modules of DRIVE. The *.kal files shown in Figure 3.4 denote Kappa-PC source code files. All other files are AutoCAD-related files.

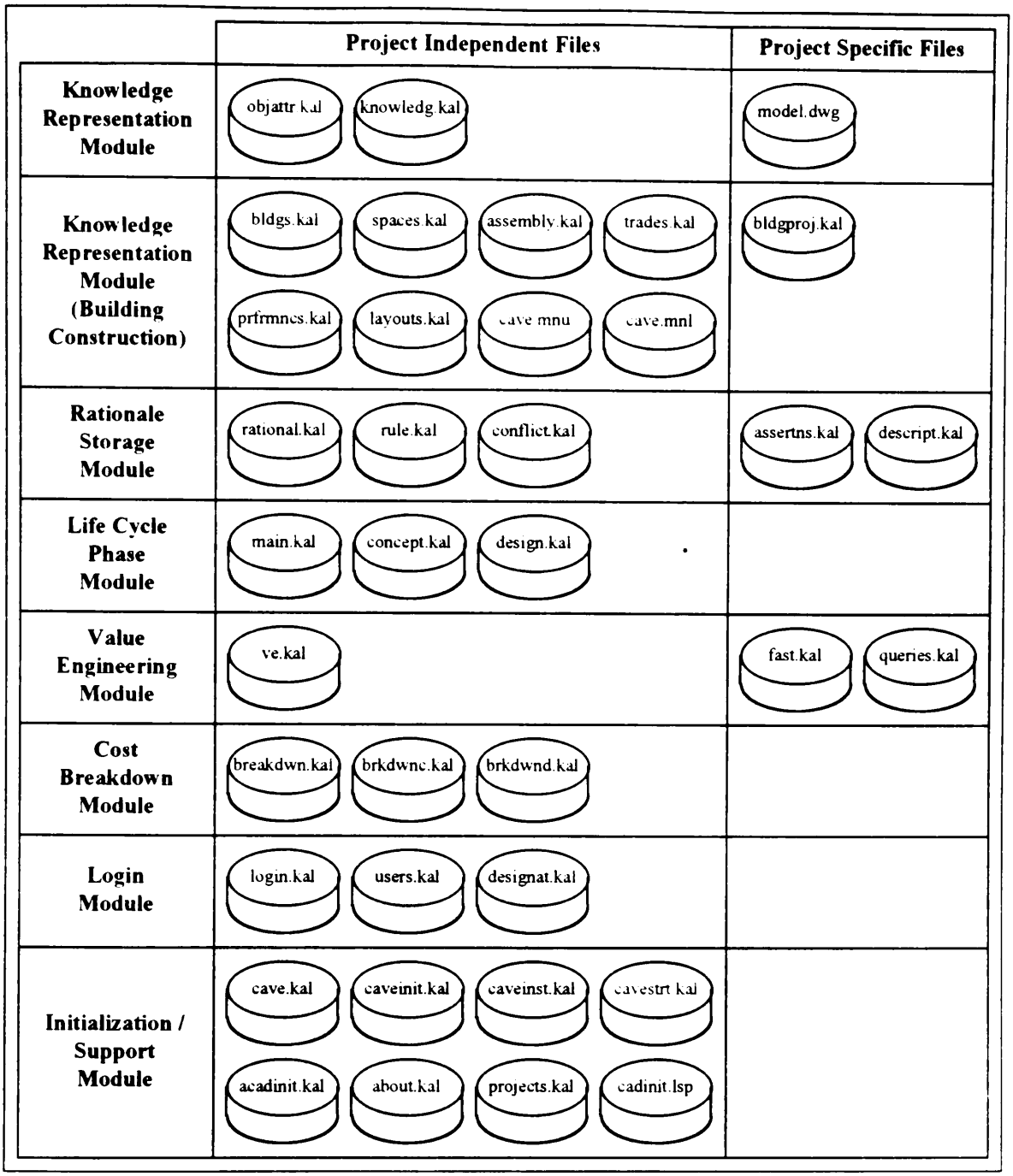


Figure 3 4 DRIVE Program Files

DRIVE has two classification schemes for its internal file structure. One classification scheme groups files according to whether or not they contain data specific for a particular project. The top row of Figure 3.4 shows the two groups under this classification scheme. The other classification scheme groups files according to their general functions. The leftmost column of Figure 3.4 shows the various groups under this classification scheme.

The Knowledge Representation Module has two portions — a domain-independent and a domain-dependent portion. The domain-independent portion of the KRM consists of files capable of performing their operations on different design domains. These files are:

- objattr.kal - handles the input of values for the attributes of the design objects
- knowledg.kal - allows modifications to the domain-dependent typological libraries
- model.dwg - the project-specific AutoCAD file containing the graphical representation of the design.

The domain-dependent portion of the KRM corresponds to the plug-in KRM modules show in Figure 3.2. For this dissertation, only the Building Construction KRM exists. The Building Construction KRM consists of the following files:

- bldgs.kal - contains a library of building types
- spaces.kal - contains a library of space types
- assembly.kal - contains a library of assembly types having a breakdown structure based on the various building systems

- trades.kal - contains a library of assembly component types having a breakdown structure based on the various building construction trades
- prfrmncs.kal - contains a library of performance objects grouping the related performance parameters of the domain objects
- layouts.kal - contains a library of graphical layouts for the domain objects
- cave.mnu - the AutoCAD menu definition file containing the customized menu items for the building construction domain
- cave.mnl - the AutoCAD menu support file containing the customized AutoLISP functions for the menu items defined in cave.mnu
- bldgproj.kal - the project-specific Kappa-PC file containing the design objects.

The Rationale Storage Module captures, stores, and performs operations on design rationale information. The RSM consists of the following files

- rational.kal - handles the capturing of design rationale information
- rule.kal - handles the creation of if-then-else relationships between various object-attributes
- conflict.kal - handles the data verification and conflict resolution operations on the stored design rationale information
- assertns.kal - the project-specific Kappa-PC file containing the design rationale
- descript.kal - the project-specific Kappa-PC file containing the list of ASCII text files having the descriptions for both design objects and rationale.

GARM (Section 2.6.2) specifies seven life cycle phases for AEC objects. The Life Cycle Phase Module provides support for this aspect of GARM. This dissertation only concentrates on the first two life cycle phases of GARM. The files in this module are:

- main.kal - contains definitions of interface objects used in both concept.kal and design.kal
- concept.kal - displays object information associated with the "as required" GARM stage
- design.kal - displays object information associated with the "as designed" GARM stage.

The Value Engineering Module provides support for various VE tasks. This module consists of the following files:

- ve.kal - handles the performance of VE tasks
- fast.kal - the project-specific Kappa-PC file containing the FAST Diagram
- queries.kal - the project-specific Kappa-PC file containing the various queries formulated on the design model.

The files in the Cost Breakdown Module presents the life cycle cost breakdown for the first two life cycle stages. These files are:

- breakdwn.kal - contains definitions of interface objects used in both brkdwnc.kal and brkdwnd.kal

- brkdwnc.kal - displays cost breakdown diagrams associated with the "as required" GARM stage
- brkdwnd.kal - displays cost breakdown diagrams associated with the "as designed" GARM stage.

The Login Module provides the framework for customizing the information presentation methods of DRIVE. For example, structural engineers have a different method of viewing design objects (e.g., walls) from architects. Currently, DRIVE does not support this information viewpoint capability. The files in this module are:

- login.kal - handles the storage and retrieval of user-specific preferences
- users.kal - contains a list of the current users of the DRIVE application
- designat.kal - contains descriptions of various user designations.

Finally, the Initialization / Support Module consists of several files performing simple initialization or support tasks.

3.6 EXAMPLE PROJECT DESCRIPTION

The purpose of this section is to present a summary description of an actual VE project used as an example in the succeeding chapters. Readers interested only in the theoretical DR data structure can proceed directly to the next chapter without adversely affecting their ability of following the discussion in the succeeding chapters.

The example building project used in the next three chapters is the Resident Engineer's Office / Visitors' Center of the McAlpine Locks Support Buildings Project. OVEST conducted the VE study for the McAlpine project (OVEST 1995). The Resident Engineer's Office / Visitors' Center, the Operations Service Building, the Operations Storage Building, and the Engineering Storage Building comprise the four support buildings related to the McAlpine Locks and Dam Replacement Project. The Louisville District of the U. S. Army Corps of Engineers need these support buildings for the various activities associated with the replacement of the locks at the McAlpine facility.

The following examples from the above-mentioned VE study illustrate the kinds of information constituting design rationale. Some of these information are text descriptions of the various design objects. Another type of DR information are text descriptions of the owner or code requirements regarding the various parameters of the design objects. Relationships between the parameters of the different design objects also constitute a form of DR information. The various limitations of the DRIVE system necessitated the use of minor modifications to the actual rationale information gathered from the VE study.

- Building Description

  The Resident Engineer's Office / Visitors' Center building serves as a dual purpose facility. It serves as the field office of the Louisville Districts' Construction Division during the four- to six-year construction phase of the

McAlpine Lock Replacement Project. It also serves as a Visitor Center both during and after the construction of the replacement locks. The walkways, ramps, outdoor seating, picnic tables, and landscaping provided in this facility allow the general public to view the new lock construction as well as existing lock activities.

- Building Location Owner Requirement

  The location of the Resident Engineer's Office / Visitors' Center is an existing rock knoll at the south end of the McAlpine Locks property on the Kentucky Bank. The elevation of this site provides an excellent vantage point to view the lock activities. This is essential for both the Construction Division monitoring the replacement lock construction activities as well as for the general public using the Visitors' Center facility.

  Another reason for the selection of this location is that it allows separate roads for construction vehicles and general public vehicles. A guard rail system supplemented with sufficient traffic signs helps direct the general public to the Visitors' Center separating them from construction activities

- Building Function Description: Provide Construction Management Office

  The estimated four- to six-year construction time for the replacement of the McAlpine Locks coupled with the 15 minute driving time from the District

office to the project site necessitates the need to provide a field office for the Construction Division personnel involved in the project.

The various alternatives considered for the Resident Engineer's Office were the use of inexpensive construction trailers, the rehabilitation of existing structures, and the construction of a new custom facility. Although constructing a new facility is not the least cost alternative, it, however, has the most flexible opportunity for re-use. This alternative not only provides a high quality work environment for Corps employees but also allows multiple use as a Visitors' Center and as a historic mitigation facility.

- Building Function Description: Mitigate Impacts to Historic Structures

  The Kentucky State Historic Preservation Officer requires the District to mitigate the impacts to historic structures. The existing locks, dating to the 1870s, are historic structures. The mitigation efforts include:

  - the preservation of two stone tablets from the 1870 locks,
  - the preservation of other stone remnants of the 1870 lock walls and abutments,
  - the documentation of the existing locks and swing bridge by photographs and drawings, and

- the documentation of the historical developments of navigation improvements on the Ohio River.

The building will house comprehensive displays and artifacts relating to the history of the McAlpine Locks and Dam.

Current statutes limit the cost of historic mitigation measures to one percent of the project cost. There is no justification to use such funds to provide more expensive construction materials for the Resident Engineer's Office. However, the availability of such a facility gives the District broader latitude and flexibility in its negotiations with the State Historic Preservation Officer.

- Building Function Description: Promote Louisville District Projects

  Corps regulations require some level of visitor information at their projects. There are minimal visitor facilities existing on Shippingport Island. The Operations Division supports moving these facilities off the island for safety and security reasons. The relocation of the Visitors' Center to the Kentucky Bank would avoid visitor inconveniences associated with the operation of the bascule bridge connecting Shippingport Island to the Kentucky Bank. The construction of a multi-function building for the Resident Engineer's Office and Visitors' Center allows the District to do away with the existing Visitors' Center on Shippingport Island.

Other Corps Districts do a much better job than the Louisville District in promoting their projects. In light of the significance of the McAlpine Locks to the development of navigation on the Ohio River and the proximity of the facility to Louisville and southern Indiana, there exists an excellent opportunity for the Louisville District to promote itself and the McAlpine Locks Replacement Project.

Considerable debate surrounded the viability of a future use Visitors' Center. The main argument against this future use is the fact that funds for such activities are diminishing. This resulted in a major emphasis on the convertibility of the building for a possible, but not yet determined, future use.

- Building Number of Occupants Owner Requirement

  The Construction Division estimates that it needs office space for 26 employees broken down as follows: 1 Resident Engineer, 1 Assistant Resident Engineer, 2 Office Management Engineers, 1 Quality Assurance Engineer, 2 Contract Administration Civil Engineers, 2 Claims Resolution Civil Engineers, 1 Geotechnical Engineer, 1 Electrical Engineer, 1 Mechanical Engineer, 1 Secretary, 2 Clerks, 4 Construction Representatives, and 7 Interns / Co-op Students / Summer Hires.

- Minimum Ground Floor Elevation Requirement

  The 100-year design flood level of 400 feet above sea level.

- Mechanical Room General Function - Mechanical Room Fire Resistance Rating Relationship

  Section 600.5 of the BOCA Building Basic Code specifies that rooms containing HVAC equipment shall be segregated by construction of not less than 2-hour fire resistance rating and with means of ingress and egress from the exterior. Also, all the equipment inside, the doors leading into the inside corridor, and the walls of the mechanical room have to have a 2-hour fire resistance rating.

- Mechanical Room Fire Resistance Rating - Partition Wall 2 Fire Resistance Rating Relationship

  The minimum value of the fire resistance rating of Partition Wall 2 must be at least equal to that of the Mechanical Room.

CHAPTER 4

DATA STRUCTURES 1:

KNOWLEDGE REPRESENTATION MODULE

The Knowledge Representation Module acts as a storage area for product knowledge. Product knowledge refers to information about the product itself (for example, dimensions, materials used). As stated in the research limitations (Section 1.3), this dissertation develops only one KRM, namely, Building Construction. This Building Construction KRM serves as a template for other researchers interested in developing other KRMs, such as for highway construction and marine construction.

A KRM has three distinct components, namely, performance objects, typological library objects, and project-specific objects. Figure 4.1 shows these three components for the Building Construction KRM and how they relate to one another. The **P**erformances **L**ibrary (PL) Object, **B**uilding **C**onstruction **L**ibrary (BCL) Object, and **B**uilding **P**roject **H**ierarchy (BPH) Object represents, respectively, the performance objects, typological library objects, and project-specific objects.

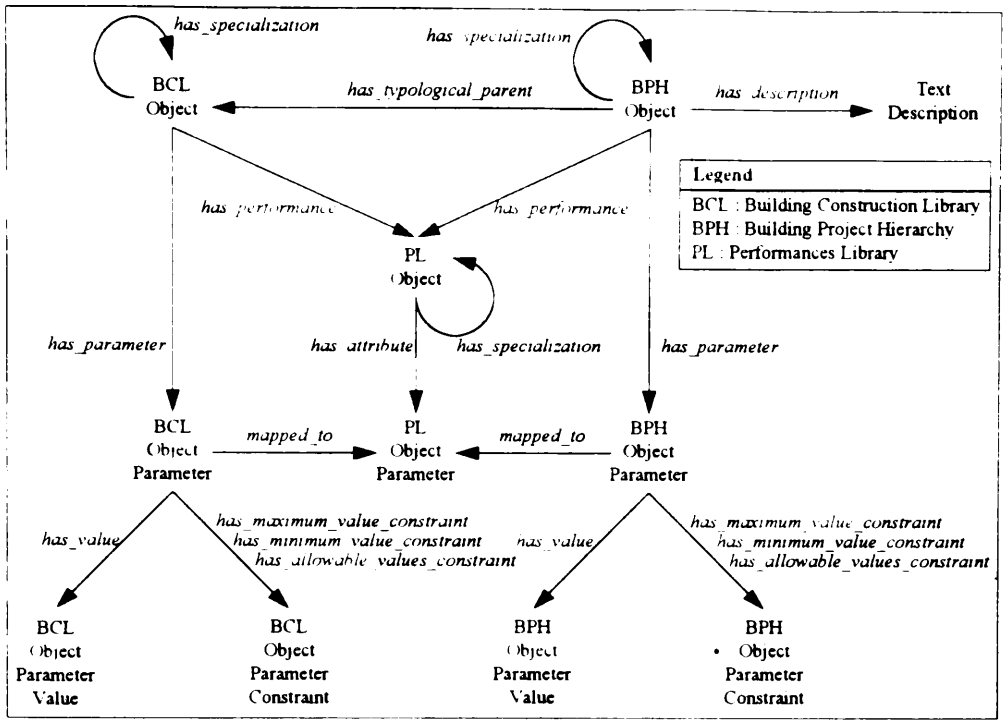


Figure 4 1 KRM Semantic Net Representation

The three major sections of this chapter detail each of the KRM components. This chapter uses a four-part breakdown structure. The four parts are theory, practical example, DRIVE interface, and DRIVE implementation. This breakdown structure allows readers interested only on a particular aspect of this dissertation to skip entire sections without affecting their ability of following the discussions. Readers concerned only on the theoretical data structures developed in this dissertation can read only the theory sections. Readers who want to know only about how the data structures represent real world information can concentrate on the practical example sections. The DRIVE interface sections target readers interested in using the DRIVE system. Finally, readers researching the technical implementation issues of DRIVE can look into the implementation sections.

4.1 PERFORMANCES LIBRARY

The Performance Library contains objects grouping together a set of related parameters. For example, the performance object `Economic Performance` groups together all cost-related parameters, such as, `Construction Cost` and `Maintenance Cost`. The white background portion of Figure 4.2 denotes the specific portions of the KRM discussed in this section.

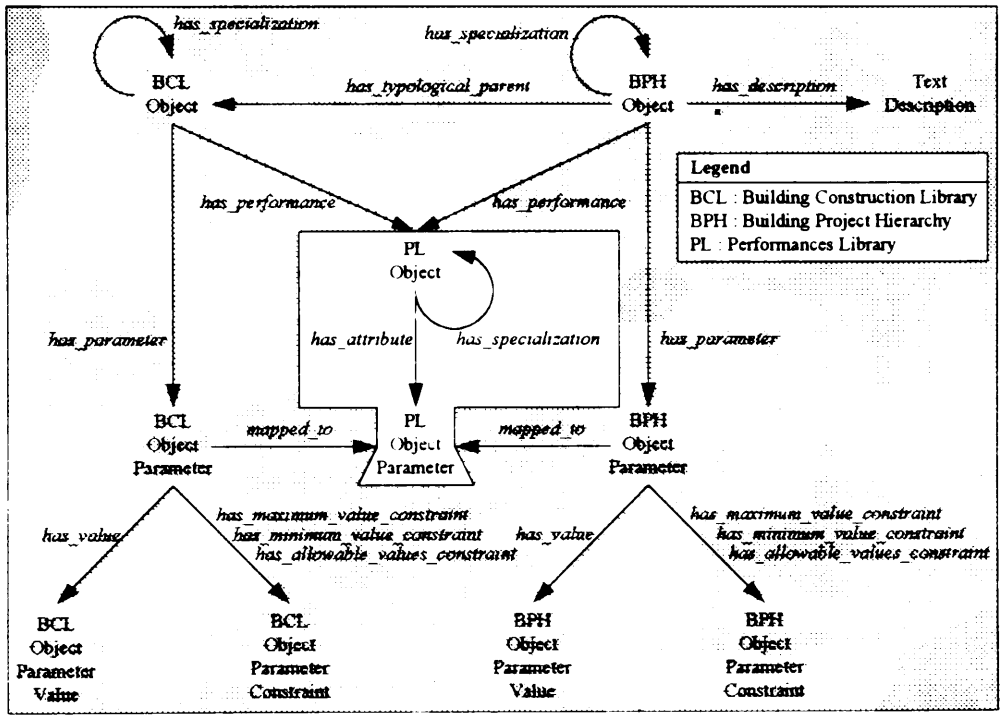


Figure 4.2 Performances Library

4.1.1 Theory

Each different KRM domain has its own particular set of performance objects. Figure 4.3 shows some of the PL Objects for the Building Construction KRM. Another KRM

domain, for example, Highway Construction, contains a different set of PL Objects. GARM (Section 2.6.2) serves as the basis for the structure of the Building Construction KRM, especially in the `Building Performances` and `Space Performances` sub-hierarchies. The *has_specialization* circular link in Figure 4.2 signifies the hierarchical nature of PL Objects. For example, the `Building Performances` PL Object in Figure 4.3 has a specialization called `Acoustical Performances`

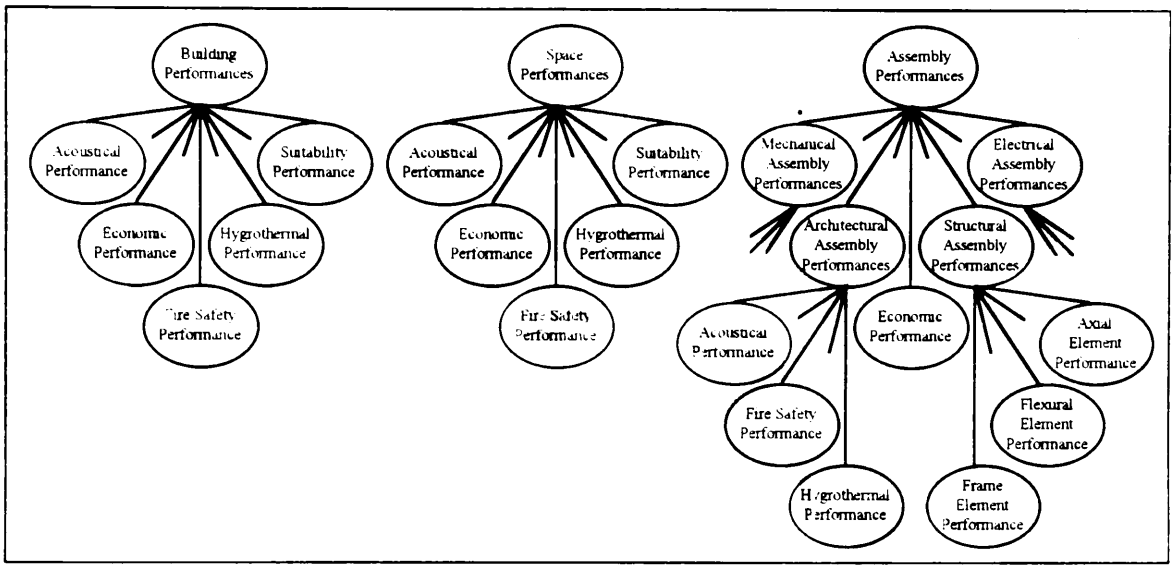


Figure 4.3 Building Construction PL Objects

The internal structure of a PL Object remains the same whether it is part of the Building Construction KRM, Highway Construction KRM, or any other KRM domain. The consistent internal structure of PL Objects supports modularity. A PL Object internally consists of several related PL Object Parameters. Figure 4.4 shows the entire internal structure for a PL Object and exemplifies the *has_parameter* link in Figure 4.2.

One advantage of using a PL is the capability of grouping together related parameters. Without a PL, a design object has no mechanism for selectively presenting information about related parameters. The hierarchical nature of the PL allows further grouping of parameter groups. In Figure 4.3, **Acoustical Performances**, **Economic Performances**, and **Fire Safety Performances** each contains groups of related parameters. The **Building Performances** object does not have any parameters in itself. It acts as a grouping mechanism for all the parameters of its lower level objects.

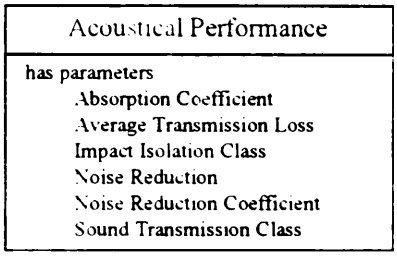


Figure 4.4 Internal Structure of a PL Object

Another advantage of using a PL is that it acts as a central repository of parameters. To illustrate, a designer wants to set the resonance frequency value of an assembly. The designer looks for the parameter under the **Acoustical Performance** object (Figure 4.4). The designer notes that it is missing and adds a new parameter **Resonance Frequency** to the **Acoustical Performance** object. All BCL (Section 4.2) and BPH (Section 4.3) objects having a *has_performance* link (dimmed in Figure 4.2) to this modified PL object also have the new parameter **Resonance Frequency**. Without a PL, designers must explicitly add a new parameter to all relevant BCL and BPH objects.

4.1.2 Practical Example

The `Acoustical Performance` object (Figure 4.4) has parameters `Absorption Coefficient`, `Average Transmission Loss`, `Impact Isolation Class`, `Noise Reduction`, `Noise Reduction Coefficient`, and `Sound Transmission Class`. This is consistent with how building engineers define acoustic parameters. If building engineers need to describe something not included in the current parameters, they simply create a new parameter. For example, resonance frequency is not in the current parameters of the `Acoustical Performance` object, so they create a new parameter `Resonance Frequency`.

4.1.3 DRIVE Interface

DRIVE contains a Domain Knowledge Editor allowing users to modify and enhance the various hierarchies representing a specific AEC domain. One of these hierarchies is the PL. Figure 4.5 shows the DRIVE interface for defining new parameters or modifying existing parameters. Currently, DRIVE only allows users to modify two parameter properties or facets (Section 2.2.1). These properties are cardinality and value type.

Clicking on the `New Parameter` button shown in Figure 4.5 allows users to create a new parameter. Typing in a parameter name, selecting the appropriate properties, and clicking on the `Create Parameter` button completes this procedure. To modify an existing parameter, users need to select a parameter to modify from the parameter list at the upper portion of Figure 4.5. This action sets both the text in the parameter name text box and the values in the parameter property boxes to correspond to those of the selected

parameter. It also changes the words in the `Create Parameter` button to `Update Parameter`. Clicking on the `Update Parameter` button completes this procedure.

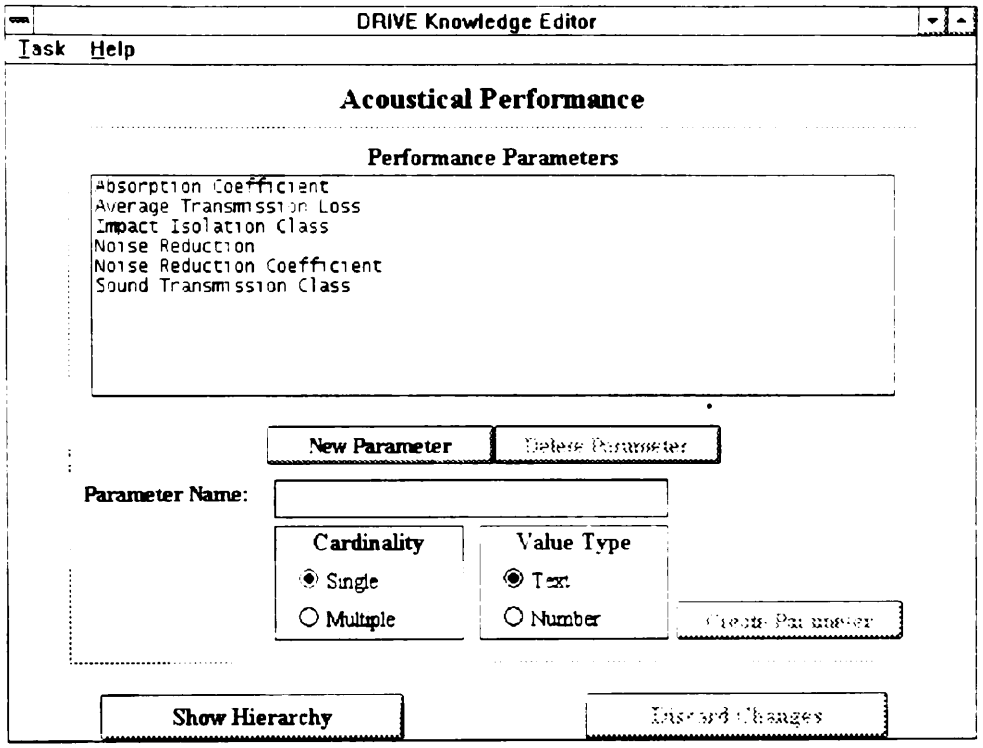


Figure 4.5 DRIVE Interface for Modifying PL Objects

4.1.4 DRIVE Implementation

DRIVE stores the PL in the `Performances` hierarchy. The module file `prfrmncs.kal` contains this hierarchy. The module file `knowledg.kal` contains the interface objects allowing users to manipulate the PL objects. Figure 4.6 shows the DRIVE implementation of the PL object shown in Figure 4.4.

DRIVE has an internal naming system for objects. `PerfType41` is the internal name for the `Acoustical Performance` PL Object. Kappa-PC limits DRIVE object names to 32 characters, with no two objects having the same name. These limitations severely restrict the name representation capability of the system. To address these limitations, DRIVE uses an internal-external name mapping procedure. The DRIVE system generates internal system names like `PerfType41`. DRIVE keeps track of the number of objects it creates and makes sure that there is no duplication of internal names. All PL Objects have `PerfType` as the prefix to their object numbers. Users are responsible for assigning appropriate external names to objects. DRIVE stores the external name of an object as an object slot. All DRIVE objects have a `Title` slot storing the external name. For example, in Figure 4.6, the external name of the object `PerfType41` is `Acoustical Performance`. DRIVE stores external names as character strings. These strings have a maximum length of 199 characters. This gives the system a greater expressive representation capability regarding object names.

PerfType41

Slot	Value	Cardinality	Value Type
Title	Acoustical Performances	Single	Text
NewAttrNumber	7	Single	Number
PerfType41Attr1		Single	Number
PerfType41Attr1Name	Sound Transmission Class	Single	Text
PerfType41Attr2		Single	Number
PerfType41Attr2Name	Absorption Coefficient	Single	Text
...	...	...	...

Figure 4.6 DRIVE Implementation of a PL Object

Another advantage of using internal-external name mapping is that it allows duplicate external object names. Figure 4.3 shows the Building Construction PL. A close inspection shows that `Acoustical Performance` appears three times — once each under `Building Performances`, `Space Performances`, and `Assembly Performances`. These three `Acoustical Performance` objects have different internal system names. Having the same external names under different branches of the PL accurately models the performance concept in design. Typically, performance parameters vary from the level of abstraction standpoint. For example, acoustical parameters for the spaces level include general noise frequency and intermittent noise frequency, while the assemblies level includes noise reduction coefficient and sound transmission class. The internal-external name mapping allows the DRIVE system to have external names repeated for different internal-named objects having entirely different sets of parameters.

DRIVE also applies the internal-external name mapping concept to parameters. Figure 4.6 shows how DRIVE implements this concept. DRIVE generates two slots for each new parameter. An example of these slots is `PerfType41Attr1` and `PerfType41Attr1-Name`. The first slot stores the cardinality and value type facets of the parameter. The second slot stores the external name of the parameter. For example, the external name `Sound Transmission Class` corresponds to the internal name `PerfType41Attr1`. Finally, the DRIVE system handles the internal-external name transformation such that users always work with external names, without the need to know the internal names.

4.2 BUILDING CONSTRUCTION LIBRARIES

The Building Construction Libraries contain information generally applicable to a type of building, space, assembly, or construction trade. These four BCLs are examples of typological libraries, another component of a KRM. Typological libraries contain a set of definitions for the various objects usually encountered within a particular domain. The white background portion of Figure 4.7 is the focus of the discussion in this section.

4.2.1 Theory

4.2.1.1 BCL Object Hierarchies

The typological libraries component of the KRM provides a mechanism for organizing domain-specific knowledge. For the building construction domain, the KRM uses four BCLs to organize the domain knowledge. Figures 4.8 through 4.11 show some of the objects in the four libraries. The *has_specialization* circular link in Figure 4.7 denotes the hierarchical nature of BCL Objects. All BCL Objects are classes. There are no instances in the BCL hierarchies. This is consistent with the purpose of typological libraries. These libraries act as templates or starting points for the definition of project-specific objects.

The Buildings Library (Figure 4.8) and the Spaces Library (Figure 4.9) are very similar to the Building Typology Library and Space Typology Library, respectively, of the SOS model (de la Garza and Oralkan 1992). Function is the basis behind the breakdown structure behind these libraries. For example, the primary function of `Corridor Spaces` is circulation. As such, they are a specialization of `Circulation Spaces`. `Closet Spaces`

have a very different function from **Circulation Spaces**. **Closet Spaces** are a specialization of **Storage Spaces**, whose primary function is storage.

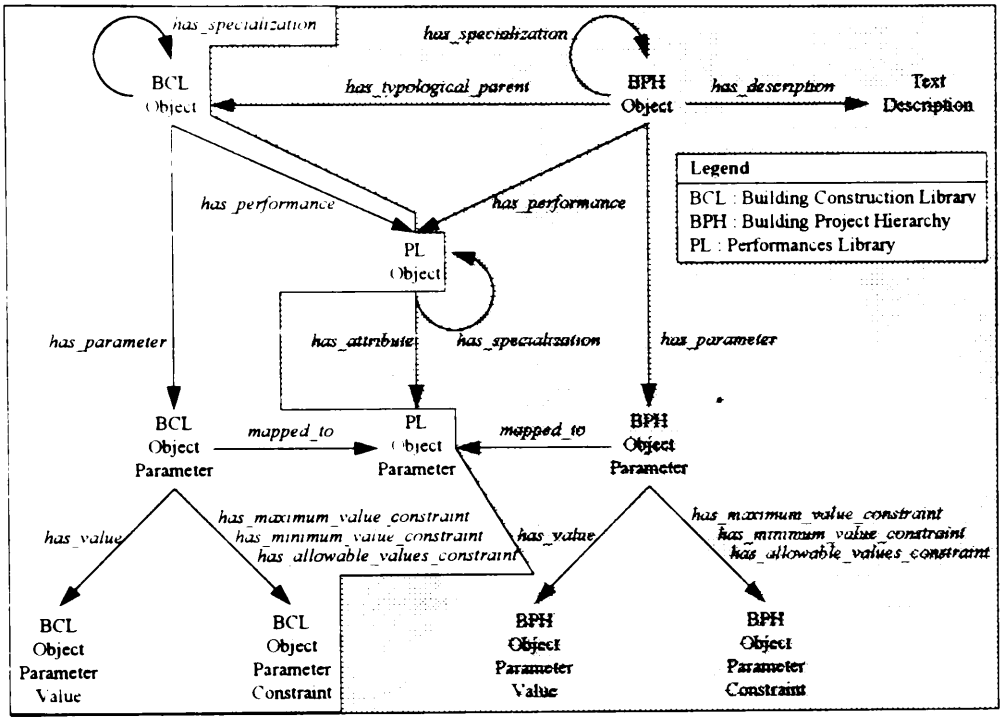


Figure 4.7 Building Construction Libraries

The Assemblies Library (Figure 4.10) uses a breakdown structure based on building systems. The Uniformat breakdown structure is a suitable basis for the Assemblies Library. The MCACES breakdown structure is another suitable system. This dissertation uses the MCACES breakdown structure to acknowledge the cooperation of OVEST in this research effort. The U. S. Army Corps of Engineers, the parent organization of OVEST, uses the MCACES system in their cost estimating work.

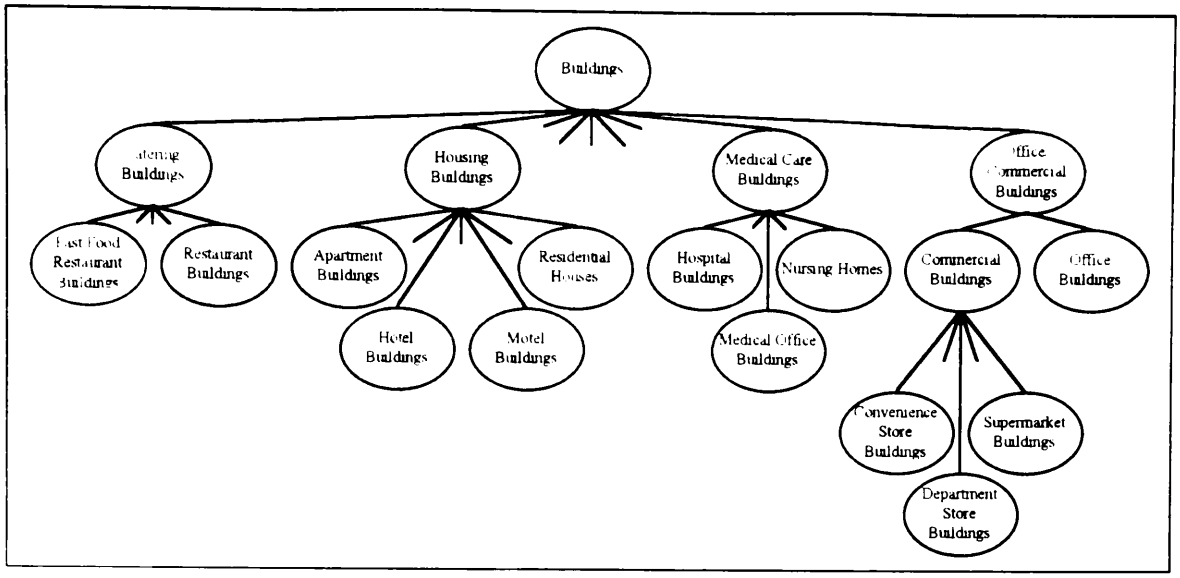


Figure 4.8 Buildings Library

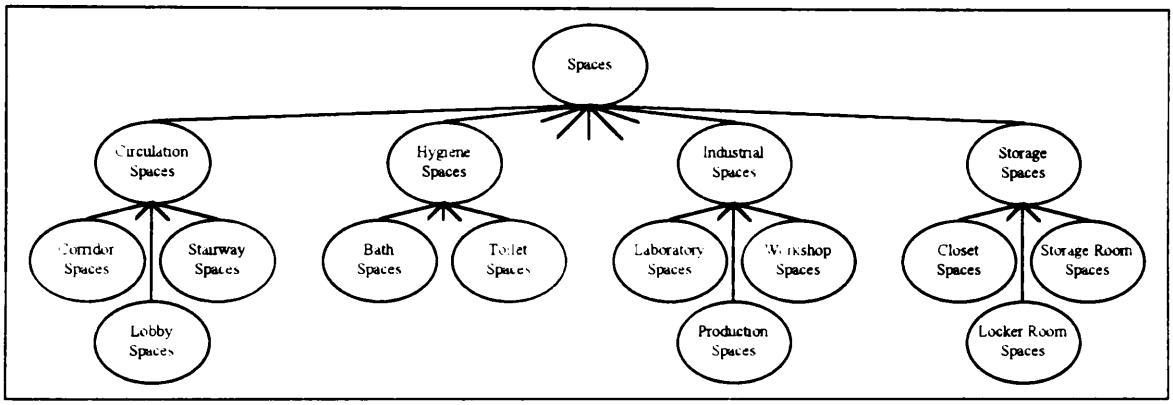


Figure 4.9 Spaces Library

The Assemblies Library is conceptually similar to the Building Element Typology Library of the SOS model (de la Garza and Oralkan 1992). The SOS model (Section 2.3.5.8) classifies assemblies into only two types — structural and non-structural. The Assemblies Library defined in this dissertation uses the entire MCACES breakdown structure. This makes it a super-set of the Building Element Typology Library of the SOS model.

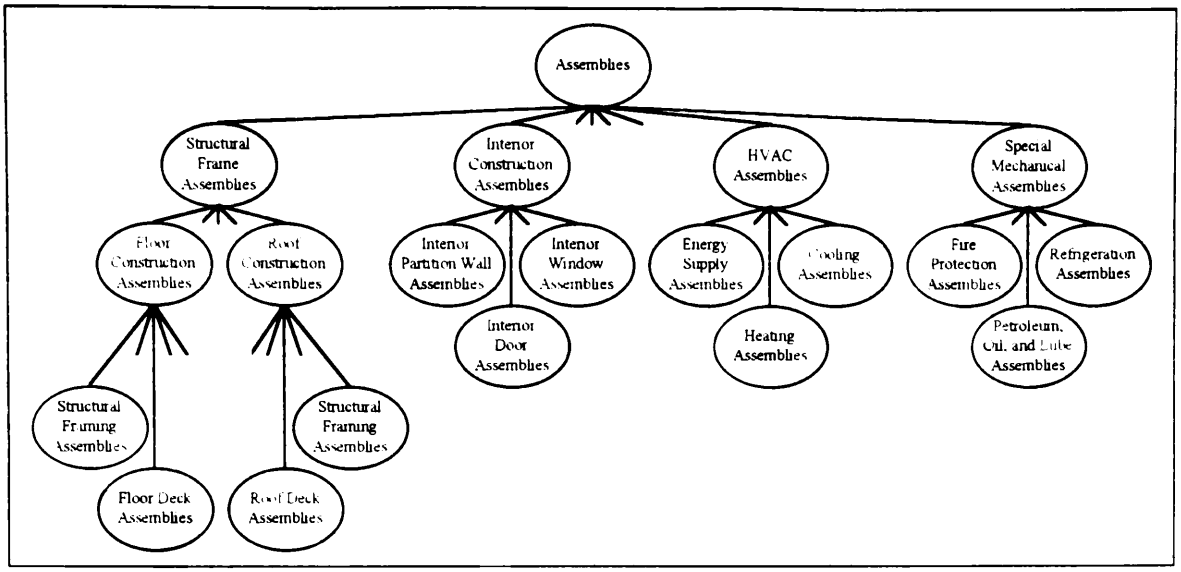


Figure 4.10 Assemblies Library

The Construction Trades Library follows the Masterformat breakdown structure. The Masterformat system groups all costs associated with a construction trade, such as concrete construction, masonry construction, and metal construction, among others. Masterformat differs from system-based breakdown structures like Uniformat and MCACES. Table 4.1 shows the major divisions of these three systems. Costs in certain trade-based divisions spread across various systems. For example, Concrete (Masterformat Division 3) pools costs from Substructure (MCACES Division 1), Structural Frame (MCACES Division 2), and Exterior Closure (MCACES Division 4), among others. Conversely, costs in several trade-based divisions combine to determine the cost of a single system-based division. For example, portions of Concrete (Masterformat Division 3), Metal (Masterformat Division 5), and Doors, Windows, and Glass (Masterformat Division 7) aggregate together to define Exterior Closure (MCACES

Division 4) costs. Both trade-based and system-based breakdown structures are necessary in a rationale capture system. Users may sometimes capture rationale for systems (for example, all first floor columns) or for trades (for example, all cast-in-place concrete).

Table 4.1 Building Component Breakdown Structures

MCACES	Uniformat	Masterformat
00 General Conditions	01 Foundations	01 General Requirements
01 Substructure	02 Substructure	02 Site Work
02 Structural Frame	03 Superstructure	03 Concrete
03 Roofing	04 Exterior Closure	04 Masonry
04 Exterior Closure	05 Roofing	05 Metals
05 Interior Construction	06 Interior Construction	06 Wood and Plastics
06 Interior Finishes	07 Conveying	07 Moisture Thermal Control
07 Specialties	08 Mechanical	08 Doors, Windows, and Glass
08 Plumbing	09 Electrical	09 Finishes
09 HVAC	10 General Conditions	10 Specialties
10 Special Mechanical	11 Specialties	11 Equipment
11 Interior Electrical	12 Site Work	12 Furnishings
12 Special Interior Electrical		13 Special Construction
13 Equipment and Conveying		14 Conveying Systems
14 Site Preparation		15 Mechanical
15 Site Improvements		16 Electrical
16 Site Utilities		

The Construction Trades Library (Figure 4.11) synthesizes the Building Component Typology Library and the Material and Equipment Library of the SOS model (de la Garza and Oralkan 1992). In the SOS model (Section 2.3.5.8), the Building Component Typology Library classifies components into only two types — structural and non-structural. The Material and Equipment Library also uses the Masterformat breakdown structure. It also contains specific product brands as instances. The Construction Trades Library combines these two SOS libraries into one, since they perform essentially the same

function. The Construction Trades Library also converts the specific brand instances in the SOS model into classes. This is a more accurate representation of building domain objects. For example, a particular brand of door, `Metal Door Brand XYZ`, can have several instances, `Exterior Door 1` and `Exterior Door 2`

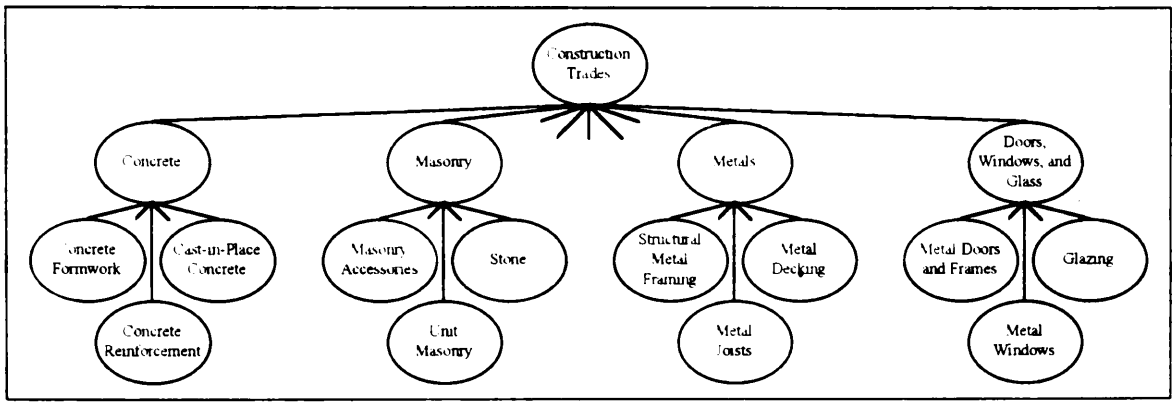


Figure 4 11 Construction Trades Library

4.2.1.2 BCL-PL Relationship

All BCL Objects have performances. These performances are PL Objects. The *has_performance* link in Figure 4.7 denotes this relationship. The PL acts as a central repository of performance parameters. BCL Objects do not have any performance parameters on their own. They acquire parameters through links with PL Objects. For example, the `Interior Partition Wall Assemblies` BCL Object (Figure 4.10) initially has no performances nor parameters. Assigning the `Acoustical Performance` PL Object (Figure 4 4) as a performance of `Interior Partition Wall Assemblies` gives the latter the parameters of the former. A PL Object only gives parameters to a BCL Object. It does not assign values to these parameters. Figure 4 7 names these parameters as BCL

Object Parameters. It also denotes the relationship between a BCL Object and a BCL Object Parameter as a *has_parameter* link.

A BCL Object Parameter also has a *mapped_to* relationship link with a PL Object Parameter. Assigning performances to a BCL Object gives parameters to the BCL Object. However, this is not sufficient. A BCL Object Parameter must also have the same properties as its PL Object Parameter counterpart. Examples of these properties are cardinality and value type. The *mapped_to* relationship link duplicates the properties stored in a PL Object Parameter to its BCL Object Parameter counterpart.

4.2.1.3 BCL Values and Constraints

The *has_performance*, *has_parameter*, and *mapped_to* links define performance parameters for a typological library object. The *has_value* and the three *has_constraint* links define the values and constraints for these performance parameters. The value type property of a parameter determines which *has_constraint* link is valid for the parameter. A numerical value type property enables the *has_maximum_value_constraint* and *has_minimum_value_constraint* links between a BCL Object Parameter and a BCL Object Parameter Constraint. A textual value type property enables the *has_allowable_values_constraint* link.

These values and constraints represent object-specific information generally applicable to the particular object type. For example, the Space Types parameter of the object Office

Buildings has the values Office Spaces, Lobby Spaces, Corridor Spaces, Toilet Spaces, Mechanical Equipment Spaces, and Electrical Equipment Spaces. This value represents information typical to Office Buildings. One of the purposes of the BCL is to provide designers with starting points for the design. The BCL Object Parameter Values and the BCL Object Parameter Constraints help achieve this purpose.

4.2.2 Practical Example

The 100mm Hollow Block Masonry Walls have an Acoustical Performance (Figure 4.4). One of the parameters of this performance is Sound Transmission Class (STC). The value for the STC is 38 for all 100mm Hollow Block Masonry Walls. All specific occurrences or instances of this wall type inherit its defined performance parameter values, such as the STC value of 38. The 100mm Hollow Block Masonry Walls is an example of a typological library object. One of the purposes of typological library objects is to store information used in initializing the various parameters of their specific instances.

4.2.3 DRIVE Interface

The Domain Knowledge Editor of DRIVE allows users to modify and enhance the typological libraries comprising an AEC domain. For the building construction domain, the typological libraries are: Buildings Hierarchy, Spaces Hierarchy, Assemblies Hierarchy, and Construction Trades Hierarchy. The procedure for modifying hierarchy information is the same for all hierarchies. Figure 4.12 shows the DRIVE interface for specifying parameter values for typological library objects. All BCL objects initially have no

performances and consequently, no parameters. Clicking the **Add Performance** button displays the DRIVE interface for assigning performances (Figure 4.13). The two arrow buttons add or remove performances from the typological library object. Attaching performances creates parameters in the typological library object. Users can then define default values for these parameters on the DRIVE window shown in Figure 4.12. Highlighting a performance parameter in the list box, typing in a value in the **Default Value** text box, and clicking the **Set Default Value** button completes this action.

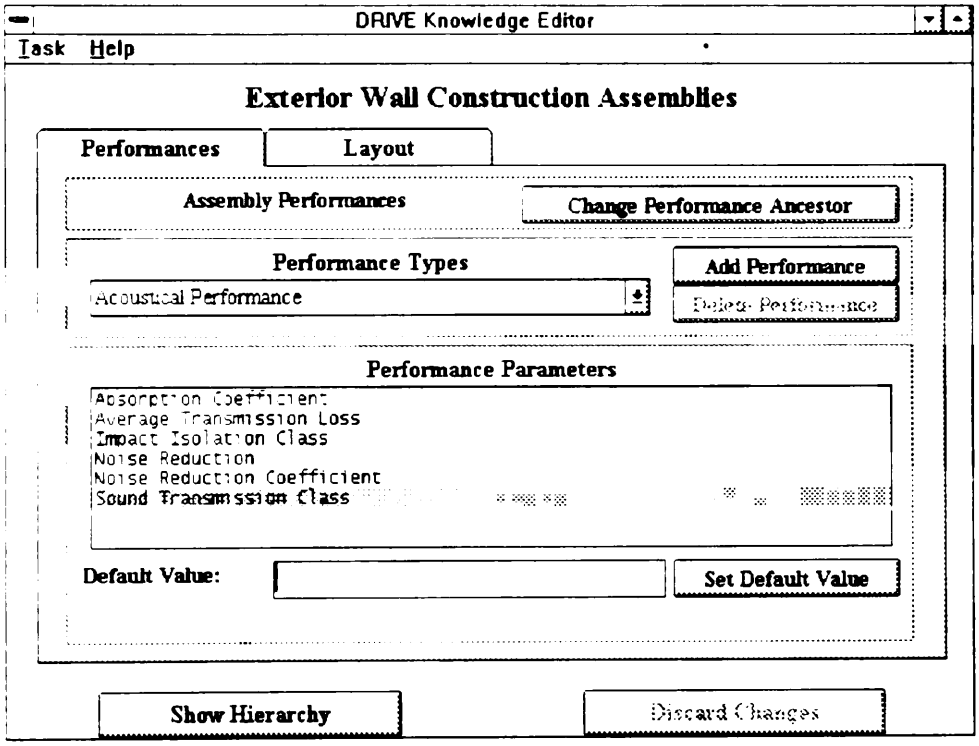


Figure 4.12 DRIVE Interface for Defining BCL Values

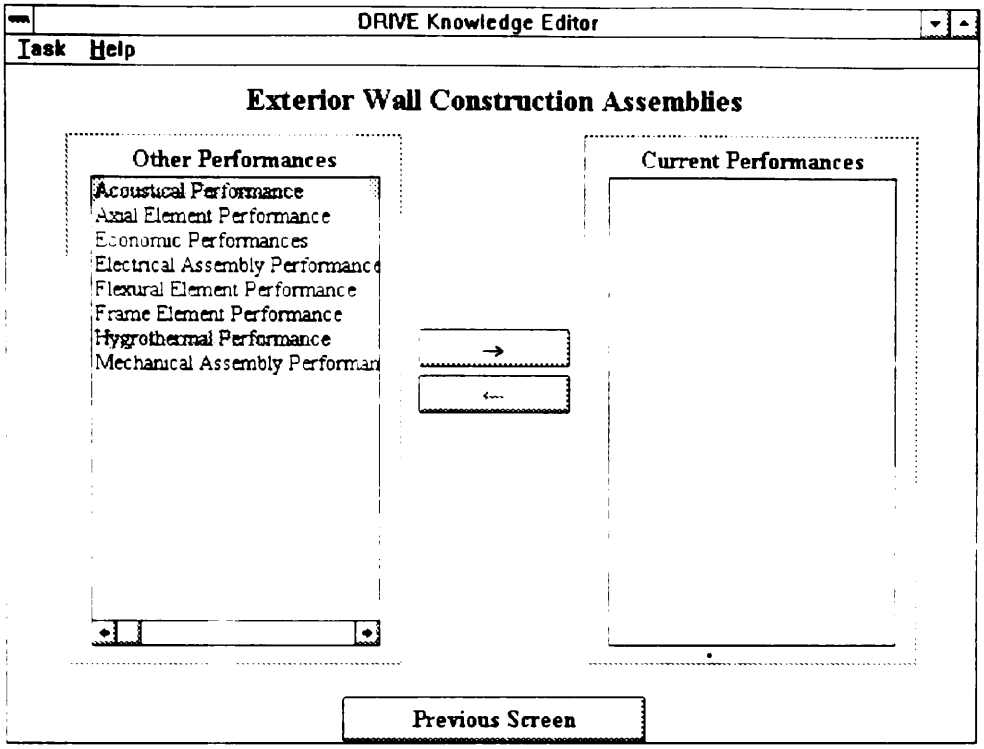


Figure 4 13 DRIVE Interface for Attaching Performances

4.2.4 DRIVE Implementation

DRIVE stores the four BCLs (Buildings Hierarchy, Spaces Hierarchy, Assemblies Hierarchy, and Construction Trades Hierarchy) in four separate module files (`bldgs.kal`, `spaces.kal`, `assembly.kal`, and `trades.kal`). The module file `knowledg.kal` contains the interface objects allowing users to modify the BCL objects. DRIVE also uses the same internal-external name mapping system used by the PL (Section 4.1.4) for BCL objects. Figure 4.14 shows the internal representation of a BCL Object and its relationship with PL Objects. The slot `Title` stores the external name `100mm Hollow Block Masonry Walls` for the BCL Object `AssyType304`. Similar to the PL, DRIVE users only work with external name representations for the BCL.

The `RelatedPerformances` slot in BCL objects stores a list of related performances. DRIVE stores performances as PL objects. The BCL objects duplicates the parameter definitions stored in the PL objects. In Figure 4.14, the BCL Object `AssyType304` has performances `PerfType41 (Acoustical Performance)` and `PerfType45 (Hygrothermal Performance)`. Figure 4.6 shows the parameter definitions for the `Acoustical Performance`. After attaching `Acoustical Performance` and `Hygrothermal Performance` to itself, the `100mm Hollow Block Masonry Walls (AssyType304)` now has the parameters of both performances. Some of these parameters are `Sound Transmission Class (PerfType41Attr1)`, `Absorption Coefficient (PerfType41Attr2)`, and `Thermal Resistance R (PerfType45Attr4)`. This illustrates the reasoning behind the internal naming scheme for the parameters of PL Objects. PL Object Parameters (for example, `Sound Transmission Class`) have internal names (for example, `PerfType41Attr1`) beginning with the name of the PL Object (for example, `PerfType41`) followed by the attribute number (for example, `Attr1`). Attaching the PL Object name as a prefix ensures unique internal names for BCL Object Parameters.

BCL Objects can have specific values, constraints, or both. The various parameters or slots act as storage spaces for these values and constraints. Initially, these values and constraints are unknown. Users can set these values and constraints to reflect object-specific information. In Figure 4.14, an example of object-specific information is the value of `38` for the parameter `Sound Transmission Class (PerfType41Attr1)`.

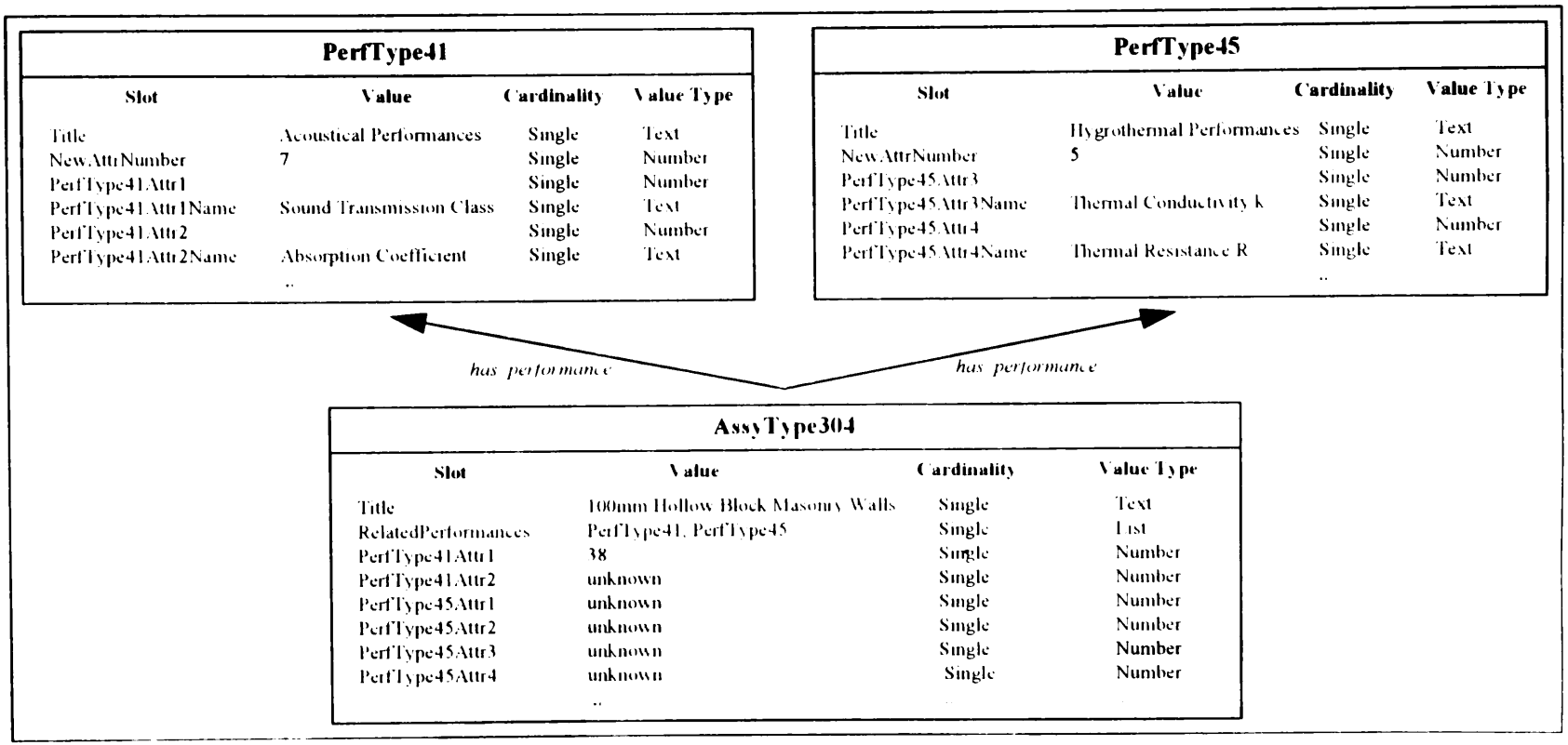


Figure 4.14 DRIVE Implementation of a BCL Object

4.3 BUILDING PROJECT HIERARCHIES

The Building Project Hierarchies contain information specifically tailored for a single project. For the building construction domain, four BPHs comprise the project-specific portion of the KRM. These BPHs are Building, Spaces Hierarchy, Assemblies Hierarchy, and Construction Trades Hierarchy. The white background portion of Figure 4.15 denotes the portion of the KRM discussed in this section.

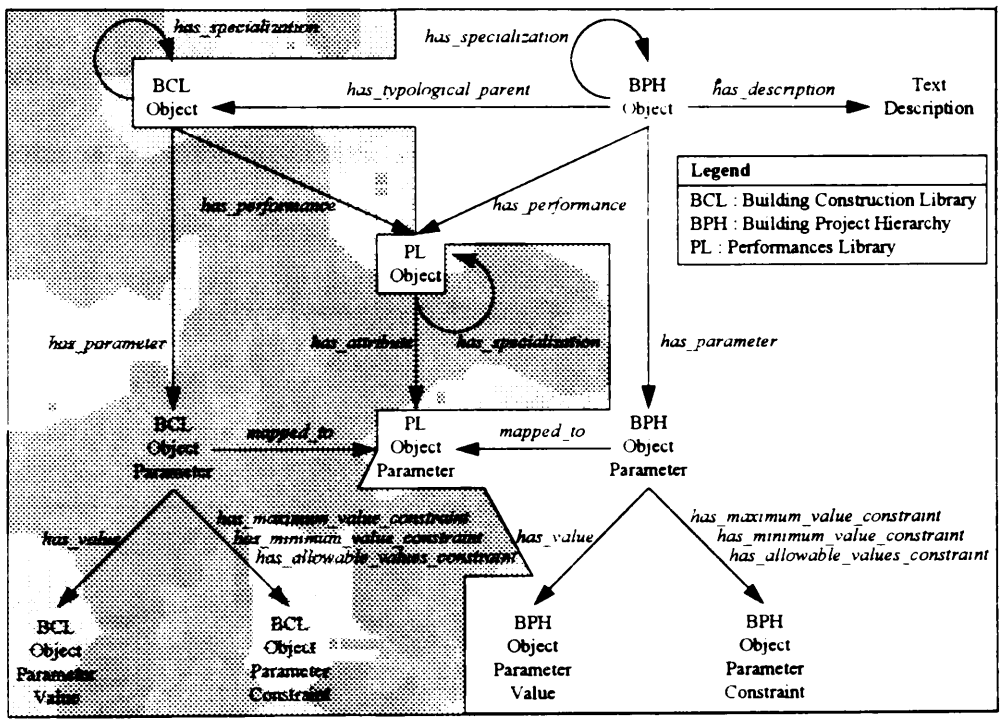


Figure 4.15 Building Project Hierarchies

4.3.1 Theory

4.3.1.1 BPH Objects

The BPHs organize project-specific product knowledge. Figure 4.16 shows some of the BPH objects for the Resident Engineer's Office / Visitors' Center of the McAlpine Locks Support Buildings Project (Section 3.4). Of the four top-level BPH Objects, only Building does not specialize any further. The *has_specialization* circular links in Figure 4.15 show the hierarchical nature of the other three top-level BPH objects, namely Spaces Hierarchy, Assemblies Hierarchy, and Construction Trades Hierarchy. These *has_specialization* links are valid unless the BPH Object has a specific graphical model representation. BPH Objects having a graphical model representation are the terminal classes in the hierarchy. These objects can not specialize any further. `Conference Room`, `Open Office Space`, and `Mechanical Room Partition Wall 2` are examples of these objects in Figure 4.16.

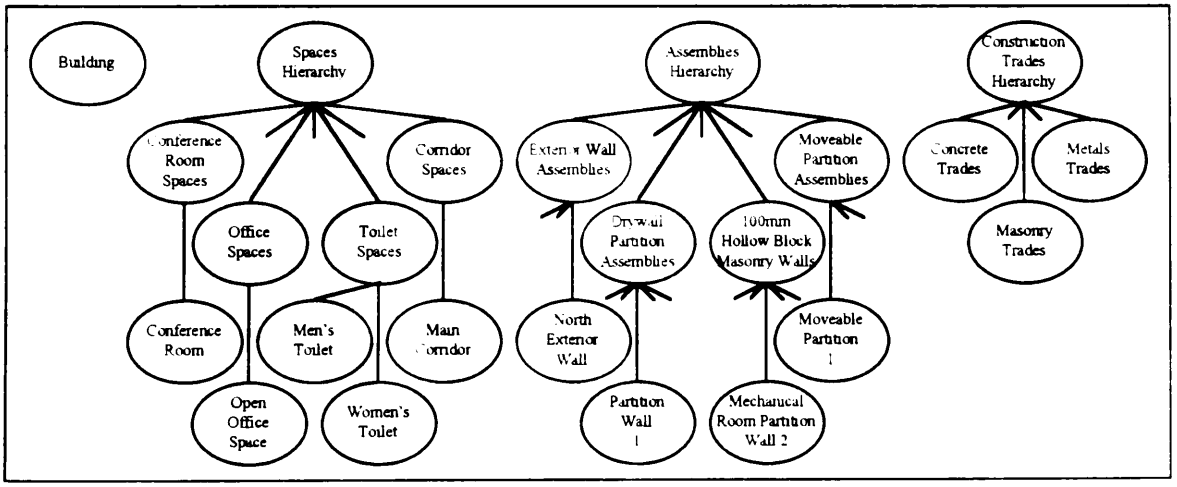


Figure 4.16 Building Construction BPH Objects

Each BPH Object has several instances corresponding to its life cycle phases. Using the GARM life cycle breakdown (Section 2.6.2), all BPH Objects have seven life cycle instances — "as required", "as designed", "as planned", "as built", "as used", "as altered", and "as demolished". Figure 4.15 does not show these instances. In all figures having references to BPH objects, a graphical representation of a BPH class object automatically implies the existence of the life cycle instances. A BPH Object continually undergoes changes as it goes through it various life cycle phases. The use of separate instances for each life cycle phase provides a method of tracking how the parameters of a BPH Object change over time. Although there are seven life cycle phases, this dissertation only concentrates on the first two phases.

4.3.1.2 BPH-PL Relationship

BPH Objects behave similarly to BCL Objects regarding their relationship with the PL. BPH Objects acquire parameters from PL Objects through the *has_performance*, *has_parameter*, and *mapped_to* links shown in Figure 4.15. The discussion in Section 4.2.1.2 is applicable not only for BCL Objects but also for BPH Objects.

4.3.1.3 BPH Values and Constraints

The *has_value* and the three *has_constraint* links define the values and constraints for the performance parameters for a BPH Object. The value type property of a parameter determines which *has_constraint* link is valid for the parameter. A BPH Object Parameter has only one value (denoted by BPH Object Parameter Value in Figure 4.15) for each life

cycle phase. To accurately model an object parameter across its life cycle phases, a BPH Object Parameter needs seven BPH Object Parameter Values — one for each stage. These seven values may differ from one another to reflect the changes occurring on an object parameter over time. The *has_constraint* links are different from the *has_value* link regarding life cycle phases. A BPH Object Parameter has only one constraint value (denoted by BPH Object Parameter Constraint in Figure 4.15) for its *has_constraint* links for all life cycle phases. BPH Object Parameter Values may change over time but BPH Object Parameter Constraints remain constant over time.

4.3.1.4 BPH-BCL Relationship

The *has_typological_parent* link between a BPH Object and a BCL Object allows the former to "inherit" performances, parameters, values, and constraints from the latter. One of the purposes of the BCL is to provide starting points for the design of a BPH Objects. The indirect inheritance feature of the *has_typological_parent* link accomplishes this purpose. The *has_typological_parent* link is not a persistent inheritance link. During its initial creation, a BPH Object copies the performances, parameters, values, and constraints of its typological parent BCL Object. Copying allows a BPH Object to indirectly inherit information from a BCL Object without having a persistent inheritance link. A persistent inheritance link means that changing a higher level object (i.e., BCL Object) parameter changes also the lower level object (i.e., BPH Object) parameter.

4.3.1.5 BPH Object Text Descriptions

All BPH Objects have text descriptions. Figure 4 15 denotes this through the *has_description* link between a BPH Object and a Text Description. Text descriptions consist of paragraphs written in plain English to describe a particular BPH Object. Since it uses natural language text, human users are the intended consumers of these information. A computer system can process these information through natural language processing techniques. Natural language processing is a branch of artificial intelligence seeking to develop techniques for allowing computers to understand natural language text. However, natural language processing is beyond the scope of this dissertation. As such, this dissertation only uses the computer to store text descriptions. The computer does not perform any operations on these descriptions. These descriptions assist the human user in increasing their understanding of BPH Objects.

4.3.2 Practical Example

`Mechanical Room Partition Wall 2` (BPH Object) is a specific occurrence of `100mm Hollow Block Masonry Walls` (BCL Object). The `100mm Hollow Block Masonry Walls` have an `Acoustical Performance` (PL Object). One of the parameters of `Acoustical Performance` is `Sound Transmission Class` (PL Object Parameter). Since the `100mm Hollow Block Masonry Walls` have an `Acoustical Performance`, they also have the parameter `STC` (BCL Object Parameter). The value of the `STC` parameter for the `100mm Hollow Block Masonry Walls` is `38` (BCL Object Parameter Value). The `100mm`

Hollow Block Masonry Walls object is the typological parent of Mechanical Room Partition Wall 2. As such, Mechanical Room Partition Wall 2 gets the performances, parameters, values and constraints of 100mm Hollow Block Masonry Walls. Thus Mechanical Room Partition Wall 2 also has Acoustical Performance and the parameter STC with an "as designed" value of 38

BPH Objects have life cycle phases. The "as designed" value of 38 for the parameter STC is a specific value for a particular life cycle stage, namely, the "as designed" stage. Other phases may have different values. For example, the "as required", "as planned", and "as built" values for the parameter STC may be 32, 38, and 36, respectively. The use of life cycle stages catalogs the evolution of an object. For the above example, the designers required that the STC rating is 32, but they used a design with a rating of 38. The construction planners followed the design and therefore planned it with a rating of 38. During construction, unfavorable conditions resulted in a rating of only 36. Although this is less than the designed and planned values, it is still above the required value.

All BPH Objects have text descriptions. These text descriptions are ordinary English paragraphs describing a particular BPH Object. To illustrate, page 78 of this dissertation shows the description for the building used in the example project (Section 3.4).

4.3.3 DRIVE Interface

DRIVE has four BPHs, Building, Spaces Hierarchy, Assemblies Hierarchy, and Construction Trades Hierarchy. The procedure for creating new BPH Objects is the same for all hierarchies. Figure 4.17 shows the DRIVE interface for creating new BPH Objects. The objects shown in Figure 4.17 are BCL Objects. Clicking on one of these objects displays a pop-up menu. One of the menu choices is `Add Assembly Type`. Selecting this menu item creates a new BPH Object based on the selected BCL Object. BCL Objects are starting points for the design of project-specific BPH Objects.

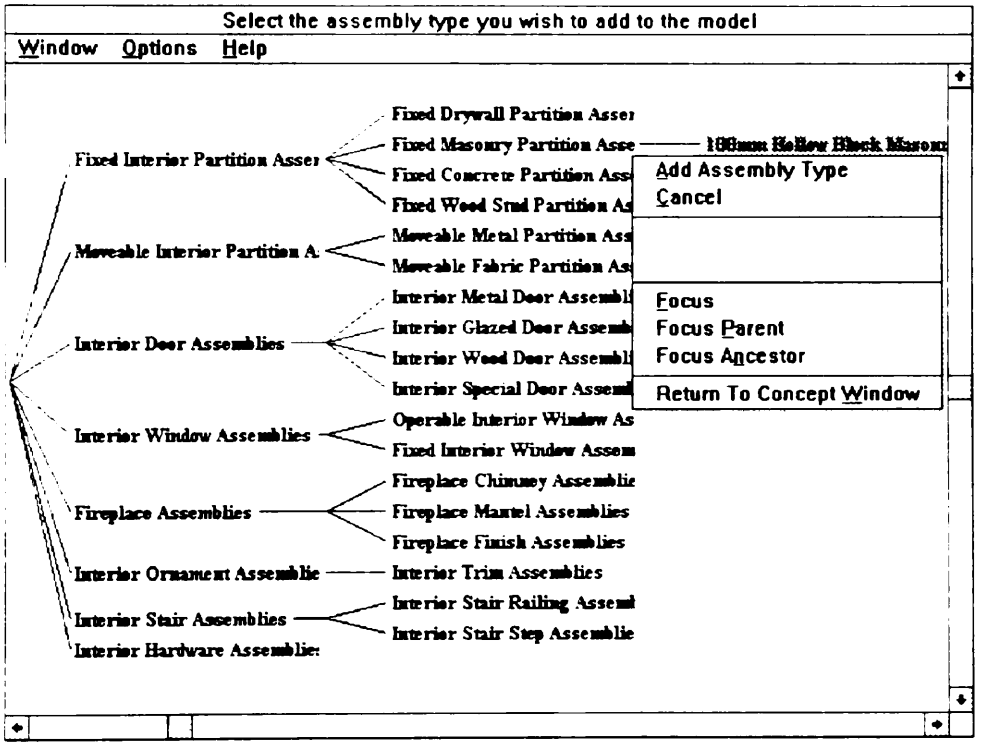


Figure 4.17 DRIVE Interface for Creating BPH Objects

BPH Objects can also have an AutoCAD graphical model representation. Figure 4 18 shows the DRIVE interface for creating a BPH Object with an AutoCAD model representation. Typing in an object name in the New AutoCAD Object Name text box, selecting the proper graphical layout, typing in the values for the selected layout, selecting the proper building story and clicking the Create Object button takes the user to the AutoCAD window shown in Figure 4 19. Figure 4 20 shows the floor plan for the example project. Once in the AutoCAD window, the system asks the user to select the corner position and rotation about the z-axis for the new AutoCAD object. This completes the object creation task.

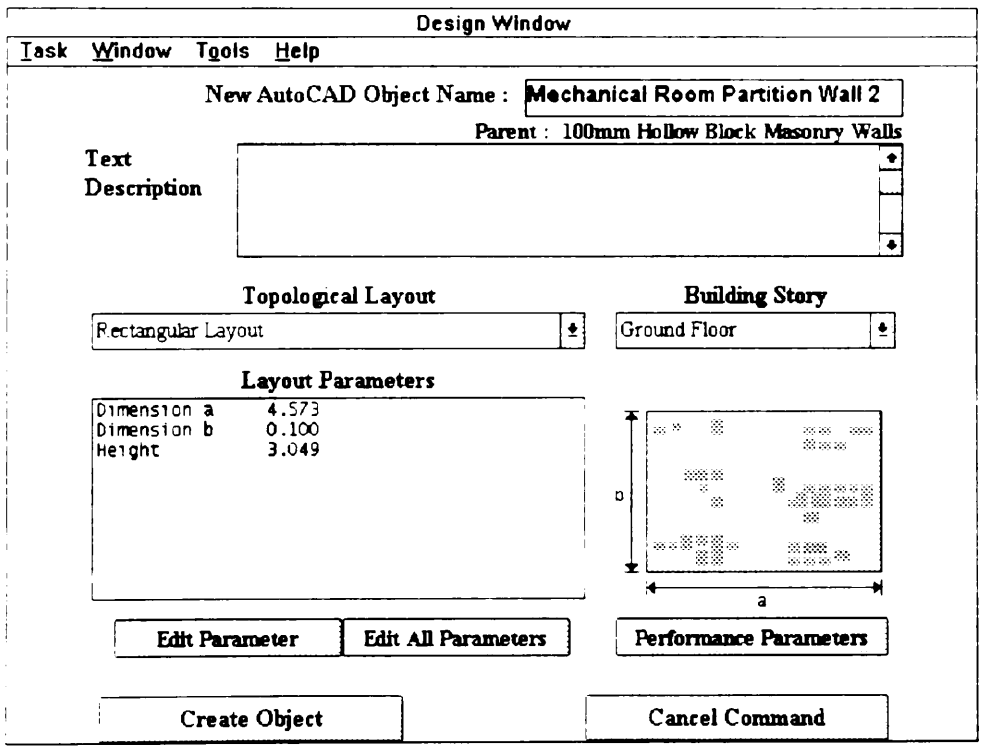


Figure 4 18 DRIVE Interface for Creating AutoCAD BPH Objects

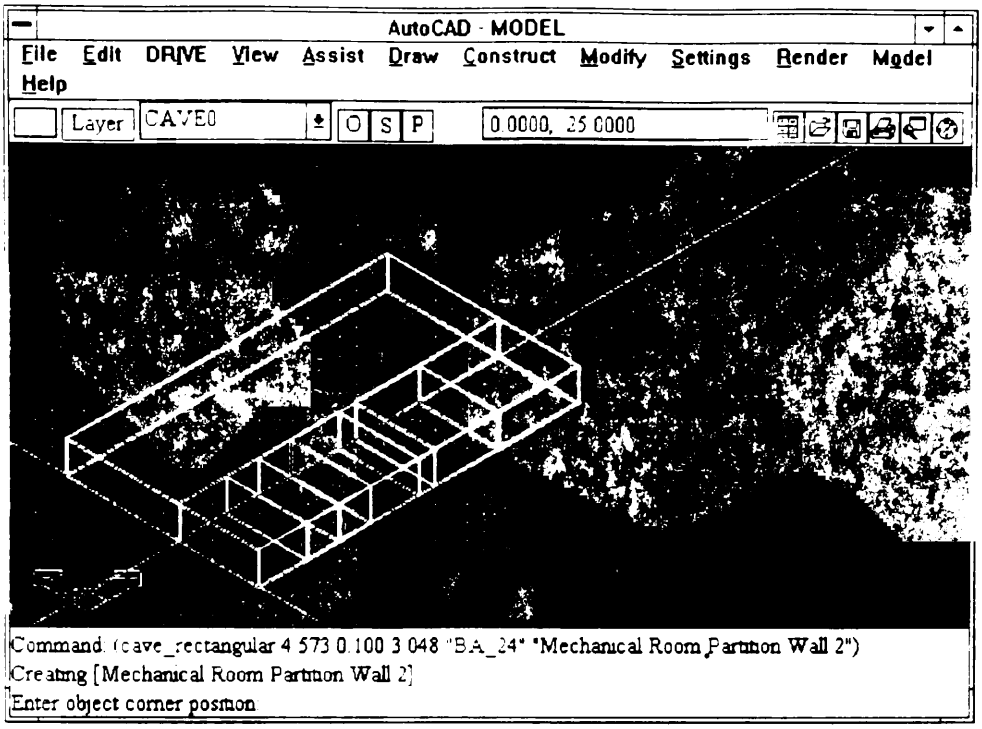


Figure 4.19 DRIVE AutoCAD Interface

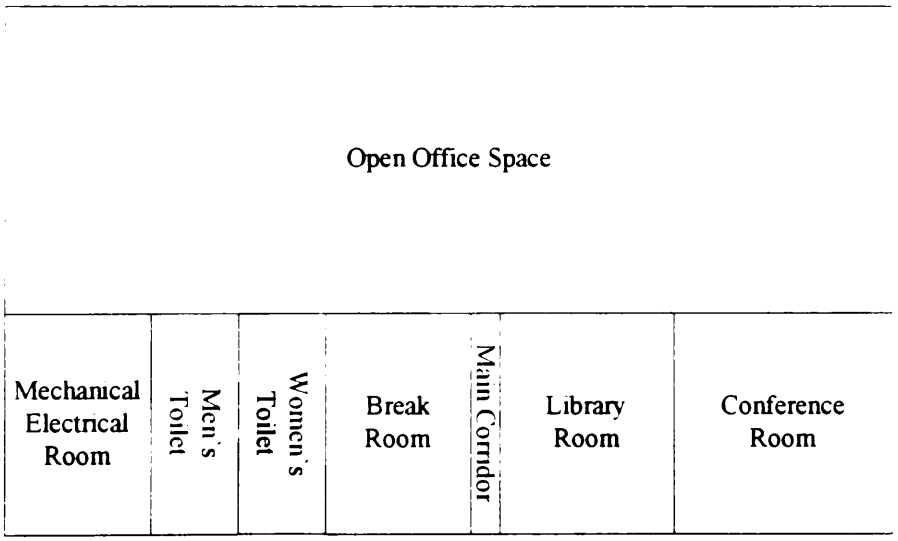


Figure 4.20 Example Project Floor Plan

Users can also set the values of the non-graphical performance parameters of a BPH Object. Clicking the **Performance Parameters** button in Figure 4.18 takes the user to the DRIVE screen shown in Figure 4.21. The **Text Description** edit box shown in Figure 4.21 allows users to type in some text describing the object currently under consideration. Selecting a performance parameter, highlighting a parameter and clicking the **Edit Parameter** button brings up the DRIVE window shown in Figure 4.22. This figure shows DRIVE's standard input window for parameter values and constraints. The various edit boxes in this window allow users to type in the appropriate values or constraints for a parameter of a BPH object.

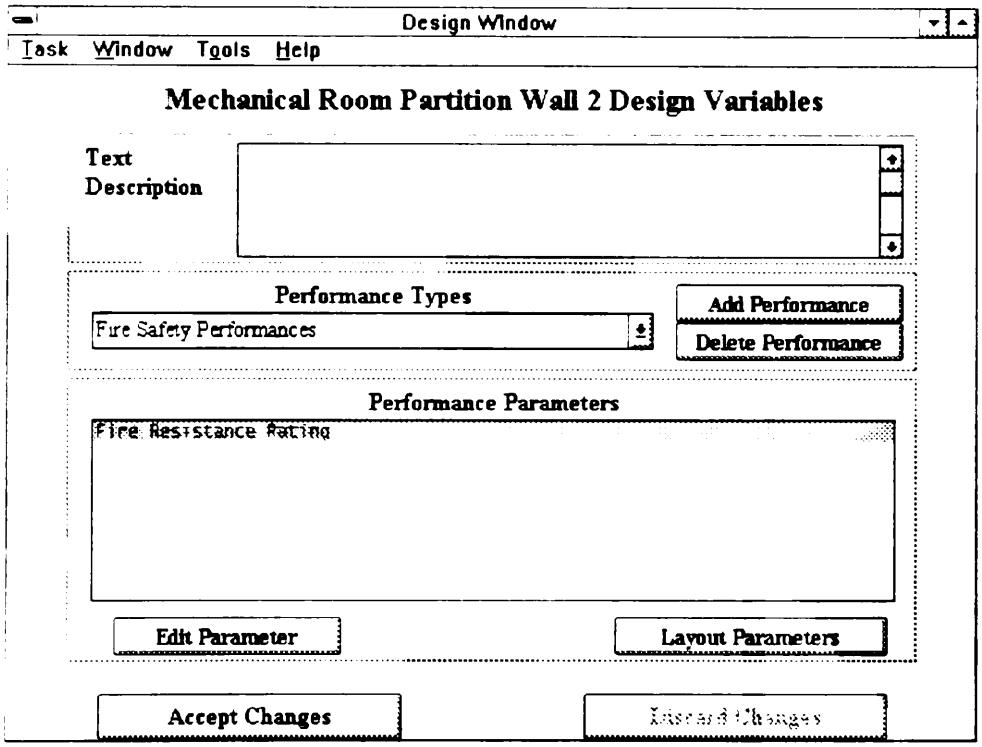


Figure 4.21 DRIVE Interface for Modifying Parameters

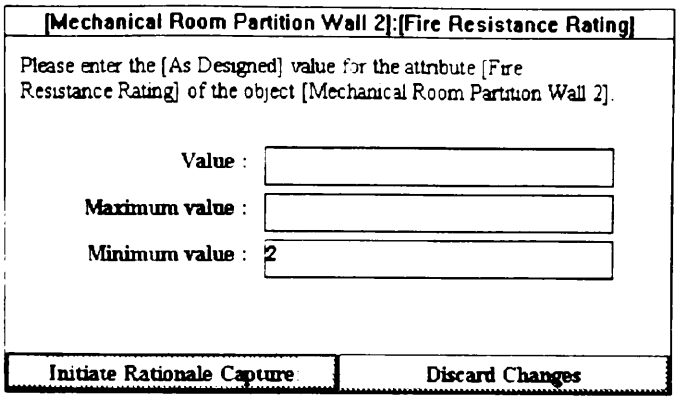


Figure 4 22 DRIVE Interface for Setting Parameter Values and Constraints

A BPH Object also has several instances denoting its life cycle phases. The user enters the values for the parameters of these instances through DRIVE's standard input window for parameter values and constraints shown in Figure 4 22. The text prompt in this window changes to reflect the life cycle phase, the parameter, and the object. DRIVE currently has two life cycle phase windows — the Concept Window and the Design Window. Accessing parameters through the Concept Window sets the life cycle stage to "as required" while doing it through the Design Window sets the stage to "as designed".

4.3.4 DRIVE Implementation

DRIVE stores the four BPHs (Building, Spaces Hierarchy, Assemblies Hierarchy, and Construction Trades Hierarchy) in one module file, namely, `bldgproj.kal`. DRIVE stores the AutoCAD model in the module file `model.dwg`. The modules `layouts.kal`, `cave.mnl`, and `cave.mnu` contain graphical layout definitions, menu items, and AutoLISP code allowing the system to create graphical model representations of BPH objects. The

modules `main.kal`, `concept.kal`, `design.kal`, and `objattr.kal` contain the interface objects allowing users to enter parameter values and constraints for BPH objects. The module file `descript.kal` contains a list of ASCII text files containing the text descriptions of the various BPH Objects.

DRIVE also uses the same internal-external name mapping system used by the PL (Section 4.1.4) for BPH Objects. Figure 4.23 shows the internal representation of the BPH Object described in Section 4.3.3 (DRIVE Interface). The slot `Title` stores the external name `Mechanical Room Partition Wall 2` representing the internal object name `BA_19`. As with the PL and the BCL, DRIVE users only work with external name representations of BPH Objects.

Figure 4.23 also shows the relationship of a BPH Object to various BCL and PL Objects. The `TypologicalParent` slot of BPH Objects stores a single BCL Object. The BPH Object `Mechanical Room Partition Wall 2` (`BA_19`) has a typological parent `100mm Hollow Block Masonry Walls` (`AssyType304`). As such, `Mechanical Room Partition Wall 2` copies the performances, parameters, values, and constraints defined in `100mm Hollow Block Masonry Walls`. Thus, because of its typological parent link, the `Mechanical Room Partition Wall 2` also has `Acoustical Performance` (`PerfType41`) and `Hygrothermal Performance` (`PerfType45`), the various parameters associated with these two performances, and the value `38` for `STC` (`PerfType41Attr1`).

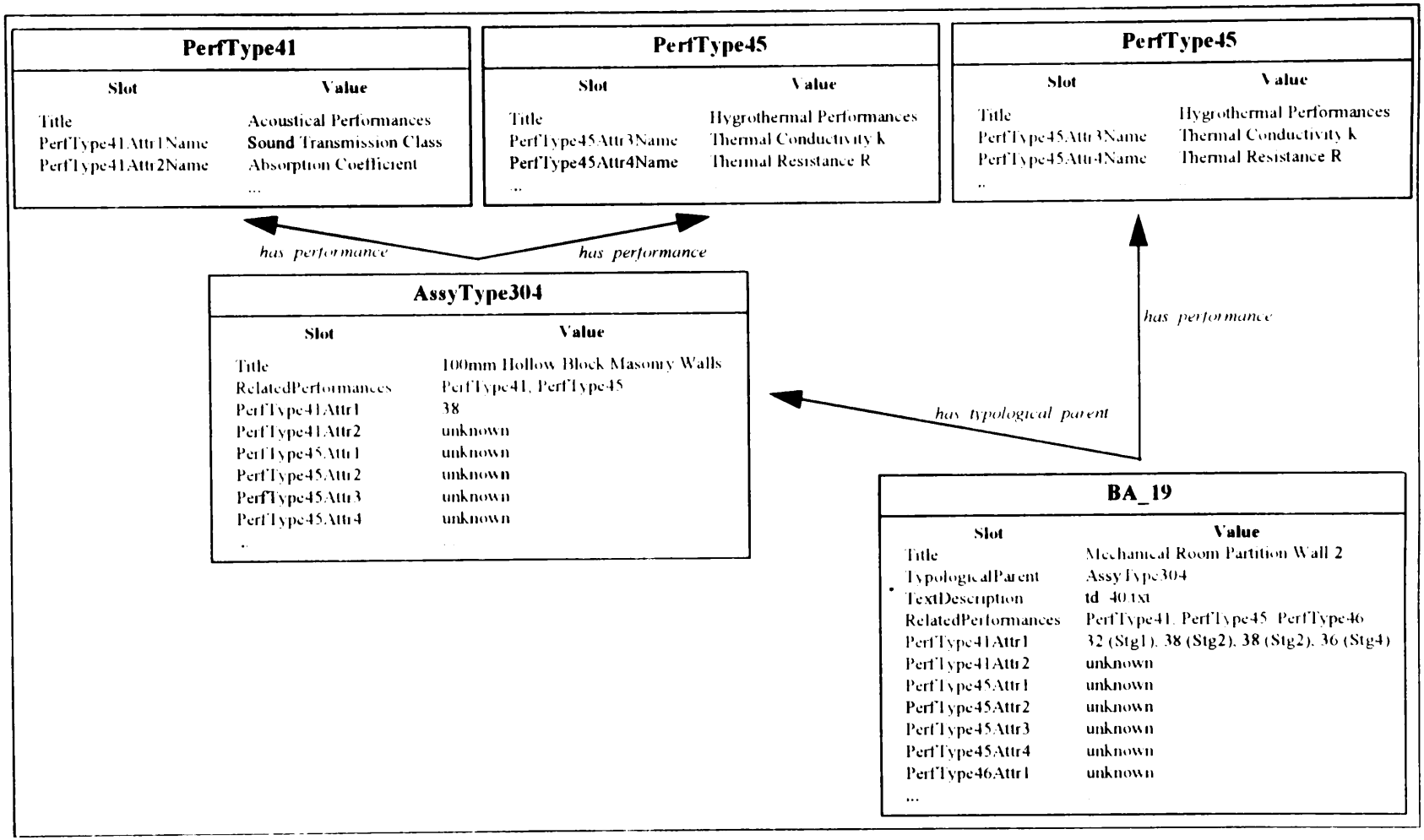


Figure 4.23 DRIVE Implementation of a BPH Object

A BPH Object contains information specific to a particular project. As noted earlier, BCL Objects are starting points for the design of BPH Objects. The information inherited by the BPH Object **Mechanical Room Partition Wall 2** from the BCL Object **100mm Hollow Block Masonry Walls** serves as an initial design for **Mechanical Room Partition Wall 2**. Seldom, if ever, does a BCL Object specify completely and exactly the design of a specific BPH object. Users usually need to modify the initial design parameters of a BPH Object. Figure 4.23 also illustrates this point. The **RelatedPerformances** slot of **Mechanical Room Partition Wall 2** contains an additional performance, namely, **Fire Safety Performance (PerfType46)**. This additional performance creates additional parameters such as **Fire Resistance Rating (PerfType46Attr1)** for **Mechanical Room Partition Wall 2**. The **TextDescription** slot of **Mechanical Room Partition Wall 2** contains a reference to the ASCII text file **td_40.txt**. This file contains a text description of **Mechanical Room Partition Wall 2**.

Figure 4.24 shows a detailed diagram of the internal life cycle representation of the BPH Object **Mechanical Room Partition Wall 2**. A BPH Object has several instances corresponding to its life cycle phases. For the **Mechanical Room Partition Wall 2 (BA_19)** Object, these instances are **Mechanical Room Partition Wall 2 As Required (BA_19Stg1)** up to **Mechanical Room Partition Wall 2 As Demolished (BA_19Stg7)**. The use of different instances for the different life cycle stages allows DRIVE to store information showing how a parameter changes in the course of time.

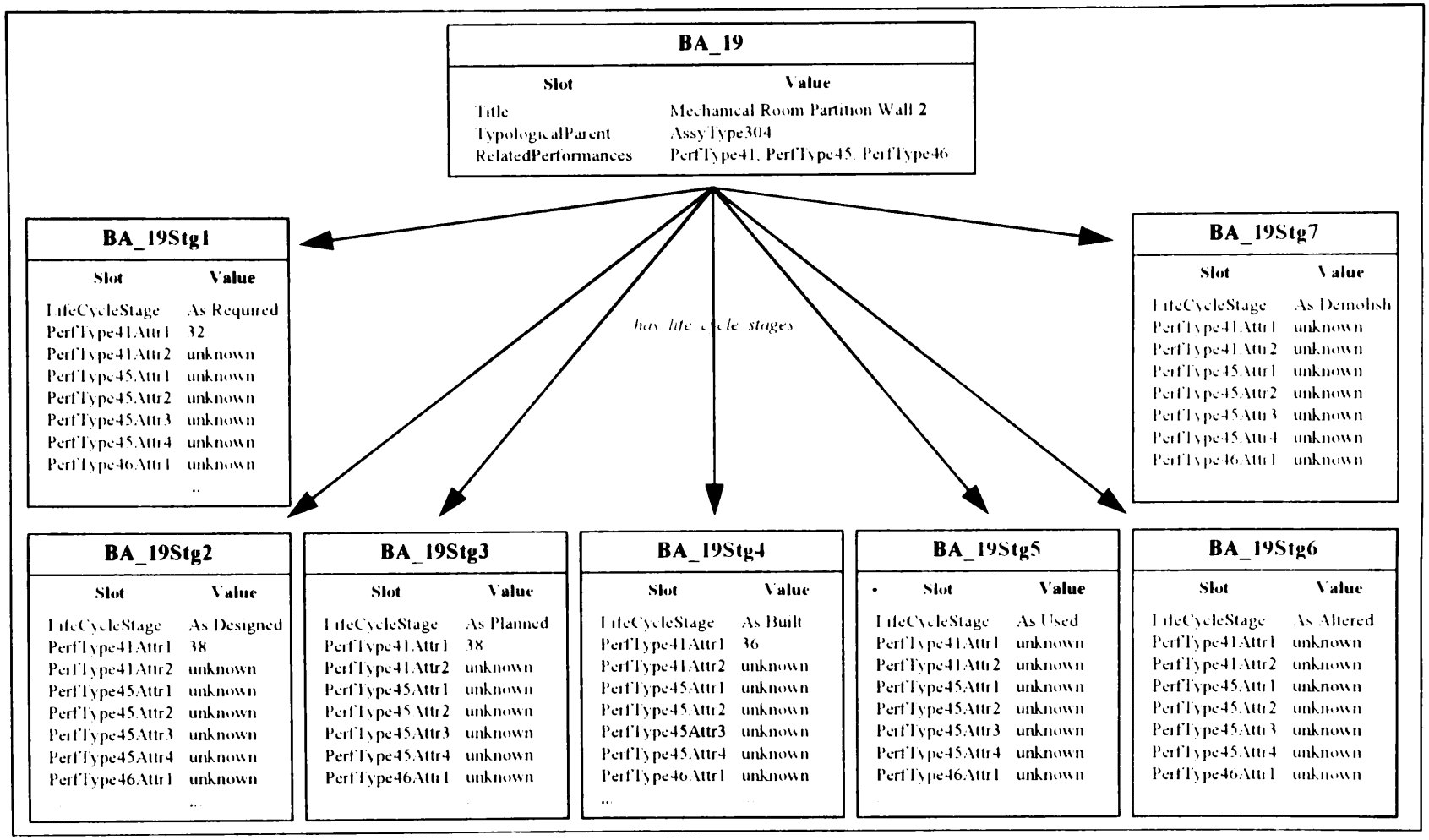


Figure 4.24 DRIVE Implementation of Life Cycle Phases for a BPH Object

CHAPTER 5

DATA STRUCTURES 2: RATIONALE STORAGE MODULE

The Rationale Storage Module acts as a storage module for process knowledge. Process knowledge refers to information about the process leading to the definition of a product description. One aspect of process knowledge is rationale. For building facilities, rationale starts accumulating from its initial conceptual definition to its final demolition. As stated in the research limitations (Section 1.3), this dissertation focuses only on rationale generated during the design stage. The primary objective of this dissertation is the creation of a data structure capable of representing design rationale. The Knowledge Representation Module (Chapter 4) coupled with the Rationale Storage Module comprise this data structure. Figure 5.1 succinctly displays the structure of the RSM.

The RSM can represent two types of design rationale, namely structured and unstructured. A computer system can perform analytical operations on structured design rationale. A computer system can only perform simple storage and retrieval operations on unstructured design rationale. Parameter relationships (Section 5.2) and text descriptions (Section 5.3) are examples of structured and unstructured design rationale, respectively.

This chapter follows the four-part breakdown structure used in the previous chapter. This breakdown structure discusses the RSM from four angles, namely, theory, practical example, DRIVE interface, and DRIVE implementation. Readers can select which angle or angles they wish to read.

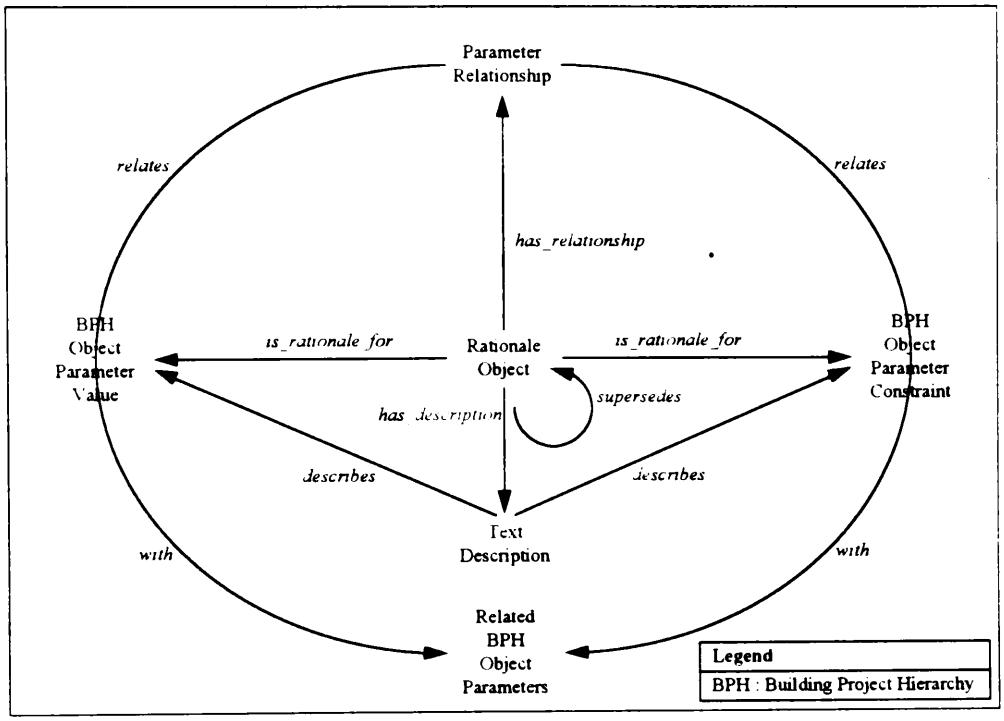


Figure 5.1 RSM Semantic Net Representation

5.1 RATIONALE OBJECTS

Both KRM and RSM use OOP representation techniques. As such, objects act as storage units for design rationale. The white background portion of Figure 5.2 denotes the specific portions of the RSM structure discussed in this section.

5.1.1 Theory

The BPHs (Section 4.3) stores information specific to a particular project. The BPH Object Parameter Values and BPH Object Parameter Constraints are the actual entities storing this project-specific information. Setting specific values and constraints for the various object parameters is the RSM's triggering mechanism for creating Rationale Objects. Defining parameter values and constraints are examples of design decisions. Thus, every design decision has design rationale behind it. Figure 5.2 denotes this rationale object creation mechanism through the *is_rationale_for* links from a Rationale Object to both a BPH Object Parameter Value and a BPH Object Parameter Constraint.

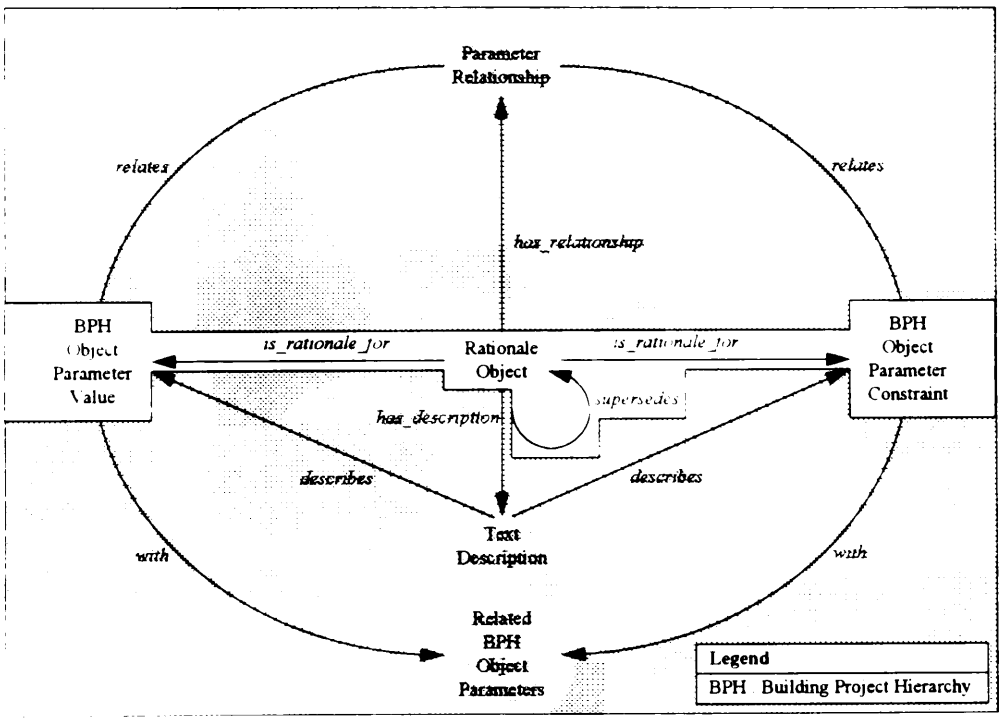


Figure 5.2 Creating Rationale Objects

Appendix A shows the Backus-Naur Form (BNF) grammar representation (Marcotty and Ledgard 1986) for a rationale object. This chapter makes several references to this appendix. This chapter denotes these references using the letters BNF and a reference number all enclosed in square brackets. For example, [BNF2] refers to the BNF representation of a design decision.

The RSM captures rationale for every design decision. [BNF2] through [BNF14] shows the complete BNF grammar representation of a design decision. The various design decision attributes of a Rationale Object provide a mechanism for referencing design decisions. Figure 5.3 shows these attributes. The `DecisionObject` [BNF5] attribute contains the name of the design object whose parameter triggered the rationale object creation process. The `DecisionParameter` [BNF6] attribute contains the name of this parameter. The `DecisionStage` attribute refers to the life cycle stage of the decision. Changing a BPH Object Parameter Value triggers the rationale object creation process. The `DecisionOldValue` [BNF8] and `DecisionNewValue` [BNF9] attributes store the value of this parameter before and after the change. Figure 4.1 shows three types of BPH Object Parameter Constraints. Changing any one of these constraints also creates rationale objects. The `DecisionConstraintType` [BNF10] attribute identifies the constraint type. The `DecisionOldConstraint` [BNF11] and `DecisionNewConstraint` [BNF12] attributes function similarly to their BPH Object Parameter Value counterparts. Section 5.1.2 provides an example illustrating the assignment of values to these attributes.

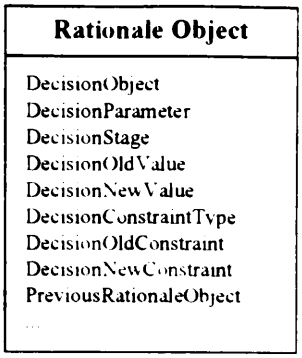


Figure 5.3 Partial Internal Structure (Design Decision Attributes) of a Rationale Object

The *supersedes* circular link of a Rationale Object stores the evolution of an object parameter within a particular life cycle stage. Initially, all BPH Object Parameters have their values and constraints set to unknown. Specifying a value or constraint creates a new Rationale Object. Sometimes, changing this specified value or constraint becomes necessary. Such a change creates a new Rationale Object superseding the original Rationale Object referring to a particular BPH Object Parameter Value or BPH Object Parameter Constraint. A BPH Object Parameter only stores its current value or constraint for each life cycle phase. Thus, a BPH Object Parameter is not capable of storing evolution information within a life cycle phase. However, the various attributes of a Rationale Object shown in Figure 5.3 assist in storing the evolution of a parameter value or constraint. The `PreviousRationale` attribute of a Rationale Object stores the name of the Rationale Object it supersedes. The `PreviousRationale`, `DecisionOldValue`, `DecisionNewValue`, `DecisionOldConstraint` and `DecisionNewConstraint` attributes trace the evolution of a parameter value or constraint. The next section contains an example illustrating this parameter evolution tracing capability.

5.1.2 Practical Example

The first life cycle phase is the "as required" stage. The original "as required" value of the parameter `STC` of the object `Mechanical Room Partition Wall 2` is unknown. Specifying an "as required" value of `32` for this object parameter creates a new Rationale Object, `Mechanical Room Partition Wall 2 STC Requirement`. The left portion of Figure 5.4 shows some of the attribute-values of this Rationale Object. Changing the "as required" value from `32` to `36` creates a new Rationale Object, `Revised Mechanical Room Partition Wall 2 STC Requirement`. This Rationale Object supersedes the Rationale Object `Mechanical Room Partition Wall 2 STC Requirement` as shown in Figure 5.4

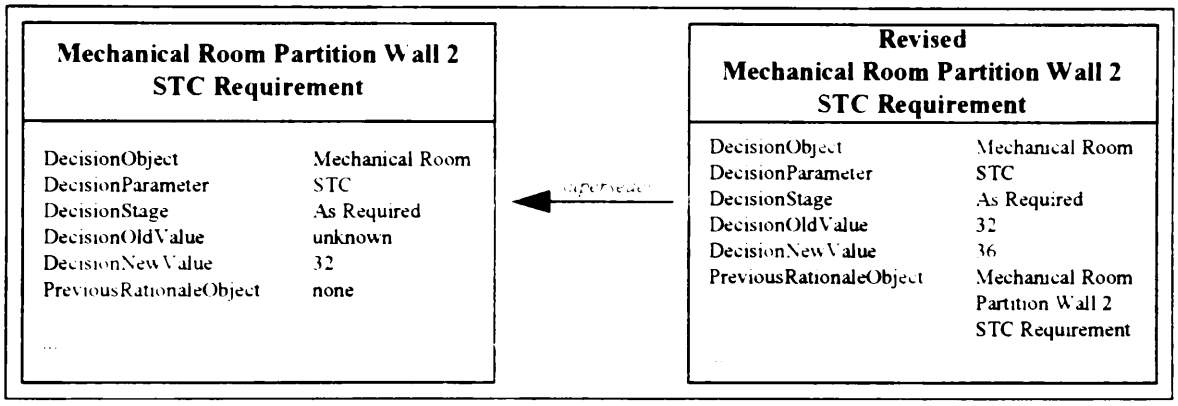


Figure 5.4 Parameter Evolution

The two Rationale Objects shown in Figure 5.4 exemplify the parameter evolution capability of the RSM. The attributes `DecisionObject`, `DecisionParameter`, and `DecisionStage` for both Rationale Objects have the following values: `Mechanical Room Partition Wall 2`, `STC`, and `As Required`, respectively. The `Revised Mechanical Room Partition Wall 2 STC Requirement` has a `PreviousRationale` attribute-value of

`Mechanical Room Partition Wall 2 STC Requirement`. The revised Rationale Object also has a `DecisionNewValue` and `DecisionOldValue` of `36` and `32`, respectively. The original one has values of `32` and `unknown` for these attributes.

5.1.3 DRIVE Interface

The rationale capture process of DRIVE involves answering a series of dialog boxes. These dialog boxes assist in capturing design rationale. Figure 5.5 shows the first dialog box in the series. This section describes only the process of creating Rationale Objects. Sections 5.2 and 5.3 further describe the rationale capture process.

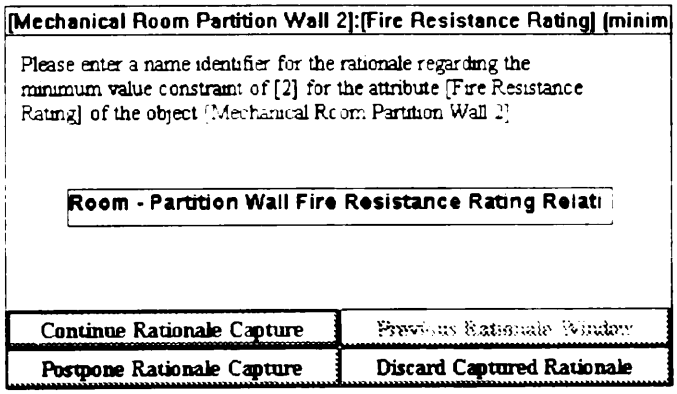


Figure 5.5 DRIVE Interface for Assigning Names to Rationale Objects

DRIVE has three modes for capturing design rationale — `Fully Automatic`, `Next Assertion Only`, and `Off`. The `Fully Automatic` mode triggers the rationale capture process every time a value or constraint changes for a design object parameter. The `Next Assertion Only` mode triggers the rationale capture process only for the next value or

constraint change and then it switches itself to the `Off` mode. The `Off` mode does not trigger the rationale capture process. Rather, it places the parameter value or constraint change into a stack of design decisions having incomplete rationale information. DRIVE terms this stack as pending rationale.

There are two ways of accessing the rationale capture dialog boxes. Figure 5.6 shows the DRIVE interface for processing the pending rationale list. This list contains all the design decisions (i.e., parameter value or constraint changes) having incomplete rationale information. Highlighting an item on this list and clicking the `Process Rationale` button brings up a rationale capture dialog box similar to the one shown in Figure 5.5.

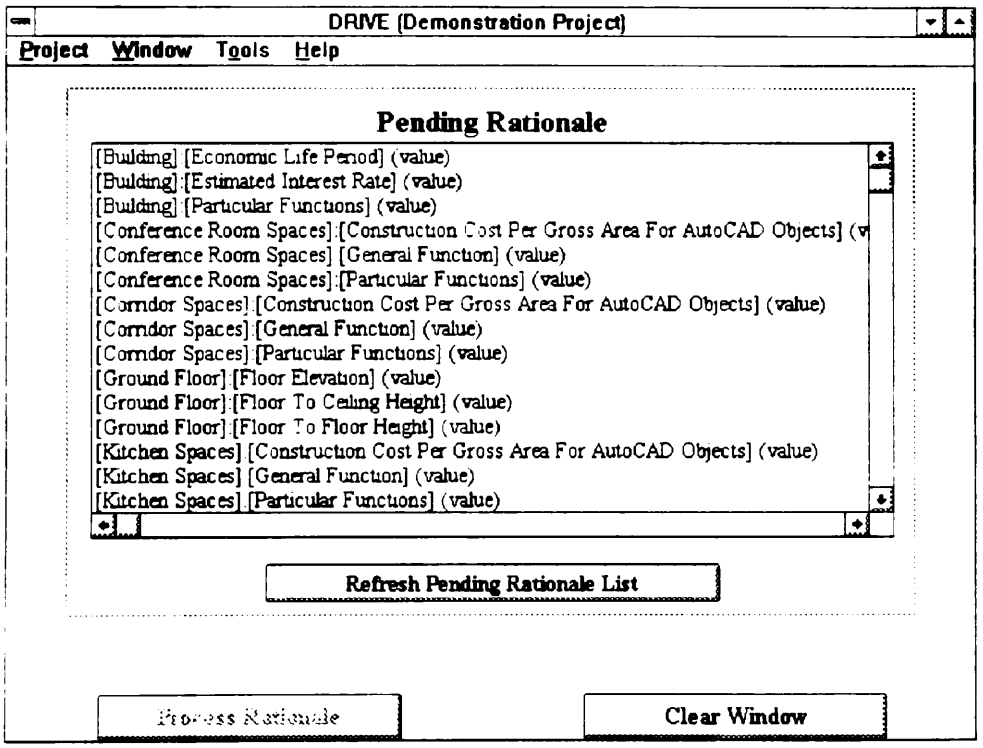


Figure 5.6 DRIVE Interface for Processing Pending Rationale

The other method of accessing the rationale capture dialog boxes is by changing a parameter value or constraint while in the `Fully Automatic` or `Next Assertion Only` rationale capture modes. Figure 4.22 shows the basic parameter value and constraint input window of DRIVE. Changing one of the values in the edit boxes in this window enables the `Initiate Rationale Capture` button. Clicking this button displays the rationale capture dialog box shown in Figure 5.5.

5.1.4 DRIVE Implementation

DRIVE creates a new Rationale Object every time a parameter value or constraint changes. DRIVE stores these Rationale Objects in the module file, `assertns.kal`. Figure 5.7 shows the Assertions Hierarchy. DRIVE calls design decisions as Assertions and stores design rationale in the Assertions Hierarchy.

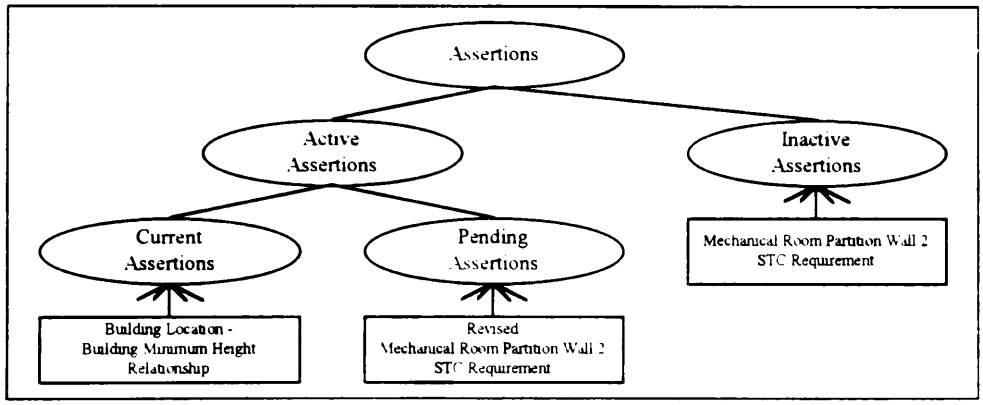


Figure 5.7 Assertions Hierarchy

There are two major types of Assertions — Active and Inactive. Active Assertions store rationale for the current value or constraint of a design object parameter. Inactive

Assertions store rationale for previous values or constraints. Figure 5.7 shows the DRIVE implementation of the example discussed in Section 5.1.2 and shown in Figure 5.4. `Revised Mechanical Room Partition Wall 2 STC Requirement` is an example of an Active Assertion. It stores rationale for the current value `36` of the parameter `STC` of the object `Mechanical Room Partition Wall 2`. This active assertion also supersedes the inactive assertion `Mechanical Room Partition Wall 2 STC Requirement`. This inactive assertion stores the rationale for the previous value of `32` for the same object parameter.

There are also two types of Active Assertions — Current and Pending. All Assertion Objects in DRIVE have several attributes representing rationale information. Figure 5.3 shows only a partial list of these attributes. Sections 5.2 and 5.3 further describe other attributes. Current Assertions are assertions having values defined for all of its attributes. In other words, Current Assertions are assertions having complete rationale information. Pending Assertions are assertions having incomplete rationale information.

The Assertions Hierarchy also uses the internal-external naming scheme used in the KRM. Figure 5.8 shows a partial structure of the DRIVE implementation of a Rationale Object. The internal name `A_28` corresponds to the external name `Revised Mechanical Room Partition Wall 2 STC Requirement`. Similar to the KRM, the RSM exposes users to only the external name representations. The other internal attributes shown in Figure 5.8 correspond to the attributes shown in the Rationale Object model (Figure 5.3).

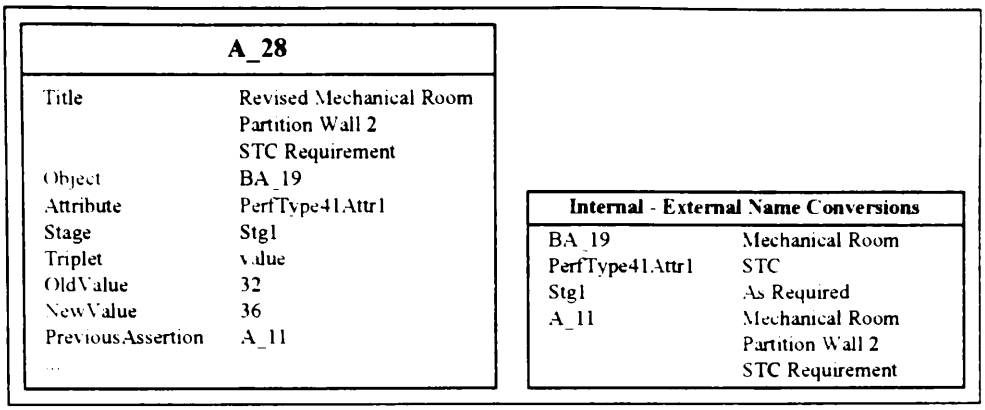


Figure 5 8 Partial Structure (Design Decision Attributes) of the DRIVE Implementation of a Rationale Object

5.2 STRUCTURED RATIONALE

Structured design rationale allows a computer system to perform analytical operations on it. Parameter relationships are examples of structured design rationale. Relationships among the various design object parameters constitute a significant amount of rationale generated during the design stage. Some examples of these relationships are:

- The width of the doors of an assembly gathering space depends on its estimated number of occupants.
- The level of lighting required for a space depends on the tasks performed by its occupants.
- The required floor slab strength depends on the weight of the machinery equipment is supports.

Traditional information exchange mechanisms, such as design drawings and specifications implicitly represent these design rationale. Explicitly stating these relationships removes any opportunities for misinterpretation of the intended design rationale.

Storing design rationale in data structures does not automatically imply that a computer system can perform analytical operations on the stored design rationale. Simple data structures only allow computers to merely perform database searches. Storing design rationale in a structured format, such as in parameter relationships, allows a computer system to perform analytical operations. The white background portion of Figure 5.9 denotes the portion of the RSM data structure supporting parameter relationships.

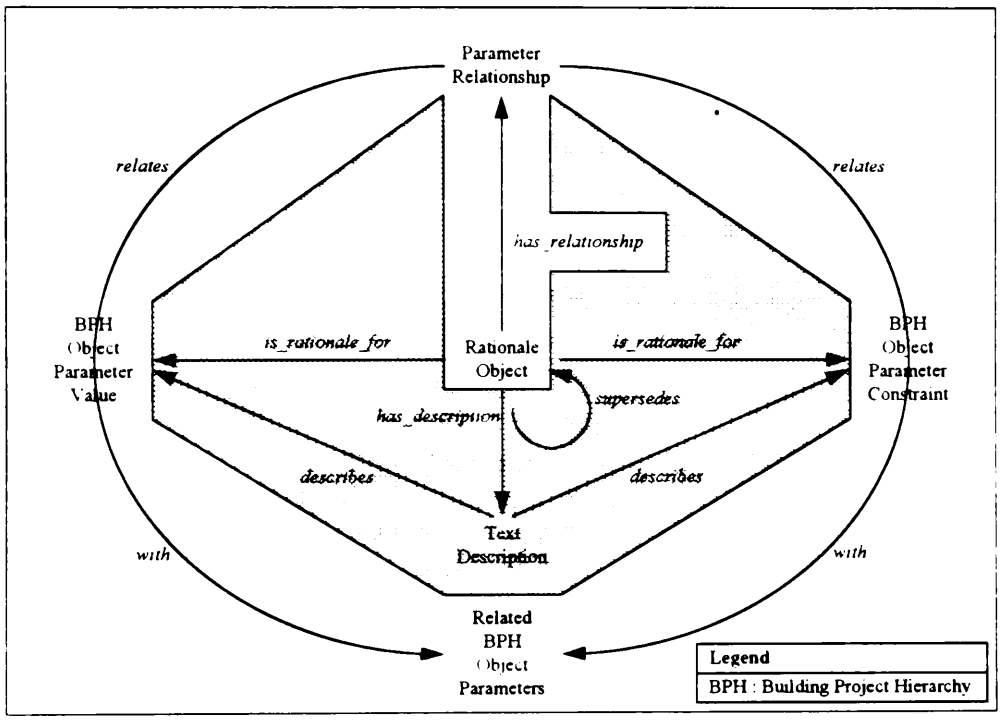


Figure 5.9 Parameter Relationships as Rationale

5.2.1 Theory

Rationale Objects act as storage units for Parameter Relationships. Figure 5.9 denotes this with the *has_relationship* link. Parameter Relationships *relate* a single BPH Object Parameter Value or BPH Object Parameter Constraint with one or more BPH Object

Parameters. A Parameter Relationship has three components, namely, a list of related object parameters, a relationship type, and a relationship. Figure 5.10 shows a partial internal structure of a Rationale Object containing the parameter relationship attributes.

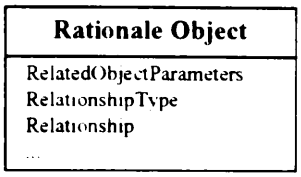


Figure 5.10 Partial Internal Structure (Parameter Relationship Attributes) of a Rationale Object

The `RelatedObjectParameters` [BNF17] attribute is a list of all BPH Object Parameters affecting a particular BPH Object Parameter Value or BPH Object Parameter Constraint. This attribute of a Rationale Object generates the **P**arameter **D**ependency **N**etwork (PDN) discussed in Section 5.4.2. The `RelationshipType` [BNF18] attribute defines the type of the `Relationship` [BNF19] existing between the Related BPH Object Parameters and either the BPH Object Parameter Value or the BPH Object Parameter Constraint. There are two types of relationships — Logical and Mathematical. Logical Relationships [BNF20] relate object-parameters using If-Then-Else constructs. Mathematical Relationships [BNF21] relate object-parameters using mathematical comparison operators, such as, less than, equal to, and greater than, among others. The Logical and Mathematical Relationships require the use of an interpreted language and a command interpreter. An interpreted language allows users to generate computer-understandable representations of Logical and Mathematical Relationships. A command interpreter

allows a computer system to understand these representations. Section 5.2.4 provides a detailed description of one possible implementation of an interpreted language and command interpreter combination for defining parameter relationships.

5.2.2 Practical Examples

5.2.2.1 Logical Relationship Example

One function of the Mechanical Room is to store HVAC equipment. The parameter Fire Resistance Rating depends on this function. The text shown on page 83 describes the rationale for the `Mechanical Room General Function - Mechanical Room Fire Resistance Rating Relationship`. In If-Then-Else format, this rationale translates to: If the value of the attribute `General Function` of the object `Mechanical Room` is equal to `House HVAC Equipment`, Then the minimum value of the attribute `Fire Resistance Rating` of the object `Mechanical Room` is not less than `2 hours`. The minimum value `2` of the attribute `Fire Resistance Rating` of the object `Mechanical Room` is an example of a BPH Object Parameter Constraint in Figure 5.9. The attribute `General Function` of the object `Mechanical Room` is an example of a Related BPH Object Parameter in Figure 5.9.

5.2.2.2 Mathematical Relationship Example

The `Partition Wall 2` separates the `Mechanical Room` and the `Open Office Space`. The parameter `Fire Resistance Rating` of the object `Partition Wall 2` depends on the parameter `Fire Resistance Rating` of the object `Mechanical Room`. The text shown on page 83 describes the rationale for the `Mechanical Room Fire Resistance Rating -`

`Partition Wall 2 Fire Resistance Rating Relationship` In Mathematical Relationship format, this rationale translates to: the value of the attribute `Fire Resistance Rating` of the object `Partition Wall 2` is not less than the value of `Fire Resistance Rating` of the object `Mechanical Room`. The minimum value `2` of the attribute `Fire Resistance Rating` of the object `Partition Wall 2` is an example of a BPH Object Parameter Constraint in Figure 5.9. The attribute `Fire Resistance Rating` of the object `Mechanical Room` is an example of a Related BPH Object Parameter in Figure 5.9.

5.2.3 DRIVE Interface

The rationale capture process of DRIVE involves answering several dialog boxes. Figure 5.11 shows the DRIVE interface for defining the related parameters list. It is the second dialog box in a two-part series for defining the related parameters list. The first dialog box asks the user to highlight all the relevant objects to the current rationale. The list box in this dialog box contains all the names of the design objects. Clicking the `Continue Rationale Capture` button in this dialog box brings up the dialog box shown in Figure 5.11. This dialog box asks the user to highlight all the relevant attributes to the current rationale. The list box in this dialog box contains all the names of the parameters of the selected objects. For the Logical Relationship example described in Section 5.2.2.1, the only relevant object is `Mechanical Room` and the only relevant attribute is `General Function`. The list boxes in both of these dialog boxes allow multiple selections. This multiple selection capability allows users to relate several different object parameters.

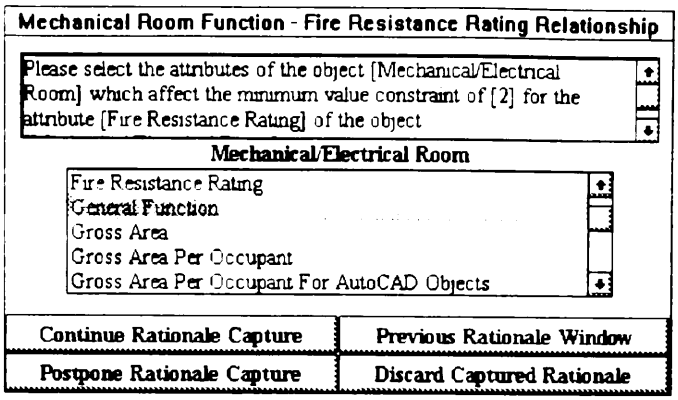


Figure 5.11 DRIVE Interface for Defining Related Object Parameters

Clicking the Continue Rationale Capture button in Figure 5.11 displays a new dialog box asking the user to select the type of relationship for the current rationale. Depending on this selection, the next dialog box is either the one shown on Figure 5.12 or in Figure 5.13. Figure 5.12 shows the DRIVE interface for creating Logical Relationships. Figure 5.13 shows the DRIVE interface for creating Mathematical Relationships. The six smaller buttons in the button bar at the lower portion of these two dialog boxes assist the user in graphically formulating the relationship. Each of these buttons generates relationship statements. The `ObjAttr` (short for Object Attributes) button creates a statement reference to a particular BPH Object Parameter Value. The `Constraints` button creates one for a particular BPH Object Parameter Constraint. The `Relations`, `Logical`, and `Math` buttons create a statement corresponding to a particular relational, logical, or mathematical operation. The Values button allows users to create a statement containing either a value already existing in an object parameter or a user-specified value.

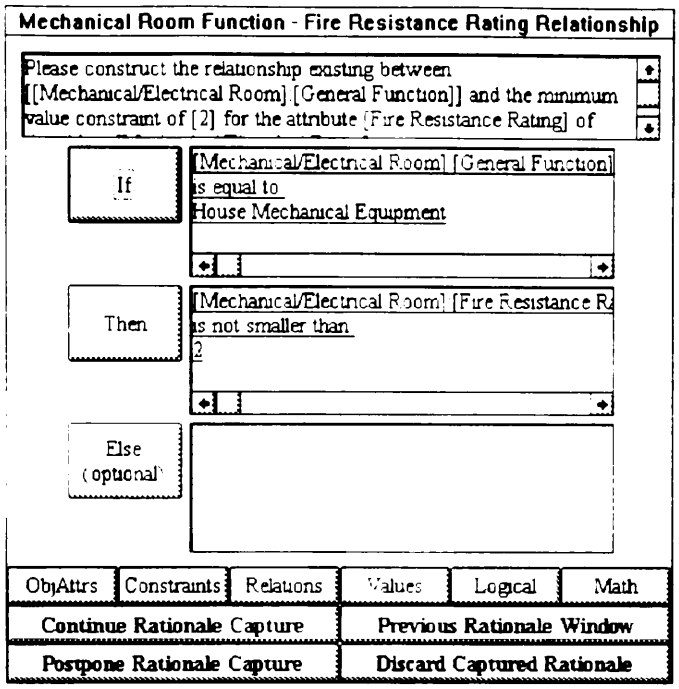


Figure 5.12 DRIVE Interface for Defining Logical Relationships

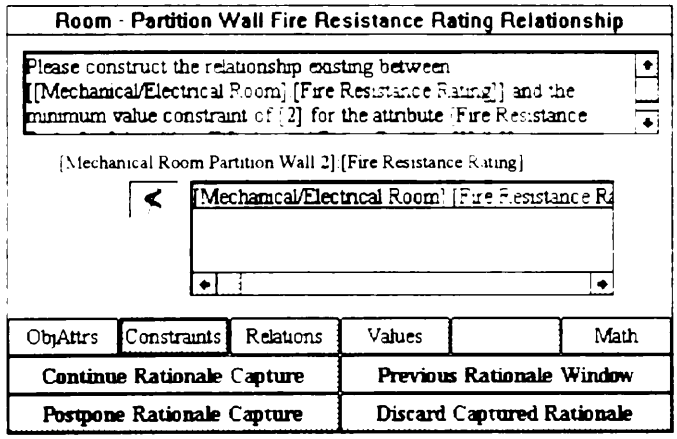


Figure 5.13 DRIVE Interface for Defining Mathematical Relationships

Clicking the Continue Rationale Capture button in either one of these dialog boxes invokes DRIVE's relationship syntax checker. If the syntax checker flags errors, DRIVE informs the user of the error and requests the user to modify the relationship definition. If

there are no errors in the relationship definition syntax, DRIVE displays another dialog box asking the user to type in a text description for the rationale. Section 5.3 further discusses this aspect of the RSM.

5.2.4 DRIVE Implementation

The module file `assertns.kal` stores the Rationale Objects for a particular project. The module file `rational.kal` generates the rationale capture dialog boxes allowing the user to define parameter relationships. The module file `rule.kal` allows users to create computer-understandable representations of logical and mathematical relationships. This is DRIVE's implementation of the interpreted language theoretical component (Section 5.2.1). The module file `rule.kal` also contains a relationship syntax checker, DRIVE's implementation of the command interpreter theoretical component (Section 5.2.1).

The three components of a Parameter Relationship are a related object parameters list, a relationship type, and a relationship. Figure 5.14 shows a partial structure of the DRIVE implementation of a Rationale Object. The attributes `SelectedObjectsList`, `Selected-ObjectsTitleList`, `SelectedAttributesList`, and `SelectedObjAttrTitleList` define the related objects parameters list. Figure 5.14 also shows the internal DRIVE representation of the Logical Relationship example used in Section 5.2.2.1.

Section 5.2.3 describes DRIVE's two-part interface for defining the related object parameters list. The first part asks the user to select which objects are relevant to the

current rationale. The `SelectedObjectsList` and `SelectedObjectsTitleList` attributes store the information captured in this dialog box. These two attributes store the internal and external names, respectively, of the selected objects. For the Logical Relationship example described in Section 5.2.2.1, the values for the attributes `SelectedObjectsList` and `SelectedObjectsTitleList` are `BS_8`, and `Mechanical Room`, respectively.

A_30

Title	Mechanical Room Fire Resistance Rating - Mechanical Room General Function Relationship
SelectedObjectsList	BS_8
SelectedObjectsTitleList	[Mechanical Room]
SelectedAttributesList	PerfType30Attr1
SelectedObjAttrTitleList	[Mechanical Room]:[General Function]
RelationshipType	Logical
IRelationshipLines	3
IRuleText1	BS_8:PerfType30Attr1
IRuleText2	=
IRuleText3	"House Mechanical Equipment"
IDisplayText1	[Mechanical Room]:[General Function]
IDisplayText2	is equal to
IDisplayText3	[House Mechanical Equipment]
TRelationshipLines	3
TRuleText1	BS_8:PerfType25Attr1
TRuleText2	-
TRuleText3	2
TDisplayText1	[Mechanical Room]:[Fire Resistance Rating]
TDisplayText2	is not smaller than
TDisplayText3	[2]
ERelationshipLines	0
MathematicalComparison	n/a
MRelationshipLines	n/a
...	

Internal - External Name Conversions

BS_8	Mechanical Room
PerfType30Attr1	General Function
PerfType25Attr1	Fire Resistance Rating

Figure 5.14 Partial Structure (Parameter Relationship Attributes) of the DRIVE Implementation of a Rationale Object

The second part of this two-part interface asks the user to select which attributes of the selected objects are relevant to the current rationale. The `SelectedAttributesList` and `SelectedObjAttrTitleList` attributes store the information captured in the dialog box

shown in Figure 5.11. These two attributes store the internal and external names, respectively, of the Related BPH Object Parameters. For the Logical Relationship example described in Section 5.2.2.1, the values for the attributes `SelectedAttributes-List` and `SelectedObjAttrTitleList` are `PerfType30Attr1` and `[Mechanical Room]: [General Function]`, respectively.

The attribute `RelationshipType` defines the type of relationship existing between the selected object parameters and the BPH Object Parameter Value or BPH Object Parameter Constraint. DRIVE currently has only two types of relationships available, namely, Logical and Mathematical. As such, these are the only two values valid for the attribute `RelationshipType`.

Selecting Logical as the `RelationshipType` instructs DRIVE to create three sets of relationship attributes — one each for the If, Then, and Else portions of the Logical Relationship. DRIVE uses a prefix to denote these three sets — I, T, and E for the If, Then, and Else portions, respectively. Figure 5.14 shows these three sets. The `RelationshipLines` attributes store the length of a portion of the relationship. For example, the `IRelationshipLines` attribute stores the length of the If portion of the relationship. If a `RelationshipLines` attribute has a value greater than zero, DRIVE creates `RuleText` and `DisplayText` attributes. These two attributes store the internal code instructions and the external text representations, respectively. Users need to generate computer-understandable representations of the logical and mathematical

relationships. The six relationship definition buttons described in Section 5.2.3 accomplish this task. These six buttons fill in the appropriate `RuleText` and `DisplayText` attributes to generate the relationship. Figure 5.14 shows the internal DRIVE representation of the Logical Relationship example used in Section 5.2.2.1.

DRIVE processes a Mathematical Relationship in a slightly different manner. DRIVE only creates one set of relationship attributes, namely, the `M` (for mathematical) attributes. The `MRelationshipLines`, `MRuleText`, and `MDisplayText` attributes function similarly to their Logical Relationship counterparts. Logical Relationships require a logical comparison operation on the BPH Object Parameter Value or BPH Object Parameter Constraint triggering the rationale. Mathematical Relationships, on the other hand, compares the BPH Object Parameter Value or BPH Object Parameter Constraint directly with the `M` relationship attributes. As such, Mathematical Relationships only require one attribute, the `MathematicalComparison` to relate the BPH Object Parameter Value or BPH Object Parameter Constraint with the `M` relationship attributes.

5.3 UNSTRUCTURED RATIONALE

Text descriptions are examples of unstructured design rationale. Unlike parameter relationships (Section 5.2), the computer system developed in this dissertation does not perform any analytical operations on text descriptions. Rationale Text Descriptions function the same way as BPH Object Text Descriptions (Section 4.3.1.5). The intended

consumers of rationale text description information are human users. The white background portion of Figure 5.15 is the focus of the discussion in this section.

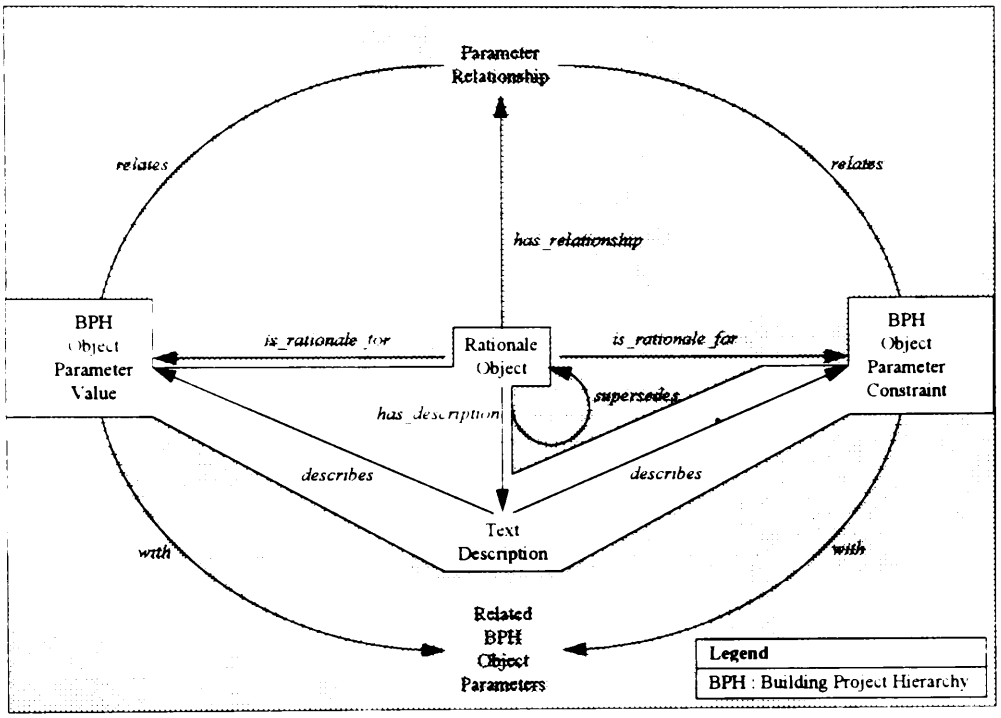


Figure 5.15 Text Descriptions as Rationale

5.3.1 Theory

Rationale text descriptions are paragraphs written in plain English to describe a Rationale Object. These descriptions supplement the rationale information conveyed by parameter relationships. In cases when parameter relationships are not suitable for representing design rationale, text descriptions provide a means of capturing design rationale. Examples of situations when text descriptions are appropriate include owner requirements, designer preferences, and code provisions.

All Rationale Objects have a Text Description. This is equivalent to stating all design decisions have a text description regarding its design rationale. A design decision occurs when a designer specifies a BPH Object Parameter Value or a BPH Object Parameter Constraint. Figure 5.15 denotes these relationships with the *has_description* and *describes* links. Figure 5.16 shows a partial internal structure of a Rationale Object containing the text description attribute. The `TextDescription` [BNF22] attribute stores the name of an ASCII text file containing the text description for the Rationale Object.

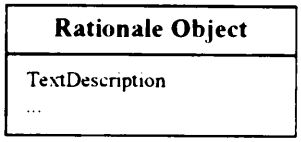


Figure 5.16 Partial Internal Structure (Text Description Attribute) of a Rationale Object

5.3.2 Practical Example

One example of a design decision from the example project (Section 3.6) is the decision to situate the Resident Engineers' Office / Visitors' Center Building on an existing rock knoll. This is an owner requirement. Section 3.6 shows the full text description of the design rationale for this design decision, as well as for several other design decisions.

5.3.3 DRIVE Interface

DRIVE supports two types of rationale — parameter relationships and plain text descriptions. Figure 5.17 shows the rationale capture dialog box where users select whether a rationale is a parameter relationship or a plain text description. Selecting either

`owner requirements` or `code requirements` and clicking the `Continue Rationale Capture` button displays the rationale text description dialog box shown in Figure 5.18. The edit box in Figure 5.18 allows users to type any amount of text necessary to describe the design rationale. Clicking the `Conclude Rationale Capture` button in Figure 5.18 completes the rationale capture process. The design rationale captured in Figure 5.18 corresponds to the example used in Section 5.3.2.

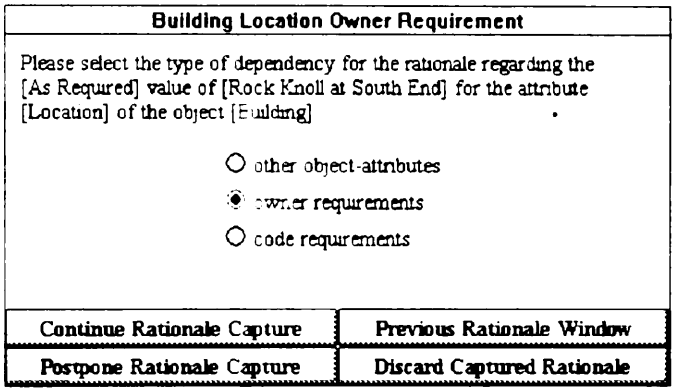


Figure 5.17 DRIVE Interface for Selecting Rationale Type

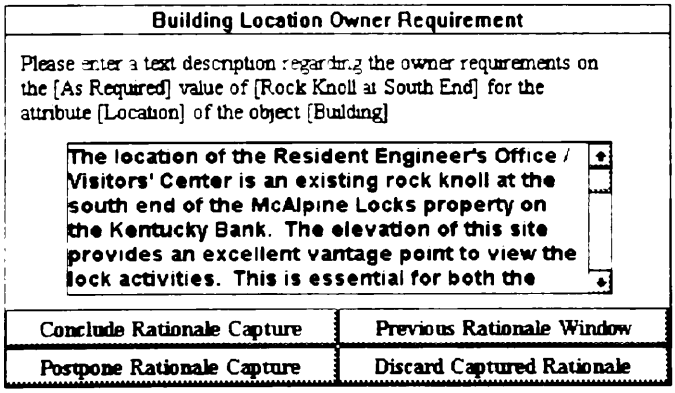


Figure 5.18 DRIVE Interface for Capturing Rationale Text Descriptions

In Figure 5.17, selecting `other object attributes` and clicking the `Continue Rationale Capture` button takes the user to the parameter relationship dialog boxes. Section 5.2.3 shows how parameter relationships reach a relationship definition dialog box (Figure 5.12 or Figure 5.13). Clicking the `Continue Rationale Capture` button in either of these two dialog boxes displays a text description dialog box similar to Figure 5.18. This allows users to supplement the parameter relationship with text descriptions.

5.3.4 DRIVE Implementation

One of the slots of a Rationale Object is `TextDescription`. This slot contains a reference to an ASCII text file containing the actual text description. The module file `descript.kal` contains a list of ASCII text files containing these text descriptions. Figure 5.19 shows an example of the DRIVE implementation of a rationale text description.

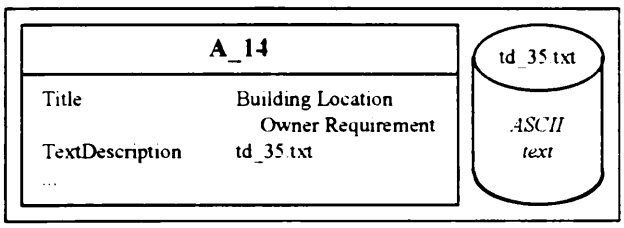


Figure 5.19 Partial Structure (Text Description Attributes) of the DRIVE Implementation of a Rationale Object

5.4 USING RATIONALE

The first three major sections of this chapter discussed the various components of the RSM. This section changes focus and concentrates on using these components to present, organize, and perform analytical operations on captured design rationale information.

This section has three sub-sections — (1) Simple Rationale Retrieval, (2) Parameter Dependency Network, and (3) Data Verification and Conflict Resolution. Each of these three sub-sections follow the familiar four-part breakdown structure consisting of: theory, practical example, DRIVE interface, and DRIVE implementation.

5.4.1 Simple Rationale Retrieval

Simple rationale retrieval involves presenting the captured rationale, both parameter relationships and text descriptions, in raw text form. Simple rationale retrieval does not perform any operations on design rationale. Rather, it merely presents information about a single Rationale Object in an easily understandable manner.

5.4.1.1 Theory

Rationale Objects store parameter relationships and text descriptions. The various attributes of a Rationale Object store design rationale information about a particular design decision. Simple rationale retrieval arranges the values of these attributes to form an English-like paragraph about the design rationale. These English-like paragraphs increase the human user's understanding of the design compared to inferring design rationale implicitly stored in design documents. Sections 5.4.1.3 and 5.4.1.4 give examples of simple rationale retrieval.

5.4.1.2 Practical Example

Design rationale has two forms — parameter relationships and text descriptions. Parameter relationships are of two types — logical relationships and mathematical relationships. Section 5.4.1.3 show example of simple rationale retrieval for a logical relationship, a mathematical relationship, and a text description.

5.4.1.3 DRIVE Interface

Figure 5.20 shows an example of simple rationale retrieval for the *logical relationship* described in Section 5.2.2.1. Figure 5.21 does the same for the *mathematical relationship* described in Section 5.2.2.2. Figure 5.22 corresponds to the rationale *text description* for the Minimum Ground Floor Elevation Requirement shown on page 83. DRIVE generates the items on the `Rationale List` box from all the instances of Current Assertions (Figure 5.7). Current Assertions are assertions having complete rationale information. Highlighting an item in the `Rationale List` box changes the text inside the lower test box to correspond to the design rationale of the highlighted item. Section 5.4.1.4 gives details on how DRIVE generates this text.

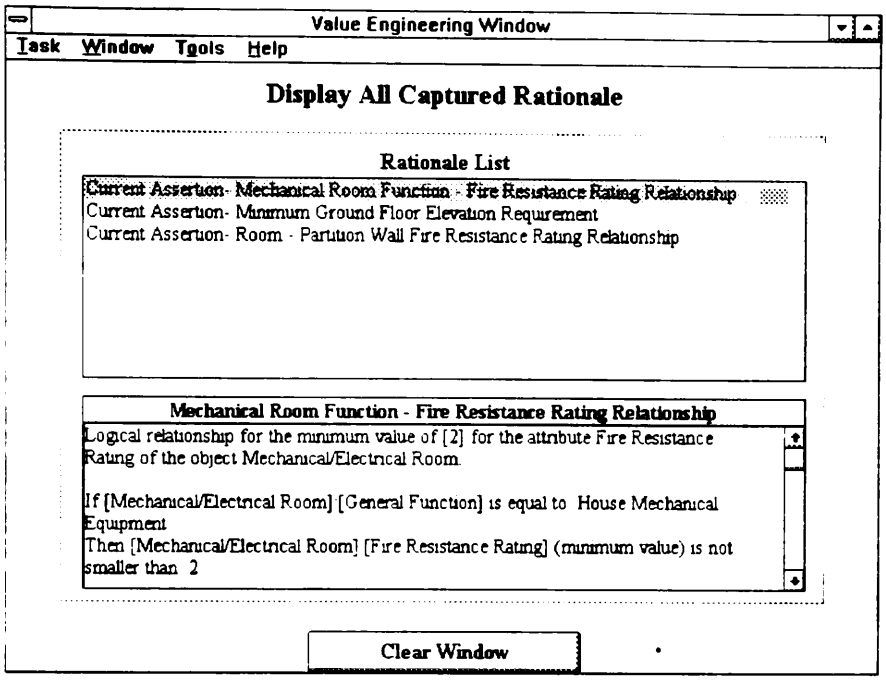


Figure 5.20 DRIVE Interface for Retrieving Logical Relationships

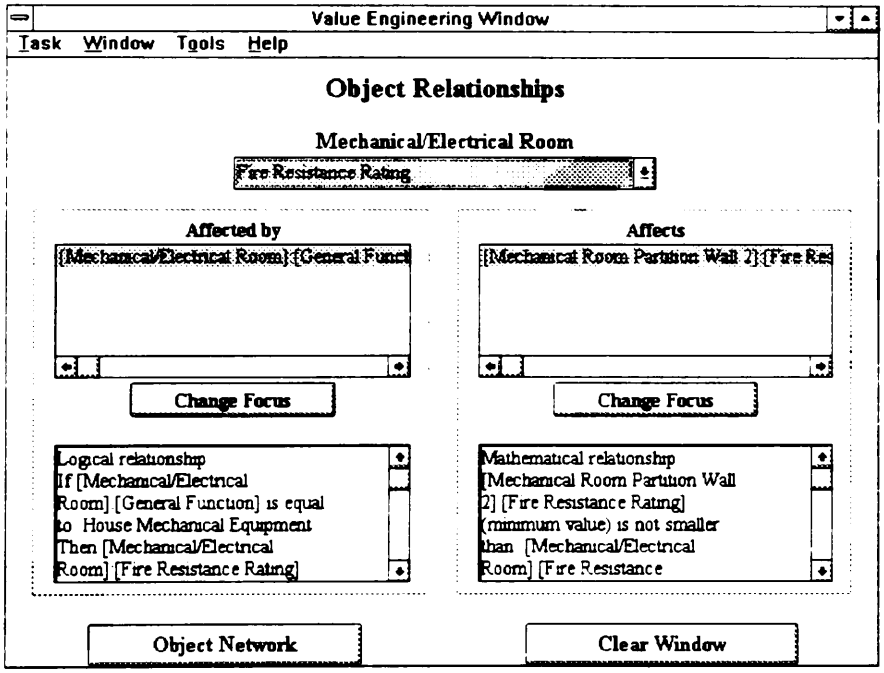


Figure 5.21 DRIVE Interface for Retrieving Mathematical Relationships

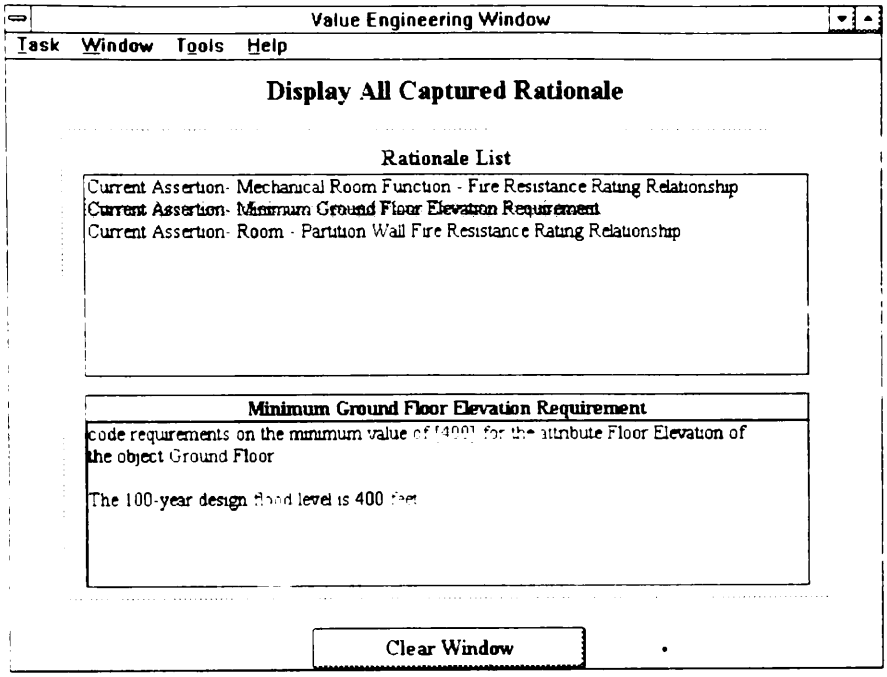


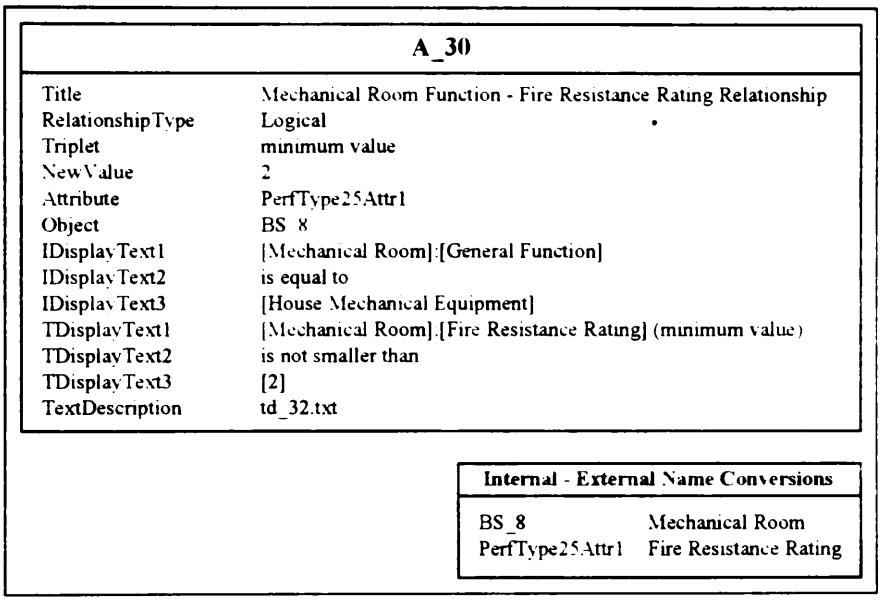
Figure 5.22 DRIVE Interface for Retrieving Text Descriptions

5.4.1.4 DRIVE Implementation

The module file **ve.kal** contains the interface objects necessary to present simple rationale retrieval information. The module files **assertns.kal** and **descript.kal** contain the Assertion objects and the list of text description files used in generating the text representing simple rationale retrieval information.

Figure 5.23 shows a partial structure of a DRIVE Assertion showing the slots relevant in constructing the simple rationale retrieval text on Figure 5.20. The arrangement of the slots in Figure 5.23 correspond to the order the values of these slots are in the text box of Figure 5.20. For example, the first slot in Figure 5.23 is **Title**. It has a value **Mechanical Room Function - Fire Resistance Rating Relationship**. This text

occurs as the title of the text box in Figure 5.20. The next slot in Figure 5.23 is `RelationshipType`. It has a value `Logical`. The first word inside the text box in Figure 5.20 is `Logical`. This goes on for the rest of the slots shown in Figure 5.23. DRIVE inserts pre-defined phrases at strategic points to present rationale information in an English-like manner. Examples of these pre-defined phrases include "relationship for the", "for the attribute", and "of the object".



A_30	
Title	Mechanical Room Function - Fire Resistance Rating Relationship
RelationshipType	Logical
Triplet	minimum value
NewValue	2
Attribute	PerfType25Attr1
Object	BS_8
IDisplayText1	[Mechanical Room]:[General Function]
IDisplayText2	is equal to
IDisplayText3	[House Mechanical Equipment]
TDisplayText1	[Mechanical Room].[Fire Resistance Rating] (minimum value)
TDisplayText2	is not smaller than
TDisplayText3	[2]
TextDescription	td_32.txt

Internal - External Name Conversions	
BS_8	Mechanical Room
PerfType25Attr1	Fire Resistance Rating

Figure 5.23 Partial Structure (Logical Relationship Textual Representation) of the DRIVE Implementation of a Rationale Object

5.4.2 Parameter Dependency Network

The **P**arameter **D**ependency **N**etwork (PDN) displays only parameter relationship rationale (Section 5.2). It does not handle simple rationale text descriptions (Section 5.3). The PDN organizes parameter relationship rationale information and presents it in an easily understandable and navigable manner.

5.4.2.1 Theory and Practical Example

Parameter relationship rationale objects store information about the relationship between the Related BPH Object Parameters and a BPH Object Parameter Value or a BPH Object Parameter Constraint as shown in Figure 5.9. These relationships form the basis for the generation of a PDN. Figure 5.24 shows an example of a PDN. Boxes to the left *affect* boxes to the right of a link. Boxes to the right are *affected by* boxes to the left of a link. For example, the `Mechanical Room - Fire Resistance Rating` is *affected by* the `Mechanical Room - Function`. Also, the `Mechanical Room - Fire Resistance Rating` *affects* the `Wall - Fire Resistance Rating`. Converting the first relationship example into general terms, the `Mechanical Room - Fire Resistance Rating` value of `2` is a BPH Object Parameter Value and the `Mechanical Room - Function` is a Related BPH Object Parameter. For the second relationship example, the BPH Object Parameter Value is the `Wall - Fire Resistance Rating` value of `2` and the Related BPH Object Parameter is the `Mechanical Room - Fire Resistance Rating`. Rationale Objects (Section 5.2.1) store the values of BPH Object Parameter Value, BPH Object Parameter Constraint, and Related BPH Object Parameters.

The PDN is a valuable tool for the design process. Designers often need to modify design object parameters as the design evolves. Sometimes, designers can not recall all the issues related to the object parameter they need to modify. Blindly modifying object parameters without understanding all the issues involved can lead to design conflicts, errors, and re-

design. A PDN can assist in this task by presenting to the designer any issues related to the relevant object parameter. The designer can then first evaluate the effects of the proposed modification before actually proceeding with the modification.

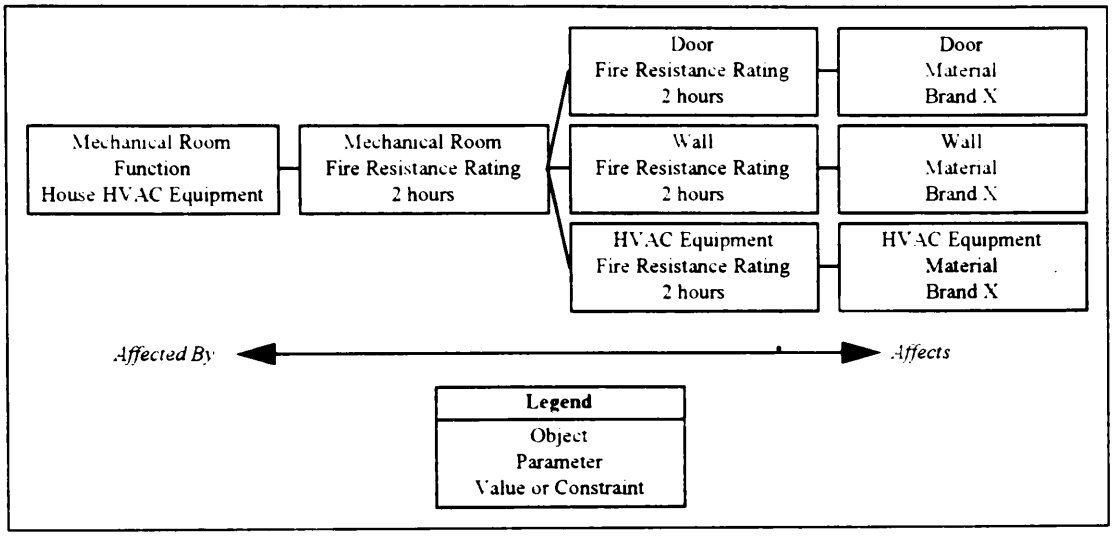


Figure 5.24 Example of a Parameter Dependency Network

The PDN is also a valuable tool for value engineers. Value engineering is the process of systematically formulating alternative designs that reduce the life cycle cost and/or increase the level of service provided by the original design. A PDN can support the VE task by giving value engineers a clear picture of the issues involved when modifying a particular object parameter.

5.4.2.2 DRIVE Interface

Figure 5.25 shows the DRIVE interface for displaying and navigating through the PDN. The text above the pull-down list box at the top center of the figure contains the current

object. The choices for this pull-down list box are the parameters of the current object. The selected parameter from this list is the focal object parameter. The **Affected by** and **Affects** list boxes at the middle portion of the figure respectively show a list of object parameters *affecting* and *affected by* the focal object parameter. For example, in Figure 5.25, the object parameter **Mechanical Room - General Function** *affects* the focal object parameter **Mechanical Room - Fire Resistance Rating**. As such, the **Affected by** object parameter list box contains **Mechanical Room - General Function**. This represents the fact that the **Mechanical Room - Fire Resistance Rating** is **Affected by** the **Mechanical Room - General Function**. A similar logic procedure generates the items in the **Affects** list box.

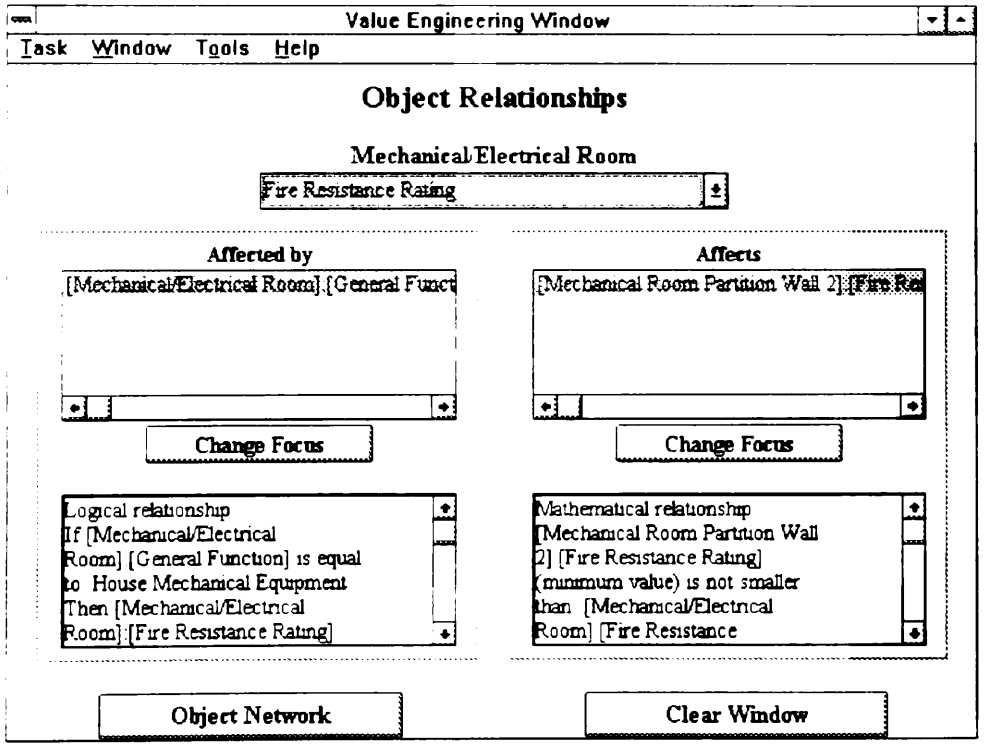


Figure 5.25 DRIVE Interface for Displaying and Navigating the PDN

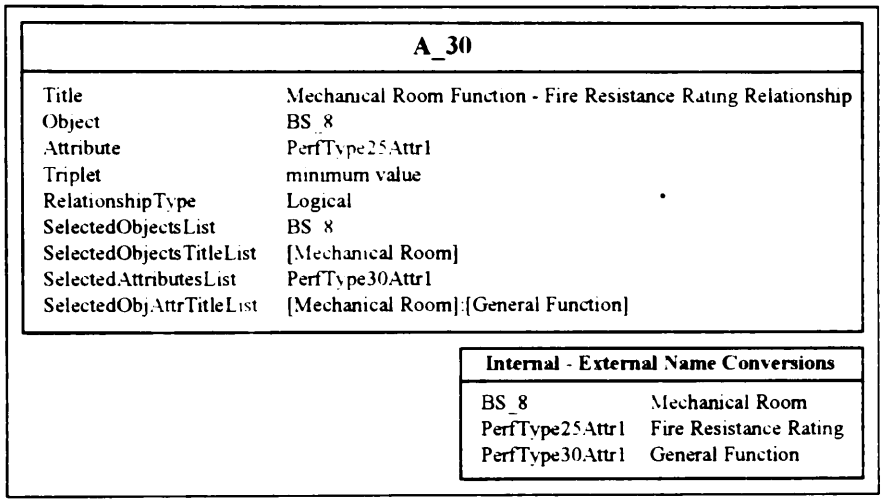
The two text boxes at the lower portion of Figure 5.25 contain the textual representation of the parameter relationship rationale existing between the highlighted object parameter in the list boxes and the focal object parameter. Highlighting a different object parameter in a list box changes the text inside the corresponding text box. Clicking either one of the `Change Focus` buttons changes the focal object parameter to the selected object parameter. This allows users to navigate through the PDN. Finally, the `Object Network` button on the bottom left of the figure allows the user to go to another object's PDN.

5.4.2.3 DRIVE Implementation

The module file `ve.kal` contains the interface objects necessary to present and navigate DRIVE's PDN interface. The module files `assertns.kal` and `descript.kal` contain the Assertion objects and the list of text description files used in generating the PDN and the textual representations of the parameter relationship rationale

Figure 5.26 displays the partial structure of an Assertions Object showing the slots relevant in constructing the PDN shown in Figure 5.25. The `Object`, `Attribute`, and `Triplet` slots correspond to the BPH Object Parameter Value or BPH Object Parameter Constraint items in Figure 5.9. For example, in the assertion `Mechanical Room Function Fire Resistance Rating Relationship (A_30)`, the `Object`, `Attribute`, and `Triplet` slots have the values, `Mechanical Room (BS_8)`, `Fire Resistance Rating (PerfType25-Attr1)` and `minimum value`, respectively. The `Mechanical Room - Fire Resistance Rating minimum value` is a BPH Object Parameter Constraint. The `SelectedObjects-`

List and SelectedAttributesList slots correspond to the Related BPH Object Parameter item in Figure 5.9. For example, in the same assertion, the SelectedObjects-List and SelectedAttributesList slots have the values Mechanical Room (BS_8), General Function (PerfType30Attr1), respectively. The Mechanical Room - General Function is a Related BPH Object Parameter.



A_30	
Title	Mechanical Room Function - Fire Resistance Rating Relationship
Object	BS_8
Attribute	PerfType25Attr1
Triplet	minimum value
RelationshipType	Logical
SelectedObjectsList	BS_8
SelectedObjectsTitleList	[Mechanical Room]
SelectedAttributesList	PerfType30Attr1
SelectedObjAttrTitleList	[Mechanical Room]:[General Function]

Internal - External Name Conversions	
BS_8	Mechanical Room
PerfType25Attr1	Fire Resistance Rating
PerfType30Attr1	General Function

Figure 5.26 Partial Structure (Parameter Dependency Network) of the DRIVE Implementation of a Rationale Object

Figure 5.25 shows DRIVE's particular interface implementation for displaying and navigating the PDN. This interface layout reflects the limitations of the prototyping system used in this dissertation. An interface similar to Figure 5.24 is a more powerful and more intuitive representation of the PDN as compared to DRIVE's PDN interface. DRIVE can only display two levels of relationships at a time. This limitation disappears with the use of an interface similar to Figure 5.24. It is unfortunate that the prototyping software have no means of easily generating an interface similar to Figure 5.24.

5.4.3 Data Verification and Conflict Resolution

Parameter relationship rationale information allows a computer system to perform analytical operations on the captured rationale. Examples of analytical operations are data verification and conflict resolution. Data verification means that the computer system can check new or existing data for inconsistencies with the stored rationale information. Conflict resolution provides a mechanism for the computer system to notify the use of an inconsistency and assist the user in resolving the conflict.

5.4.3.1 Theory

There are three areas where a computer can provide data verification assistance. These are value-constraint, value-relationship, and constraint-relationship verification. A BPH Object Parameter can have a BPH Object Parameter Value and zero or more BPH Object Parameter Constraints as shown in the lower right portion of Figure 4.1. Value-constraint verification compares the BPH Object Parameter Value with its BPH Object Parameter Constraints. A BPH Object Parameter Value must not be greater than the maximum value constraint. It must not be less than the minimum value constraint. It also must conform to its allowable values constraint. If a BPH Object Parameter Value violates one of these constraints, then the conflict resolution module takes over. The conflict resolution module notifies the user of the conflict. It assists the user in removing the conflict by changing the BPH Object Parameter Value, or by relaxing the offending BPH Object Parameter Constraint. Section 5.4.3.2.1 provides a practical example of value-constraint verification.

Value-relationship and constraint-relationship verification are similar to each other. As such, the following discussion on constraint-relationship is also applicable to value-relationship verification. Parameter Relationships relate a BPH Object Parameter Constraint with Related BPH Object Parameters as shown in Figure 5.1. Constraint-relationship verification involves comparing the BPH Object Parameter Constraint and the values of the Related BPH Object Parameters with the Parameter Relationship existing between them. Figure 5.27 shows a diagram of constraint-relationship verification using the Logical Relationship described in 5.2.2.1. Data verification involves comparing the minimum value constraint of `Mechanical Room - Fire Resistance Rating` (BPH Object Parameter Constraint) and the value of `Mechanical Room - General Function` (Related BPH Object Parameter) with the `Mechanical Room Function - Fire Resistance Rating Relationship` (Parameter Relationship). If the parameter values and constraints do not agree with the parameter relationship, then the conflict resolution module takes over. The conflict resolution module notifies the user of the conflict. It then assists the user in removing the conflict by changing the parameter values and/or constraints, by modifying the relationship, or by invalidating the relationship. Section 5.4.3.2.2 provides an example of constraint-relationship verification.

5.4.3.2 Practical Example

5.4.3.2.1 Value-Constraint Verification

The minimum value requirement for the object parameter `Ground Floor - Elevation` is `400 feet`. Setting the value of the same object parameter to 350 feet creates a conflict.

There are several ways for the system to acquire this value like user specification, result of a mathematical relationship calculation, and graphical database query from a CAD model. Section 5.4.3.3 describes DRIVE's data verification and conflict resolution interface.

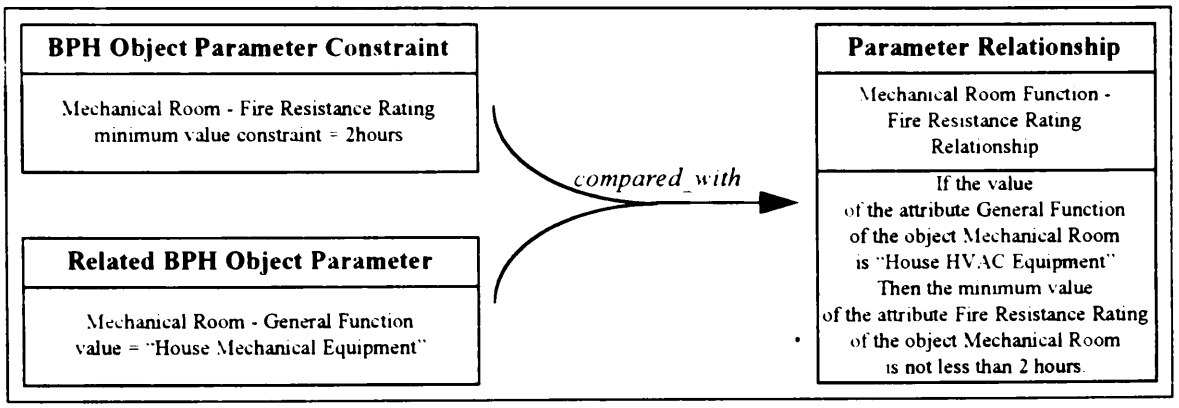


Figure 5.27 Constraint-Relationship Verification

5.4.3.2.2 Constraint-Relationship Verification

The minimum value constraint for the object parameter **Mechanical Room - Fire Resistance Rating** is **2 hours**. This minimum value constraint depends on the value of the object parameter **Mechanical Room - General Function**. Their relationship is as follows: If the **Mechanical Room - General Function** is equal to **"House HVAC Equipment"**, then the **Mechanical Room - Fire Resistance Rating minimum value** must be at least **2 hours**. Setting the minimum value of **Mechanical Room - Fire Resistance Rating** is **1.5 hours** triggers a conflict. Section 5.4.3.3 describe DRIVE's data verification and conflict resolution interface.

5.4.3.3 DRIVE Interface

DRIVE has two modes of data verification, namely, interactive and batch mode. Interactive data verification means that every time a design object parameter value or constraint changes, DRIVE checks it for inconsistencies with the entire database. Batch mode data verification means that the computer system checks its entire database for inconsistencies. If an inconsistency occurs, DRIVE's conflict resolution interface assists the user in resolving the conflict.

DRIVE displays Figure 5.28 after detecting the conflict described in Section 5.4.3.2.1. Figure 5.28 is representative of all value-constraint conflict resolution dialog boxes. These dialog boxes show the value and the constraints of a BPH Object Parameter. The user can then modify either the value or the constraint to resolve the conflict. If the user improperly modifies these items (for example, setting the value to only 375), the DRIVE system then checks if there are no more conflicts with the current values. With these improper values, DRIVE detects a conflict and once again displays the conflict resolution dialog box. Alternatively the user can also press the Process Conflict Later button. This action accepts the current inconsistent values. Using batch mode data verification, DRIVE rediscovers this inconsistency. This allows the user to resolve an inconsistency at a later time.

DRIVE displays Figure 5.29 after detecting the conflict described in Section 5.4.3.2.2. The value-relationship and conflict-relationship conflict resolution process involves a

series of dialog boxes. Figure 5.29 is an example of the first dialog box in the series. This dialog box allows the user to select one of four possible courses of action, namely, `Modify Attribute Values`, `Modify Relationship`, `Invalidate Relationship`, and `Process Conflict Later`. The `Process Conflict Later` button functions exactly the same as the same button in the value-constraint dialog boxes.

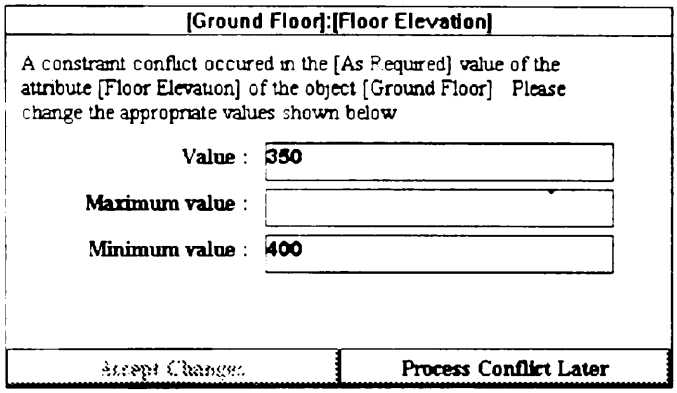


Figure 5.28 DRIVE Interface for Resolving Value-Constraint Conflicts

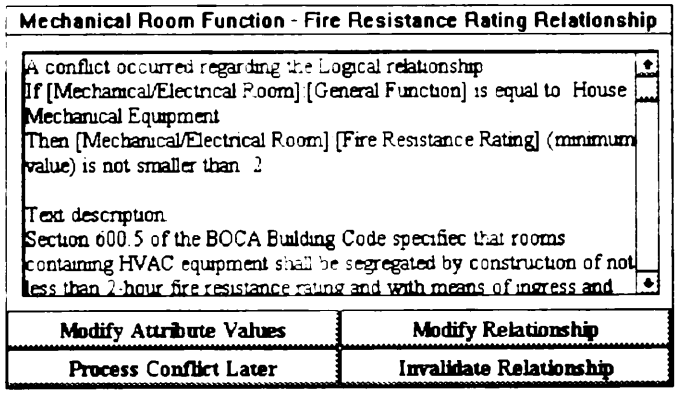


Figure 5.29 DRIVE Interface for Resolving Value-Relationship Conflicts (Resolution Method Selection)

Clicking the **Modify Attribute Values** button displays the dialog box shown in Figure 5.30. This dialog box asks the user which object parameter to modify. DRIVE builds the list of object parameters from the relationship definition. After selecting an object parameter to modify, the user can click the **Modify Attribute Values** button again. This brings up the dialog box shown in Figure 5.31. This dialog box allows the user to modify a specific object parameter to resolve the conflict.

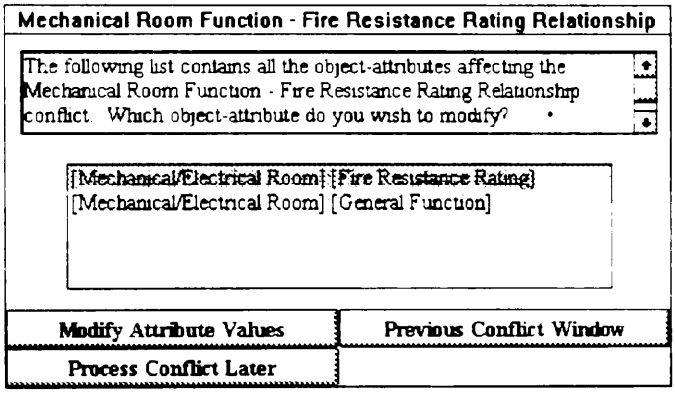


Figure 5.30 DRIVE Interface for Resolving Value-Relationship Conflicts (Object Parameter Selection)

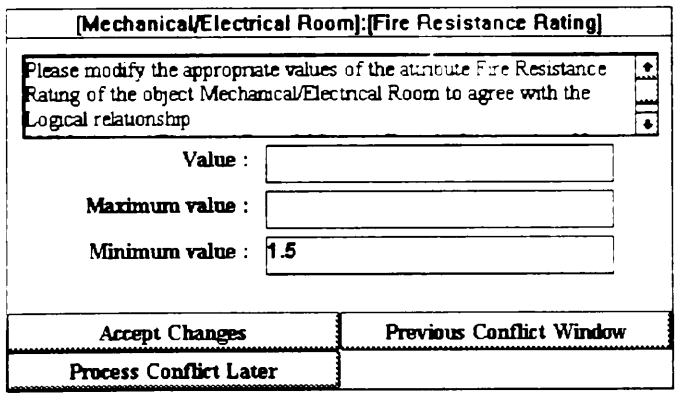


Figure 5.31 DRIVE Interface for Resolving Value-Relationship Conflicts (Object Parameter Modification)

Clicking the **Modify Relationship** button in Figure 5.29 displays the dialog box shown in Figure 5.32. This dialog box allows users to modify the relationship to resolve the conflict. Section 5.2.3 describes the functions and capabilities of the various interface objects in this dialog box.

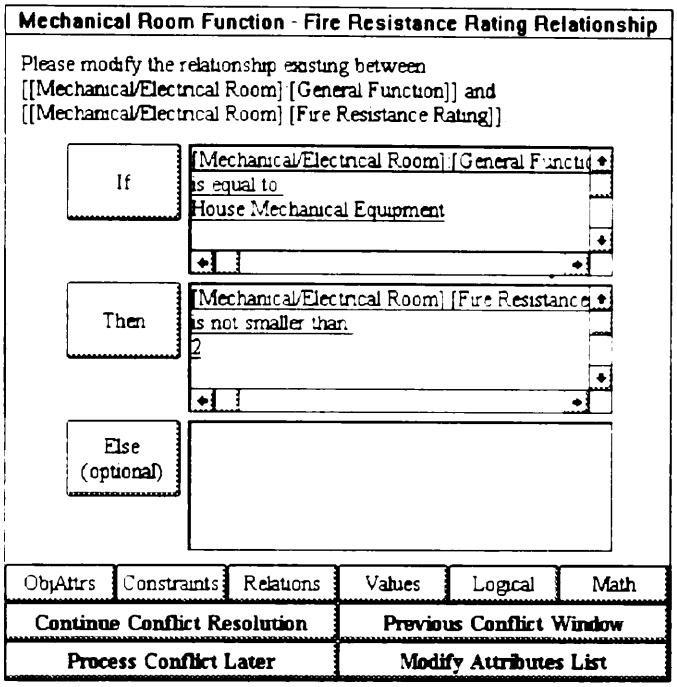


Figure 5.32 DRIVE Interface for Resolving Value-Relationship Conflicts (Relationship Modification)

Clicking the **Invalidate Relationship** button in Figure 5.29 displays a confirmation question asking the user if this is really what he or she wants to do. If the user confirms this action, DRIVE deletes the relationship, thereby resolving the conflict.

5.4.3.4 DRIVE Implementation

The module file `conflict.kal` generates the conflict resolution dialog boxes allowing the user to resolve design conflicts. The conflict resolution module also requires the assistance of the following modules `objattr.kal`, `rational.kal`, and `rule.kal`.

DRIVE manipulates a If-Then-Else rationale representation to an If-Then procedural rule. DRIVE collapses the entire If-Then-Else rationale representation into the If portion of the procedural rule. The Then portion of the procedural rule contains instructions to trigger the conflict resolution module. Thus, the If portion of the procedural rule must evaluate to true when there is a value-relationship conflict.

DRIVE uses the following transformation to convert an If-Then-Else rationale relationship to the If portion of the procedural rule. Let ARel, BRel, and CRel be the entire set of rationale relationship statements for the If, Then, and Else portions, respectively. In other words, the relationship rationale is If ARel Then BRel Else CRel. DRIVE converts this into If (ARel And Not BRel) Or (Not ARel And Not CRel) Then TriggerConflictResolution.

Figure 5.33 shows the Logical relationship described in Section 5.2.2.1. For this relationship, ARel corresponds to `[Mechanical Room]:[General Function] is equal to "House HVAC Equipment"`. BRel corresponds to `[Mechanical Room]:[Fire Resistance Rating](minimum value) is greater than 2`. CRel is empty (i.e., no

Else portion). Transforming this into procedural form: If `[Mechanical Room]:[General Function] is equal to "House HVAC Equipment"` And Not(`[Mechanical Room]:[Fire Resistance Rating](minimum value) is greater than 2`) Then TriggerConflictResolution.

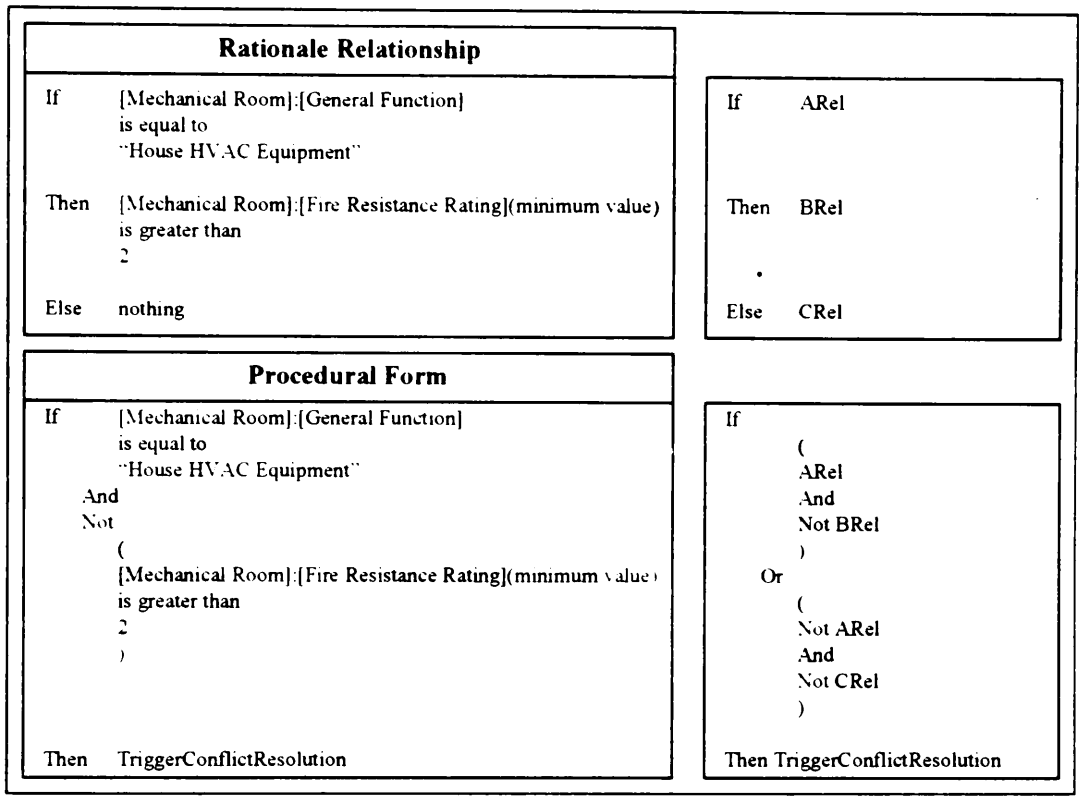


Figure 5.33 DRIVE Implementation of a Constraint-Relationship Conflict

Figure 5.34 graphically shows the conflict validation process. Validating the procedural form shown in Figure 5.33 with the values `"House HVAC Equipment"` and `1.5` for `General Function` and `Fire Resistance Rating`, respectively, gives a TRUE result as shown in Figure 5.34. As such, DRIVE triggers the conflict resolution module. These values are consistent with a relationship conflict.

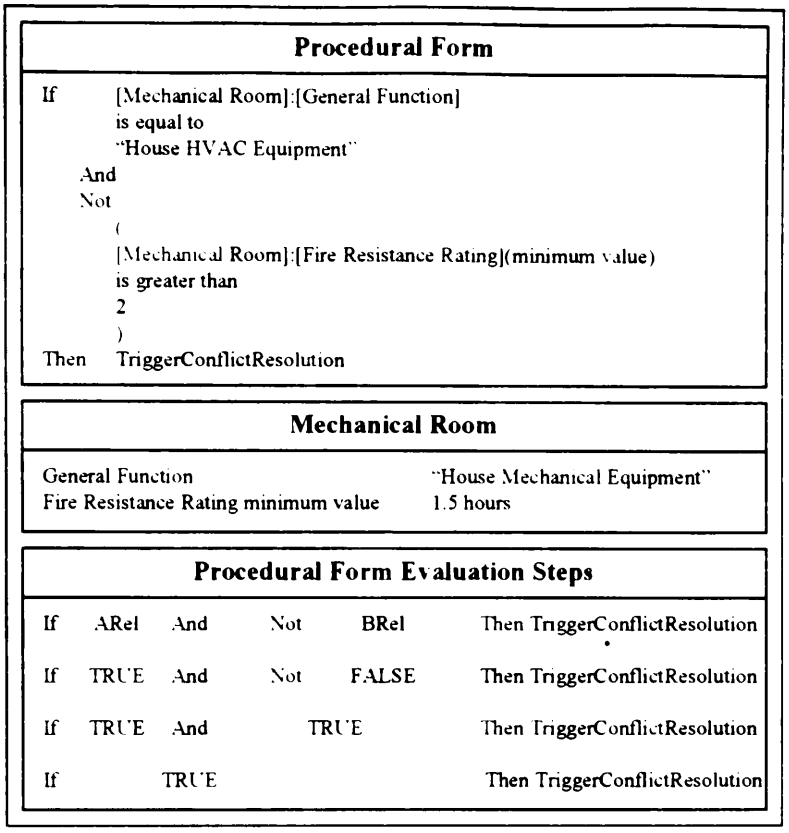


Figure 5.34 DRIVE Implementation of a Constraint-Relationship Conflict Validation

An interesting side effect of the DRIVE conflict resolution implementation is that it is able to detect malformed relationship rationale. A malformed relationship rationale is a set of parameter relationships that is impossible to satisfy. An example of a malformed relationship rationale is as follows

If A > 10 Then B > 20

If B > 20 Then C > 30

If C > 30 Then A < 10

DRIVE handles this cyclic and malformed relationship rationale quite well. A value-relationship conflict resolution dialog box keeps recurring until the user modifies or invalidates a relationship in this malformed relationship.

CHAPTER 6

VALUE ENGINEERING CAPABILITIES

The computer program by-product of this research assists in the capture, storage, retrieval, and processing of design rationale information. As stated in the research limitations (Section 1.3), the intended end users of this computer program are designers and value engineers. The Value Engineering Module provides support for various VE tasks. Providing support only to the VE Information Gathering phase is another limitation imposed on this research. Portions of Section 5.4 discuss the use of design rationale to support the VE Information Gathering phase. The purpose of this chapter is to present other specific VE Information Gathering phase tasks suitable for computerized support. This chapter has three main sections. Each section describes a different VE Information Gathering phase task. These sections also use the four-part breakdown structure used in the previous two chapters.

6.1 FAST DIAGRAMS

6.1.1 Theory and Practical Example

FAST diagrams depict the functional breakdown of a product. VE seeks to generate alternative designs that lower life cycle costs without sacrificing essential functions. Another method of VE is the creation of alternative designs that significantly increase the performance level of the essential functions without increasing life cycle costs. As such, FAST diagrams are important in VE. Section 2.5.4 discusses the theory behind FAST diagrams. It also provides a detailed practical example.

6.1.2 DRIVE Interface

Figure 6.1 shows how DRIVE displays a FAST diagram. DRIVE's FAST diagram interface reflects the limitations of the prototyping software. Value engineers draw FAST diagrams using the format shown in Figure 2.7. It is unfortunate that the prototyping software has no means of easily generating an interface similar to Figure 2.7.

6.1.3 DRIVE Implementation

The domain area used in this dissertation is building construction. As such, the procedure used in generating FAST diagrams is specific to building construction. Other design domains may or may not have a similar procedure for creating FAST diagrams.

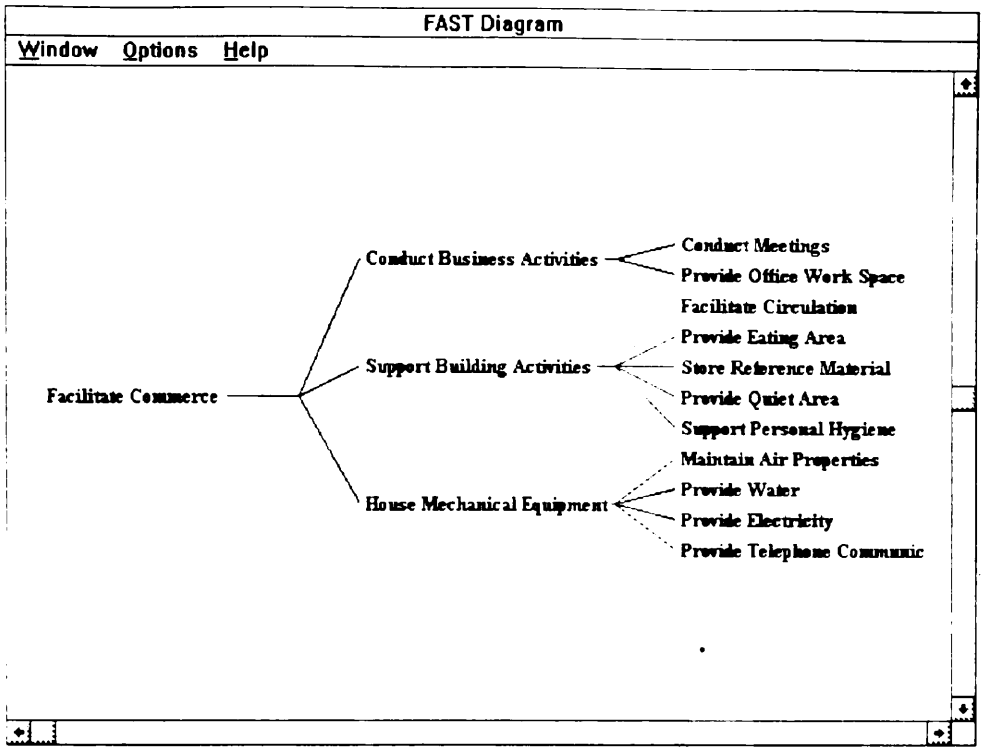


Figure 6.1 DRIVE Interface Displaying the FAST Diagram

There are several different methods of decomposing the functions of a building. DRIVE uses a decomposition structure based on spaces. A building has functions and spaces. A space fulfills one of the functions of the building. In DRIVE, buildings and spaces have two levels of functions, namely, `General Function` and `Particular Functions`. `General Function` is a single-valued attribute representing the broad or general function of an entity. `Particular Functions` is a multiple-valued attribute representing the specific or particular functions of an entity. For example, the `General Function` of a `Building` is `Facilitate Commerce` and its `Particular Functions` are `Conduct Business Activities`, `Support Building Activities`, and `House Mechanical Equipment`. DRIVE limits the `General Function` of `Spaces` to any of the `Particular`

Functions of the Building. Extending the previous example, Spaces can only have Conduct Business Activities, Support Building Activities, and House Mechanical Equipment as their General Function. This limitation ensures that a link exists from the General Function of the Building to the Particular Functions of the Spaces. DRIVE uses this link mechanism to generate a FAST diagram. Table 6.1 shows the General Function and the Particular Functions of the Building and its various Spaces. The information on this table generates the FAST diagram shown in Figure 6.1. DRIVE stores the FAST diagram in the module file fast.kal.

Table 6.1 Functions of the Example Building and Its Spaces

Design Object	General Function	Particular Function
Building	Facilitate Commerce	Conduct Business Activities Support Building Activities House Mechanical Equipment
Conference Room	Conduct Business Activities	Conduct Meetings
Main Corridor	Support Building Activities	Facilitate Circulation
Break Room	Support Building Activities	Provide Eating Area
Library Room	Support Building Activities	Store Reference Material Provide Quiet Area
Mechanical Room	House Mechanical Equipment	Maintain Air Properties Provide Water Provide Electricity Provide Telephone Communications
Open Office Space	Conduct Business Activities	Provide Office Work Space
Men's Toilet	Support Building Activities	Support Personal Hygiene
Women's Toilet	Support Building Activities	Support Personal Hygiene

6.2 COST BREAKDOWN DIAGRAMS

6.2.1 Theory

Cost breakdown diagrams are another set of diagrams that assist value engineers. They use these diagrams to determine items having the highest potential for cost savings. There are several cost breakdown models available for building systems. Cost breakdown models group related items together in a hierarchical manner. Table 4.1 shows three cost breakdown models. Estimating and controlling construction costs are the basis for these cost breakdown models. VE, however concentrates on life cycle costs. Although originally intended for construction costs, these cost breakdown models are also applicable when considering life cycle costs.

Life cycle costing (Dell'Isola and Kirk 1981) involves estimating all the costs incurred by a facility over its entire life cycle. These costs include owning, design, construction, operating, maintenance, renovation, and demolition costs. These costs occur at different periods over the entire life cycle of a facility. However, the relative value of money is not constant over time. This necessitates converting all cost items into a uniform unit of measure. The interest rate, i, and the economic life period, n, are factors to consider when converting cost items into a uniform unit of measure. A common measurement unit is Present Worth. Figure 6.2 shows the formulas for converting two common expenditure scenarios to Present Worth.

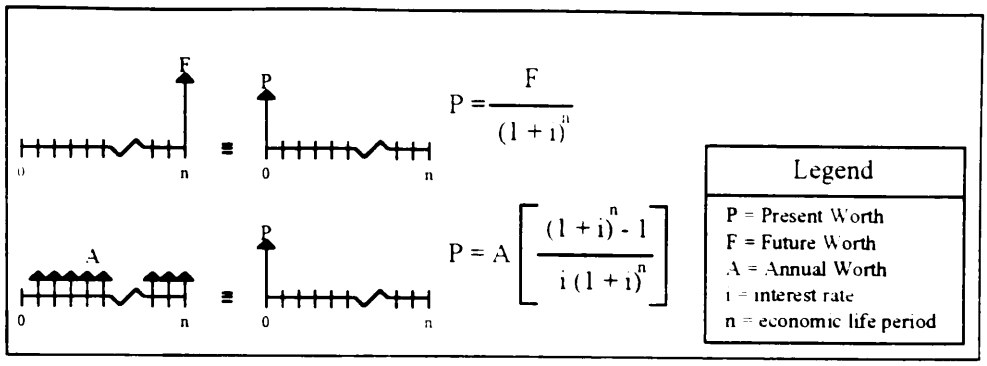


Figure 6.2 Present Worth Formulas

6.2.2 DRIVE Interface and Practical Example

DRIVE is currently capable of displaying two types of cost breakdown systems. Figure 6.3 shows an example of a space type cost breakdown. This diagram displays the relative proportions of the costs associated with the various space types. Figure 6.4 shows an example of a functional cost breakdown. The FAST diagram (Section 6.1) forms the basis for this breakdown structure. This breakdown system displays the relative proportions of the costs associated with the various functions of a building. Value engineers use these diagrams to direct their investigative efforts on the space types or functions having an unusual proportion of the building cost.

The Assemblies and Construction Trades Hierarchies (Figure 4.16) use the MCACES and Masterformat breakdown structures, respectively. As such, these two hierarchies can generate cost breakdown diagrams based on the MCACES or Masterformat systems. However, it is beyond the scope of this dissertation to provide this capability to the DRIVE proof-of-concept system.

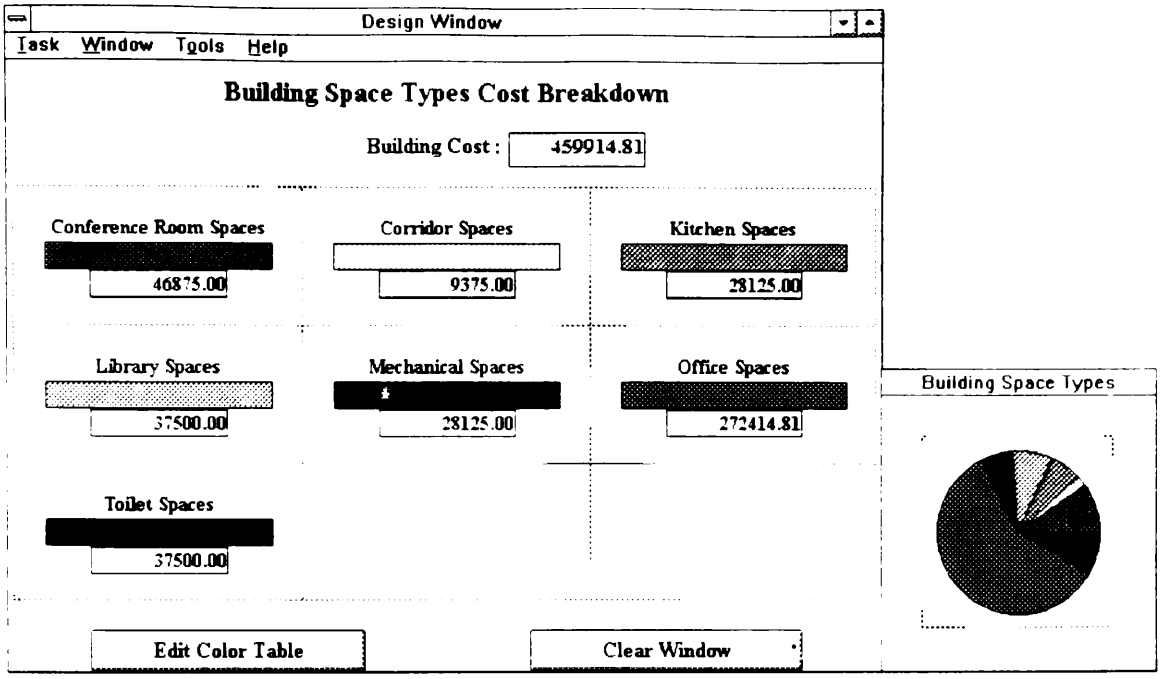


Figure 6.3 DRIVE Interface for Life Cycle Cost Diagrams (Spaces Breakdown)

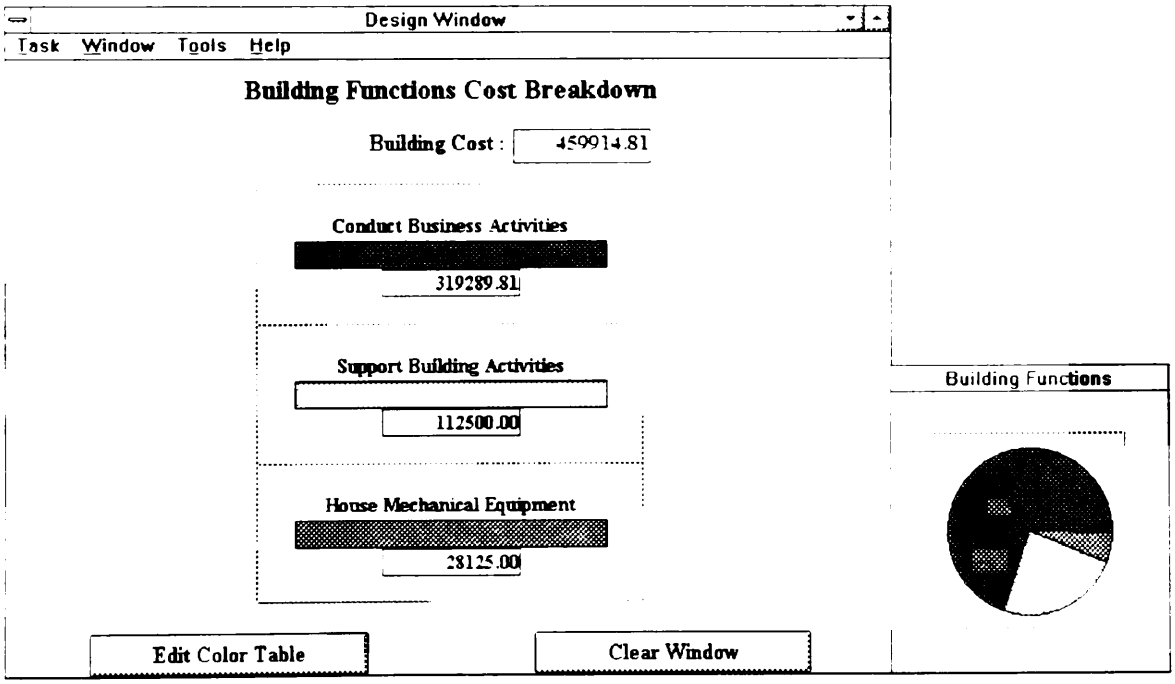


Figure 6.4 DRIVE Interface for Life Cycle Cost Diagrams (Functions Breakdown)

The DRIVE cost breakdown diagrams also act as navigation tools. Clicking on one of the pie slice wedges displays a menu allowing the user to go down deeper into the category corresponding to the pie slice wedge. For example, clicking on the wedge corresponding to `Toilet Spaces` in Figure 6.3 allows the user to display the cost breakdown of `Toilet Spaces`. The functional cost diagram shown in Figure 6.4 functions in the same manner. Clicking outside any of the pie slice wedges allows the user to navigate upwards.

The cost numbers shown in Figures 6.3 and 6.4 are total life cycle cost numbers. Table 6.2 shows in detail the information used in generating the cost breakdown diagram shown in Figure 6.3. Table 6.2 shows that all the spaces in the model have a $75 per square foot construction cost. DRIVE takes CAD data on the gross areas of the spaces and uses it to calculate the total construction costs shown on the third column of Table 6.2. The `Open Office Space` design object also has an annual maintenance cost of $1,000. Using the annual worth - present worth formula shown in Figure 6.2 and the user-specified values for interest rate and economic life period, DRIVE calculates a value of $9,915 for the total maintenance cost of the `Open Office Space` object. Figure 6.5 shows the DRIVE interface for defining life cycle costs. Users can not change the items enclosed in square brackets in Figure 6.5. These items correspond to items in the third column of Table 6.2. The system calculates these values based on other input parameters defined by the user.

Table 6.2 Life Cycle Cost Diagram Data

Design Object	User-Specified Values	System-Calculated Values
Building	interest rate i = 10% economic life period n = 50 years	$ 459,915 total life cycle cost $ 450,000 total construction cost $ 9,915 total maintenance cost
Conference Room	$ 75 / S.F. construction cost 625 S.F. gross area	$ 46,875 total construction cost
Main Corridor	$ 75 / S.F. construction cost 125 S.F. gross area	$ 9,375 total construction cost
Break Room	$ 75 / S.F. construction cost 375 S.F. gross area	$ 28,125 total construction cost
Library Room	$ 75 / S.F. construction cost 500 S.F. gross area	$ 37,500 total construction cost
Mechanical Room	$ 75 / S.F. construction cost 375 S.F. gross area	$ 28,125 total construction cost
Open Office Space	$ 75 / S.F. construction cost 3500 S.F. gross area $ 1000 annual maintenance cost	$ 262,500 total construction cost $ 9,915 total maintenance cost
Men's Toilet	$ 75 / S.F. construction cost 250 S.F. gross area	$ 18,750 total construction cost
Women's Toilet	$ 75 / S.F. construction cost 250 S.F. gross area	$ 18,750 total construction cost

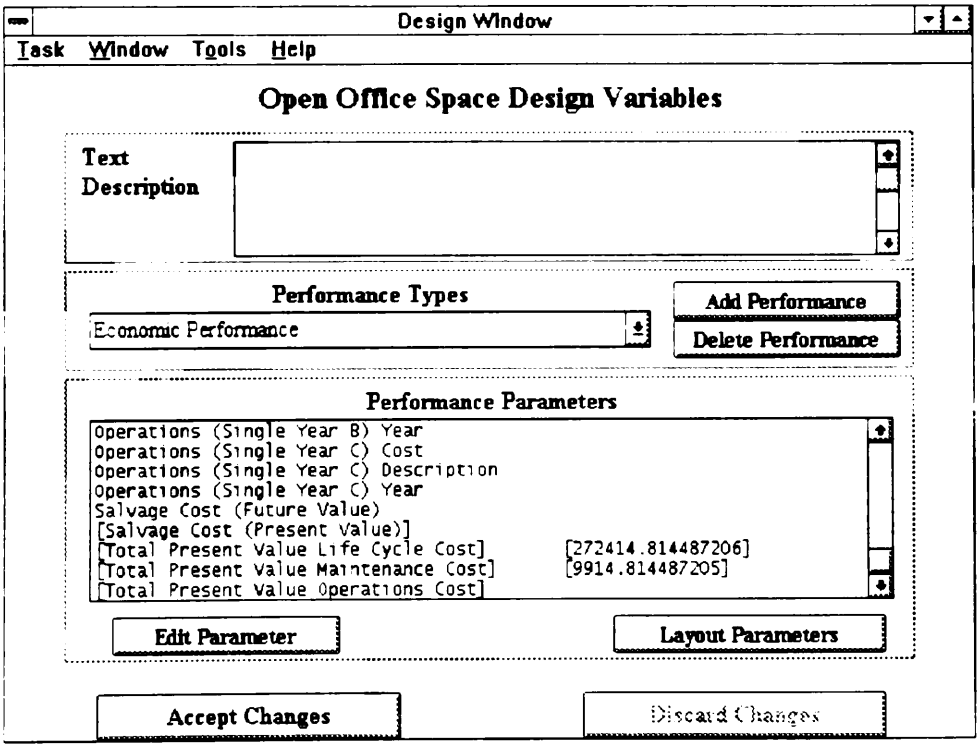


Figure 6.5 DRIVE Interface for Defining Life Cycle Costs

6.2.3 DRIVE Implementation

The module files `breakdwn.kal`, `brkdwnc.kal`, and `brkdwnd.kal` contain the interface objects necessary to generate, display, and interact with cost breakdown diagrams. The module files `brkdwnc.kal` and `brkdwnd.kal` contain the interface objects for the "as required" and "as designed" stages, respectively. It is necessary to have two separate sets of breakdown diagram interface objects because users sometimes need the capability of comparing an "as required" with an "as designed" life cycle cost estimate. The module file `breakdwn.kal` contains the interface object class definitions for the interface object instances in the `brkdwnc.kal` and `brkdwnd.kal` module files.

The Performances Library (Figure 4.3) contains `Economic Performance` objects for the `Building`, `Spaces Hierarchy`, and `Assemblies Hierarchy`. These `Economic Performance` objects contain the life cycle costing parameters. DRIVE currently provides for the following components of life cycle costs: design, construction, operating, maintenance, and salvage costs. Design, construction, and salvage costs are single year costs. Operating and maintenance costs each have two components, namely, single year and annual. Annual costs occur yearly for the entire economic life period of the building. DRIVE provides the capability of specifying up to three each of `Operating (Single Year) Costs`, `Operating (Annual) Costs`, `Maintenance (Single Year) Costs`, and `Maintenance (Annual) Costs`. Figure 6.6 displays the DRIVE Implementation of the `Open Office Space` object showing the life cycle costing parameters. For example, in Figure 6.6, one type of annual maintenance cost is `$1000` for `General Cleaning`.

BS_12	
Title	Open Office Space
PerfType23Attr1	262500
PerfType23Attr2	75
PerfType30Attr2	3500
PerfType23Attr29	1000
PerfType23Attr30	General Cleaning
PerfType23Attr41	9915
...	

Internal - External Name Conversion	
PerfType23Attr1	Construction Cost
PerfType23Attr2	Construction Cost Per Gross Area
PerfType30Attr2	Gross Area
PerfType23Attr29	Annual Maintenance Cost A
PerfType23Attr30	Annual Maintenance Cost A Title
PerfType23Attr41	Total Maintenance Cost

Figure 6.6 Partial Structure (Life Cycle Costs)
of the DRIVE Implementation of a Rationale Object

6.3 GRAPHICAL MODEL QUERIES

6.3.1 Theory and Practical Example

JSpace (Section 2.7.1), CIFECAD (Section 2.7.2), and ICADS (Section 2.7.3) supplement CAD databases with non-geometric information. Linking non-geometric information with CAD drawings enhances the expressive power of CAD drawings. Without non-geometric information, CAD drawings can only convey locations, distances, areas, and volumes. With non-geometric information, CAD drawings have a multitude of information such as material type, construction cost, manufacturer, color, etc. This greatly enhanced expressive power can easily overwhelm the user with irrelevant information. This necessitates the use of graphical model queries.

Graphical model queries are similar to query requests in relational databases. Both a graphical model and a relational database operate on the same principle. Their only difference lies in the fact that relational databases use records and fields while graphical models use object-entities and properties. Examples of graphical model queries include:

all first floor spaces having more than 25 occupants, all reinforced concrete walls thicker than 150 mm, and all windows manufactured by XYZ corporation. Color, shading, and hiding are some of the methods used to display results of graphical model queries.

The graphical display of query results is a valuable tool for value engineers. This capability allows value engineers to search for suitable items for value engineering. Some examples of queries value engineers might use include: objects having costs greater than a specified value, spaces having a unit area cost above the normal value, and objects having a parameter value less than a code requirement value.

6.3.2 DRIVE Interface

DRIVE currently has two available types of queries, namely, `Object Query`, and `Object-Attribute Query`. `Object Queries` allow users to highlight entire classes of design objects (for example, all `Partition Walls`) or a specific design object (for example, `Partition Wall 2`). `Object-Attribute Queries` allow users to highlight design objects satisfying certain criteria.

Figure 6.7 shows the DRIVE interface for formulating `Object-Attribute Queries`. The combination edit box - pull down list box on the upper right of Figure 6.7 allows users to type in a text identifier for the query, such as `Spaces with LCC > 100000`. It also allows users to retrieve previously created queries for possible modification. The text box image underneath this combination box contains the query statements. This text box, as well as

the four buttons underneath it function exactly the same way as their counterparts in the Logical and Mathematical Relationship Rationale Capture Windows (Section 5.2.3). These four buttons together with the pull-down list boxes at the left side of Figure 6.7 allow the user to formulate queries. An example of a query statement generated by the list boxes is `[Building Spaces]:[Total Present Value Life Cycle Cost]` The `Relations` and `Values` buttons generated the other two statements in the query definition text box shown in Figure 6.7. The English-like translation of this query is "Display all spaces having a total present value life cycle cost larger than $100,000". DRIVE manipulates the AutoCAD model to highlight all the objects satisfying this query. DRIVE is currently capable of simultaneously presenting the results of up to six different queries using six different colors. Figure 6.8 shows an example of the results of a graphical query

6.3.3 DRIVE Implementation

The module file `query.kal` contains all the query definitions created for a particular project. The Graphical Model Query module requires the assistance of the module file `rule.kal`. This module file assists by performing a syntax check on the structure of a query. If `rule.kal` detects an error, it asks the user to modify the current query definition.

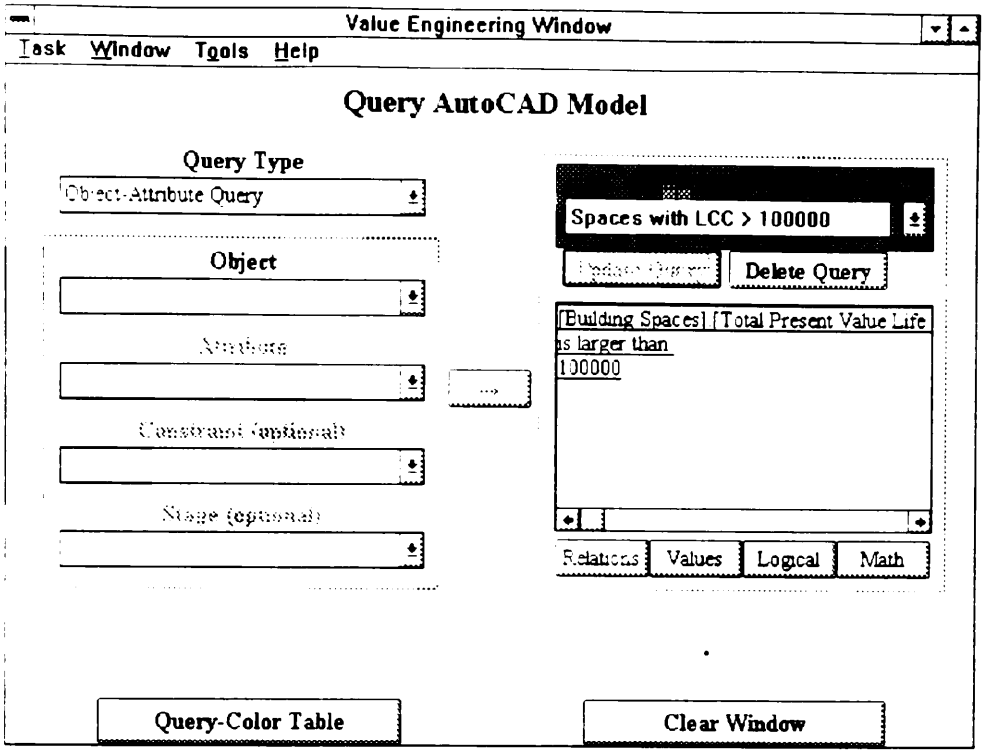


Figure 6 7 DRIVE Interface for Formulating Graphical Model Queries

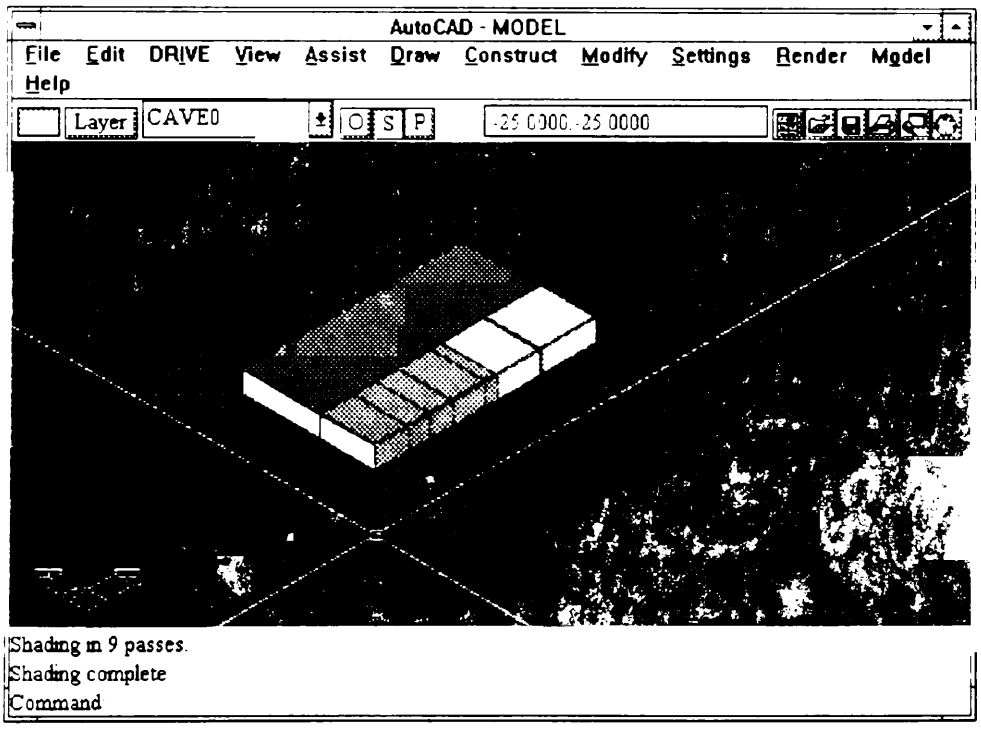


Figure 6.8 DRIVE Interface for Displaying Graphical Model Query Results

Figure 6.9 shows the DRIVE implementation of a graphical model query. DRIVE queries are also objects. Similar to all other objects in the DRIVE system, it has an internal-external naming scheme. For the query shown in Figure 6.7, its internal name is `Q_5` and its external name is the one entered by the user, `Spaces with LCC > 100000`. The `QueryLines`, `DisplayText`, and `RuleText` attributes function similarly to the `RelationshipLines`, `DisplayText`, and `RuleText`, attributes in the Relationship Definition Dialog Boxes (Section 5.2.4). Thus, the discussion on the determination of values for these attributes is similar to the one in Section 5.2.4.

Q_5	
Title	Spaces with LCC > 100000
QueryLines	3
DisplayText1	[Building Spaces]:[Construction Cost]
DisplayText2	is greater than
DisplayText3	100000
RuleText1	BuildingSpaces:PerfType23.Attr1
RuleText2	
RuleText3	100000

Figure 6.9 DRIVE Implementation of a Graphical Model Query

CHAPTER 7

SUMMARY AND RECOMMENDATIONS

7.1 RESEARCH SUMMARY

Facilities produced by the AEC industry typically go through several distinct phases in their life span. There are different participants in each of these phases. These participants generate and use a lot of information about a facility specific to their particular life cycle phase. These participants give to the next stage's participants only a summarized version of the information they generated and used. This results in a lot of information being implicitly conveyed by these summary documents. This implicit representation often leads to misinterpretations and unnecessary expenditures. One example of these summary documents is design drawings. These drawings contain a lot of explicit information about a design product. They also contain a lot of implicit information about the design process. This implicit information is useful to all participants in the various life cycle phases. This dissertation uses the term design rationale to refer to this implicit information about the design process.

The AEC industry has already acknowledged the importance of the implicit information or design rationale embedded in design documents. Value engineering is one proof of this acknowledgment. When applied during the design stage, a value engineering study brings together representatives from the various design fields, owners, construction personnel, facility operators, and end users. The participants formulate design alternatives that either decrease life cycle costs, increase facility level of service, or both. During the discussion of a specific alternative, participants orally state the rationale behind their design drawings. This provides the opportunity to make design trade-offs with other designers. It is unfortunate that the rationale already explicitly stated during a value engineering study remains "trapped" in the design phase and does not flow through to other life cycle phases, such as construction.

The main issue addressed by this dissertation is the need to communicate design rationale information embedded inside current information exchange mechanisms, such as design drawings. Several researchers are working on the subject area of design rationale, especially in the civil engineering and computer science fields. As such, the first step in this research effort was to conduct a literature review on design rationale. This step allowed the author to select a particular aspect of design rationale research that contributes to the existing body of knowledge.

The primary objective of this research effort was to create a data structure allowing designers to explicitly express their design rationale for later retrieval by others. An additional requirement on this data structure was that it must allow a computer system to perform analysis on the design rationale data. The large scope of this primary objective required the selection of specific target areas — building construction as the design domain and value engineering as the application area. These limitations further expanded the literature review to include architectural programming, building modeling, computer-aided design, object-oriented programming, and value engineering. The literature review on value engineering was supplemented by field studies with OVEST, the value engineering unit of the U. S. Army Corps of Engineers.

The concepts investigated in the literature review and the issues identified during the field studies formed the foundation of the data structure developed in this research. Two separate modules, a domain-dependent Knowledge Representation Module, and a domain-independent Rationale Storage Module comprise this data structure. Chapters 4 and 5 discussed these two modules in detail. This dissertation proposed the separation of product information, represented by the KRM, and process information, represented by the RSM. This dissertation developed a data structure for both these two modules allowing them to work together.

The research scope was still broad even after limiting the design domain and application areas. Further limitations imposed on the research included: (1) supporting rationale only for the design stage, (2) developing a building model with only sufficient entities to illustrate the concept of using the data structure to represent design rationale, and (3) supporting the Information Gathering phase of value engineering. The reasons for these limitations were: (1) rationale occurs not only on the design stage, but on all life cycle stages, (2) the creation of a robust building model capable of representing all possible design entities is a very huge undertaking, and (3) there are several phases of value engineering and each of these phases have different uses of design rationale.

Two secondary objectives of this research were to define a computer program architecture and to create a computer program interface proving the possibility of implementing the data structure developed in this research. An additional requirement on the computer program architecture and consequently, on the data structure, was that it must have easy extensibility. This allows other researchers to address the limitations of this dissertation. The development of the computer program by-product of this research also involved the selection of the computer platform and prototyping software. Section 3.3 provided an overview of the computer program architecture. The DRIVE Interface and DRIVE Implementation sections of Chapters 4, 5, and 6 detailed the computer program architecture and interface.

Another secondary objective of this research was to verify the appropriateness of the data structure in representing design rationale. Two industry validation studies accomplished this objective. In both these studies, the author gathered design rationale information from the project architects. The author also entered these information into the computer program by-product of this research. Section 3.4 and the Practical Example sections of Chapters 4, 5, and 6 showed that the data structure can indeed represent design rationale.

7.2 CONTRIBUTIONS TO THE BODY OF KNOWLEDGE

The primary and most significant contribution to the body of knowledge by this research effort is the data structures allowing designers to express their design rationale for later retrieval by others. The difference between the data structures developed in this dissertation and those done by others is its generic and extensible nature. This dissertation concentrated on the building construction domain. However, the data structures developed in this dissertation can be used for other domains, such as highway construction. The BNF grammar representation of the data structures allows other researchers to build on the knowledge gained from this research effort. The advantage of representing the data structures in a BNF grammar format is that it is generic. Program developers can use this grammar representation to create their own computer implementations in other systems different from the one used in this dissertation.

The Parameter Dependency Network (PDN) representation of design rationale is another significant contribution to the body of knowledge. Design rationale is a very fertile field of research in both the civil engineering and computer science domains. The literature review on design rationale (Section 2.3) showed some of the various representation mechanisms other researchers developed to represent design rationale information. The PDN is another representation scheme.

The current application areas of design rationale are all in design — HVAC design, kitchen design, and elevator design. The identification of value engineering (VE) as an excellent application area for design rationale is another contribution to the body of knowledge. VE is a logical choice as an application area for design rationale. The Information Gathering phase of VE is nothing more than explicitly stating vocally the rationale behind the design decisions. The computerized representation techniques developed in this dissertation helps support the VE Information Gathering phase by providing a permanent storage mechanism for the vocally-stated design rationale.

Focusing only on the design stage is one of the limitations imposed on this dissertation. However, design is only one stage in the life cycle of a facility. The facility life cycle also includes the concept definition, construction, operation, maintenance, and demolition stages. The data structures developed in this dissertation allow for the capture of rationale for entire life cycle, not only for the design stage. Thus, the identification of the extension

of the design rationale concept to life cycle rationale is another significant contribution of this research to the existing body of knowledge.

Architectural programming is the term used to refer to the concept definition stage for building systems. The literature review on architectural programming (Section 2.4) noted that the rationale concept is also applicable for the concept definition stage. By providing the capability of capturing rationale for the design stage, the designers may rekindle the fading practice of architectural programming. Designers are under pressure to produce the design drawings quickly and cheaply. They sometimes forgo the formal architectural programming task and distill it inside the design development task. Thus, the process of architectural programming becomes part of the design rationale implicitly represented in current design drawings and specifications. The ability of design rationale to stimulate interest in architectural programming is another contribution to the body of knowledge.

The internal-external name mapping used extensively in the DRIVE implementation is a minor contribution to the body of knowledge in the field of applied object-oriented programming. With regard to object names, the 32 character limit of the prototyping software was effectively replaced with a 200 character capacity.

Another minor contribution of the DRIVE implementation occurs in the field of computer technology. The DRIVE implementation is another proof of the possibility of integrating

two different computer program applications. DRIVE used Kappa-PC and AutoCAD to assist the designer in capturing, storing, retrieving, and processing design rationale.

7.3 RECOMMENDATIONS FOR FUTURE WORK

Very few design firms, if any, extensively document their design process. Documenting the design process is synonymous to capturing design rationale. As such, there is no definite proof that capturing design rationale reduces life cycle costs as shown in Figure 1.1. One future research work is conducting case studies to verify the speculated life cycle cost savings associated with capturing design rationale.

Design firms are reluctant to capture design rationale because current design rationale tools, including DRIVE, are very cumbersome to use and interferes with the designers' normal work process. A second future research area is the investigation of human-computer interaction issues regarding the AEC design process and design rationale capture. This dissertation used a retroactive rationale capture approach. The author gathered design rationale from the project architects of two completed design projects. The author attached these captured design rationale to the already completed design drawings. Real-time rationale capture involves engaging design firms to work with a design rationale capture tool to document their design process as it progresses.

Building design involves designers from several different fields. Usually, these designers are in different firms spread geographically across the country, or even the world. Recent

advances in computer communications technology allow designers to exchange information quickly and cheaply. A third research topic is the development of a communication infrastructure system to support concurrent, collaborative design and VE.

Conducting technology transfer activities is a fourth future work task. The proof-of-concept computer program developed in this dissertation is a purely academic tool. Presenting this academic tool to interested computer program developers, design firms, and construction firms gives an opportunity for the industry to convert this academic tool to a useful commercial product.

Finally, a fifth possible future work item involves addressing the various limitations imposed on this dissertation. Sub-tasks in this work item include: (1) developing additional Knowledge Representation Modules, such as for highway construction; (2) expanding the Rationale Storage Module to handle rationale for other life cycle stages, such as construction rationale; (3) developing computer program capabilities to support tasks other than the value engineering information gathering phase, such as the other value engineering phases, estimating, and scheduling; (4) modifying the computer program to support the capture of rationale information at the domain level — currently, DRIVE can only capture rationale at the project-specific level; and (5) enhancing the building modeling capability by creating additional entities.

REFERENCES

AutoCAD Release 12 Reference Manual (1992). AutoDesk, Sausalito, California.

Bytheway, C. W. (1992). "FAST - An Intuitive Thinking Technique." *Proceedings of the 1992 International Conference of the Society of American Value Engineers*, Phoenix, Arizona, 229-232.

Chandrasekaran, B., Goel, A. K., and Iwasaki, Y. (1993). "Functional Representation as Design Rationale." *Computer*, 26(1), 48-56.

Choi, K. C. and Ibbs, C. W. (1990). "CAD/CAE in Construction: Trends, Problems, and Needs." *Journal of Management in Engineering*, ASCE, 6(4), 394-415.

Code of Federal Regulations (1991). U.S. Government Printing Office, Washington, D.C.

The Construction Management Committee of the ASCE Construction Division (1991). "Constructability and Constructability Programs: White Paper." *Journal of Construction Engineering and Management*, ASCE, 117(1), 67-89.

Coyne, R. D., Roseman, M. A., Radford, A. D., Balachandran, M., and Gero, J. S. (1990). *Knowledge-Based Design Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts.

de la Garza, J. M., Alcantara, P. T., Kapoor, M., and Ramesh, P. S. (1994). "Value of Concurrent Engineering for A/E/C Industry." *Journal of Management in Engineering*, ASCE, 10(3), 46-55.

de la Garza, J. M. and Oralkan, G. A. (1995). "Using Design Intent for Interpreting Brand Name or Equal Specifications." *Journal of Computing in Civil Engineering*, ASCE, 9(1), 43-56.

de la Garza, J. M. and Oralkan, G. A. (1992). "An Object Space Framework for Design/Construction Integration." *Building and Environment: International Journal of Building Science and Its Applications*, Pergamon Press, 27(2), 243-255.

de la Garza, J. M. and Ramakrishnan, S. (1995). "A Tool for Designers to Record Design Rationale of a Constructed Project." *Proceedings of the 10th International Conference On Applications of Artificial Intelligence in Engineering*, Udine, Italy, 533-540.

Dell'Isola, A. J. (1982). *Value Engineering in the Construction Industry*, Van Nostrand Reinhold Company, New York, New York.

Dell'Isola, A. J. and Kirk, S. J. (1981). Life Cycle Costing for Design Professionals, McGraw-Hill Incorporated, New York, New York.

Dym, C. L. and Levitt, R. E. (1991). *Knowledge-Based Systems in Engineering*, McGraw-Hill Incorporated, New York, New York.

East, E. W., Roessler, T., and Lustig, M. (1995). "Improving the Design-Review Process: The Reviewer's Assistant." *Journal of Computing in Civil Engineering*, ASCE, 9(4), 229-235.

Evans, B. H. and Wheeler, C. H. (1969). *Architectural Programming*, American Institute of Architects, Washington, D. C.

Favela, J., Wong, A., and Chakravarthy, A. (1993). "Supporting Collaborative Engineering Design." *Engineering with Computers*, Springer-Verlag London Limited, 9(3), 125-132.

Fikes, R. and Kehler, T. (1985). "The Role of Frame-Based Representation in Reasoning." *Communications of the ACM*, Association for Computing Machinery, 28(9), 904-920.

Fisher, D. J., and O'Connor, J. T. (1991). "Constructability for Piping Automation: Field Operations." *Journal of Construction Engineering and Management*, ASCE, 117(3), 468-485.

Fischer, G., Lemke, A. C., and McCall, R. (1991). "Making Argumentation Serve Design." *Journal of Human-Computer Interaction*, Lawrence Erlbaum Associates Incorporated, 5(6), 393-419.

Ganeshan, R., Finger, S., and Garrett, J. (1991). "Representing and Reasoning with Design Intent." *Proceedings of the 1st International Conference on Artificial Intelligence in Design*, Edinburgh, United Kingdom, 737-755.

Garcia, A. C. B. and de Souza, C. S. (1995). "Rhetorical Structures in the Delivery of Design Intent." *Proceedings of the 2nd ASCE Congress on Computing in Civil Engineering*, Atlanta, Georgia, ASCE, 1443-1452.

Garcia, A. C. B., Howard, H. C., and Stefik, M. J. (1994). "Improving Design and Documentation by Using Partially Automated Synthesis." *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, Cambridge University Press, 8(4), 335-354.

Gero, J. S. (1990). "Design Prototypes: a Knowledge Representation Schema for Design." *AI Magazine*, 11(4), 26-36.

Gibbs, R. E. (1989). "Value Engineering Expert System." *Value World*, 12(2), 13-14.

Gielingh, W. (1988). "General AEC Reference Model." *ISO TC184 SC4 WG1 doc. 3.2.2.1*, Delft, Netherlands.

Hix, D. and Hartson, H. R. (1993). *Developing User Interfaces*, John Wiley and Sons, Incorporated, New York, New York.

Howard, H. C. (1991). "Project-Specific Knowledge Bases in AEC Industry." *Journal of Computing in Civil Engineering*, ASCE, 5(1), 25-41.

Howard, H. C., Levitt, R. E., Paulson, B. C., Pohl, J. G., and Tatum, C. B. (1989). "Computer Integration: Reducing Fragmentation in AEC Industry." *Journal of Computing in Civil Engineering*, ASCE, 3(1), 18-32.

Ito, K., Ueno, Y., Levitt, R. E., and Darwiche, A. (1989). "Linking Knowledge-Based Systems to CAD Design Data with an Object-Oriented Building Product Model." *Technical Report Number 17*, Center for Integrated Facility Engineering, Stanford University, Stanford, California.

Jacobus Technology (1996a). "JSpace Class Editor." *http: www.jacobus.com products jspce01.htm*.

Jacobus Technology (1996b). "JSpace Object Engine." *http: www.jacobus.com products jspoe01.htm*.

Jacobus Technology (1996c). "JSpace Product Overview." *http: www.jacobus.com products jspov01.htm*.

Jacobus Technology (1996d). "JSpace Viewer." *http: www.jacobus.com products jspv01.htm*.

Jacobus Technology (1996e). "Technology: Integration Objects." *Jacobus Technology Perspective*, 6(1).

Jacobus Technology (1995). "Technology: Intelligent Schematics: A New Approach." *Jacobus Technology Perspective*, 5(1).

Jacobus Technology (1994). "Technology: Heterogenous Data." *Jacobus Technology Perspective*, 4(2).

Kappa-PC Reference Manual Version 2.3. (1994). Intellicorp, Mountain View, California.

Kirk, S. J. (1989). "Post-occupancy Value Engineering." *Ekistics*, 336, 141-146.

Klein, M. (1993). "Capturing Design Rationale in Concurrent Engineering Teams." *Computer*, 26(1), 39-47.

Lee J. and Lai K-Y. (1991). "What's in Design Rationale?" *Journal of Human-Computer Interaction*, Lawrence Erlbaum Associates Incorporated, 5(6), 251-280.

Marcotty, M., and Ledgard, H. F. (1986). *Programming Language Landscape*, Science Research Associates Incorporated, Chicago, Illinois.

Marcus, S. and McDermott, J. (1989). "SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems." *Artificial Intelligence*, Elsevier Science Publishers, 39(1), 1-37.

Mostow, J. (1985). "Toward Better Models of the Design Process." *AI Magazine*, 6(1), 44-57.

O'Connor, J. T., Hugo, F., and Stamm, E. M. (1991) "Improving Highway Specifications for Constructability." *Journal of Construction Engineering and Management*, ASCE, 117(2), 242-258.

OVEST (1995). "Value Engineering Study for McAlpine Support Facilities." *Report CEORL-VE-95-06*, Savannah, Georgia.

Peña, W. (1977). *Problem Seeking*, CBI Publishing Company, Boston, Massachusetts.

Peña-Mora, F., Sriram, D., and Logcher, R. (1995). "Design Rationale for Computer-Supported Conflict Mitigation." *Journal of Computing in Civil Engineering*, ASCE, 9(1), 57-72.

Pohl, J., Myers, L., Chapman, A., and Cotton J. (1989). "ICADS: Working Model Version 1." *Design Institute Research Report: CADRU-03-89*, CAD Research Unit, California Polytechnic State University, San Luis Obispo, California.

Reinschmidt, K. F., Griffis, F. H., and Bronner, P. L. (1991). "Integration of Engineering, Design, and Construction." *Journal of Construction Engineering and Management*, ASCE, 117(4), 756-772.

Russell, J. S., Swiggum, K. E., Shapiro, J. M., and Alaydrus, A. F. (1994). "Constructability Related to TQM, Value Engineering, and Cost/Benefits." *Journal of Performance of Constructed Facilities*, ASCE, 8(1), 31-45.

Schildt, H. (1991). *C++: The Complete Reference* McGraw-Hill Incorporated, Berkeley, California.

Shen, Q. and Brandon, P. S. (1991). "Can Expert Systems Improve VM Implementation?" *Proceedings of the 1991 International Conference of the Society of American Value Engineers*, Kansas City, Missouri, 168-176.

Stefik, M. and Bobrow, D. G. (1985). "Object-Oriented Programming: Themes and Variations." *AI Magazine*, 6(4), 40-62.

Sturges, R. H. (1992). "Towards Computer Aided Value Engineering." *Proceedings of the 1992 International Conference of the Society of American Value Engineers*, Phoenix, Arizona, 282-290.

Taher, K. A. and Diekmann, J. E. (1992). "A Knowledge Based System for Value Engineering." *Proceedings of the 1992 International Conference of the Society of American Value Engineers*, Phoenix, Arizona, 51-56.

Turner, J. A. (1990). "AEC Building Systems Model." *ISO TC184 SC4 WG1 doc. 3.2.2.4*, Delft, Netherlands.

Ullman, D. G. (1994). "Issues Critical to the Development of Design History, Design Rationale and Design Intent Systems." *Design Theory and Methodology*, 68, 249-258.

Ullman, D. G., Dietterich, T. G., and Stauffer, L. (1988). "A Model of the Mechanical Design Process Based on Empirical Data." *Artificial Intelligence for Engineering Design, Analysis and Manufacture*, Cambridge University Press, 2(1), 33-52.

Value Engineering (1991). Department of the Army, Office of the Chief of Engineers, Washington, D.C.

ValuLink (1995). http://www.libra.wcupa.edu/ValuLink.

Wade, J. W. (1992) "Essential Typologies for the Construction Industry." *Building and Environment: International Journal of Building Science and Its Applications*, Pergamon Press, 27(2), 117-124.

White, E. T. (1972). *Introduction to Architectural Programming*, Architectural Media, Tucson, Arizona.

Zimmerman, L. W. and Hart, G. D. (1982). *Value Engineering: A Practical Approach for Owners, Designers, and Contractors*, Van Nostrand Reinhold Company, New York, New York.

APPENDIX A

BNF GRAMMAR

---

1. <rationale_object> ::==
   (<design_decision>
   <has_identifier>
   <has_dependency_type>
   <depends_on>
   <has_relationship_type>
   <has_relationship>
   <has_description>)

2. <design_decision> ::== <object_parameter_value> | <object_parameter_constraint>

3. <object_parameter_value> ::==
   (<object_x>
   <parameter_y_of_object_x>
   <value_type_of_parameter_y_of_object_x>
   <old_value_of_parameter_y_of_object_x>
   <new_value_of_parameter_y_of_object_x>)

4. <object_parameter_constraint> ::==
   (<object_x>
   <parameter_y_of_object_x>
   <constraint_type_of_parameter_y_of_object_x>
   <old_constraint_of_parameter_y_of_object_x>
   <new_constraint_of_parameter_y_of_object_x>)

5. <object_x> ::== (**Object** <object>)

6. <parameter_y_of_object_x> ::== (**Parameter** <object_x_parameter>)

7. <value_type_of_parameter_y_of_object_x> ::= (**Value_Type "value"**)

8. <old_value_of_parameter_y_of_object_x> ::== (**Old_Value** <value>)

9. <new_value_of_parameter_y_of_object_x> ::== (**New_Value** <value>)

10. <constraint_type_of_parameter_y_of_object_x> ::=
    (**constraint_type "minimum value"** | **"maximum value"** | **"allowable values"**)

11. <old_constraint_of_parameter_y_of_object_x> ::== (**Old_Constraint** <value>)

12. <new_constraint_of_parameter_y_of_object_x> ::== (**New_Constraint** <value>)

13. <object> ::== *a sequence of* <character>*s referring to any DRIVE object*

14. <object_x_parameter> ::==
    a sequence of <character>*s referring to any parameter of* <object_x>

15. <has_identifier> ::== (**Rationale_Name** <string>)

16. <has_dependency_type> ::==
    (**Dependency_Type**
      "owner requirement" | **"code requirement"** | **"other object parameters"**)

17. <depends_on> ::==
    (**Dependent_Object_Parameters_List**
      ((<object_x> <parameter_y_of_object_x>+)+))

18. <has_relationship_type> ::==
    (**Relationship_Type "logical relationship"** | **"mathematical relationship"**)

19. <has_relationship> ::==
    (**Relationship** <logical_relationship> | <mathematical relationship>)

20. <logical_relationship> ::==
    ((**Logical_If_Portion** <logical_comparative_declaration>)
       (**Logical_Then_Portion** <logical_comparative_declaration>)
       { (**Logical_Else_Portion** <logical_comparative_declaration>) })

21. <mathematical_relationship> ::==
    ((**Object_Parameter_Pair** <object_x> <parameter_y_of_object_x>)
       (**Mathematical_Comparison_Operator** < | = | > | ≤ | ≠ | ≥)
       (**Mathematical_Comparison** <mathematical_comparative_declaration>))

22. <has_description> ::== (**Text_Description** <ASCII_text_filename>))

23 <ASCII_text_filename> ::==
a sequence of <character>*s referring to an ASCII text file*

24 <value> ::= <numeric_value> | <text_value>

25 <numeric_value> ::= {<number_sign>} <unsigned_number>

26. <text_value> ::= <string>

27. <number_sign> ::== + | -

28. <unsigned_number> ::== <digit>+ | <digit>* .<digit>+

29. <string> ::== "<character>+"

30. <character> ::== *any ASCII character*

31. <digit> ::== **0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**

APPENDIX B

DRIVE TUTORIAL MANUAL

Table of Contents

List of Figures

List of Tables

Tutorial Chapter 1
Starting Up

1.1 Overview

The **D**esign **R**ationale for the **I**nformation phase of **V**alue **E**ngineering (DRIVE) prototype system captures the owners' and designers' rationale behind the creation of a design. The DRIVE prototype is a highly graphical application that allows users to create a three-dimensional building model as well as document their design decisions in an unobtrusive manner. DRIVE also allows users to attach the rationale behind their design decisions to existing design drawings, whether two-dimensional or three-dimensional. DRIVE can therefore capture design rationale in real-time or in a retrospective basis.

The rationale behind the various design decisions is essential in the performance of a value engineering study. A value engineering study attempts to identify unnecessarily costly design items and suggest design alternatives to reduce costs without reducing the quality of a design. A value engineering study team must have access to the rationale behind a particular design. Access to design rationale allows them to properly suggest design alternatives.

DRIVE focuses on helping a value engineering study team during the information gathering phase only. DRIVE accomplishes this task through a graphical interface to retrieve design rationale embedded within the three-dimensional building model. DRIVE can be the foundation of a much larger Computer-Aided Value Engineering system capable of supporting the entire value engineering process.

1.2 System Requirements

Please make sure that your computer has the following hardware and software requirements before installing the DRIVE application:

- 486 processor or higher
- Math coprocessor
- 8MB RAM or more
- VGA or higher resolution monitor
- 1 MB of available disk space for the DRIVE system files
- 5 MB of available disk space for the DRIVE tutorial files (optional)
- DOS 5.0 or higher
- Windows 3.1 or higher running in 386-Enhanced mode
- AutoCAD Release 12 or higher
- Kappa-PC 2.3 or higher

1.3 Installation Steps

The following steps show how you can install the DRIVE system on your computer if the Program Manager is your Windows shell. If you are using another shell, please contact the developers for assistance.

- Start Windows.
- Place the DRIVE Installation disk into one of your floppy drives (usually A or B).
- Choose `File|Run` from the Program Manager menu
- Type `x:\install`, where `x` is the drive where you inserted the installation disk. For example, type `a:\install` if the installation disk is in drive A.
- Press the `Enter` key.
- The installation program now displays a window asking you for the directory to install DRIVE, which options to install, and the locations of Kappa-PC and AutoCAD. Please modify the various parameters as necessary, then click the `Start Installation` button to begin installing files to your hard disk. The DRIVE program starts after the installation program has completely installed the files to your hard disk. If you have not yet installed either Kappa-PC or AutoCAD on your computer, click the `Quit Installation` button, install the required applications, and follow these installation steps again.

1.4 Login Procedure

The DRIVE system starts by displaying a login window similar to the one shown in Figure 1-1. If you are a new user, click over the text marked `<new user>`. DRIVE now asks you for some information as shown in Figure 1-2. If you click over the text corresponding to an existing user, DRIVE asks for the password and encryption key values.

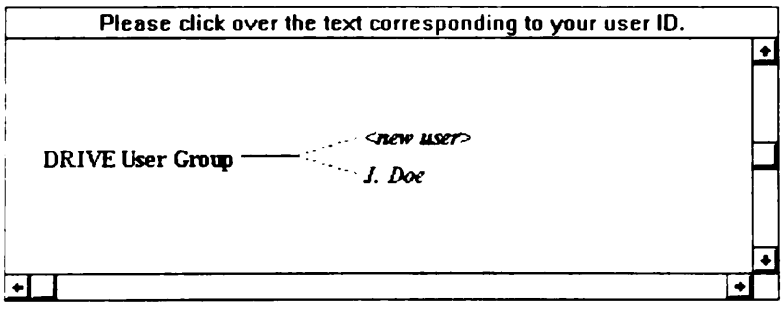


Figure 1-1 Login Window

The `User ID` text entry box contains the text displayed in the Login Window (for example, `J. Doe` in Figure 1-1). The `User ID` text can be as long as 200 characters and can contain upper and lower case letters, numbers, and spaces. The `Password` text must be less than 32 characters and is case-sensitive. The

`Encryption Key` value must be an integer between 1 and 65535. This integer encrypts the password. The `Password` and `Encryption Key` input boxes are optional. If you wish to have a password, please complete both the `Password` and `Encryption Key` input boxes. The other input boxes ask for additional personal information that is self-explanatory. After completing all the necessary information, click the `Continue` button to continue on with the login process. After confirming your password and encryption key, DRIVE asks for your designation as shown in Figure 1-3.

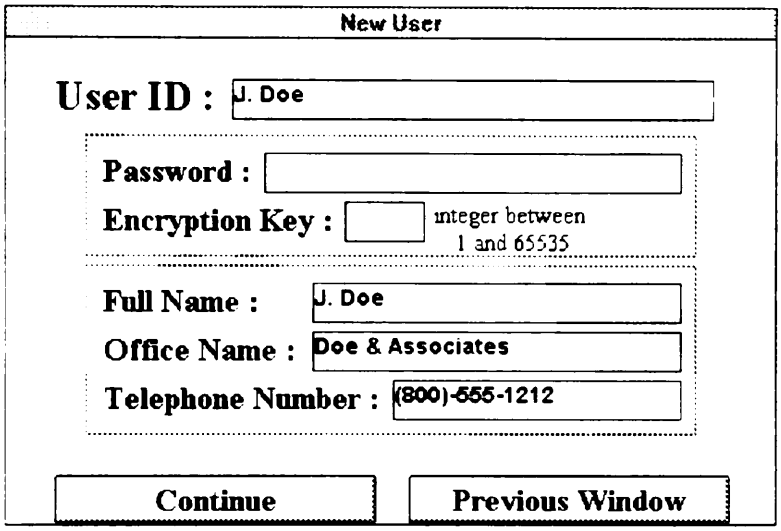


Figure 1-2 New User Login Window

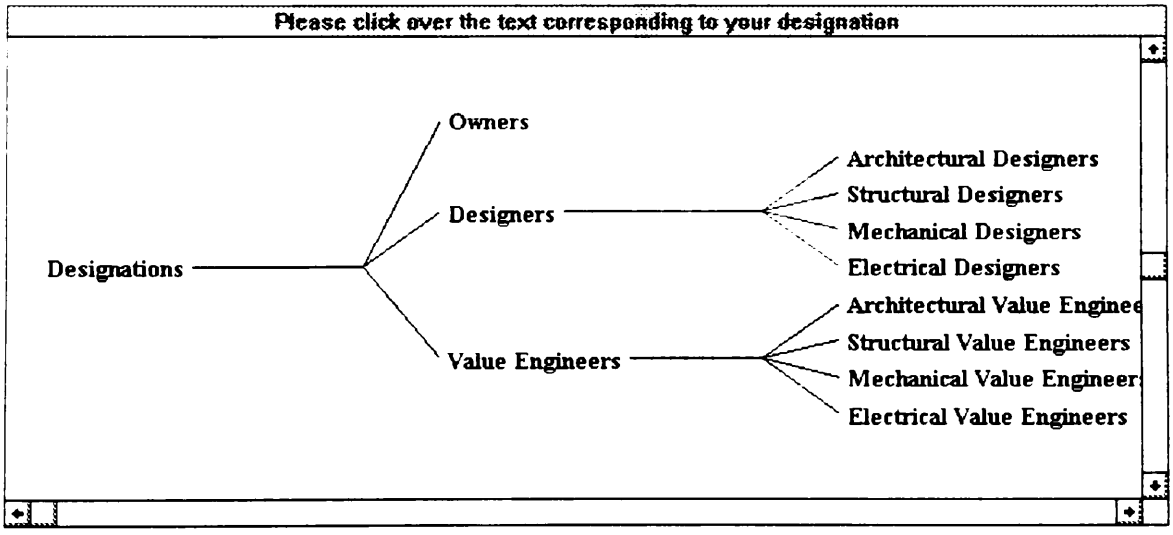


Figure 1-3 Designations Window

The Designations Window is one of several Entity Hierarchy Windows used in the program. For the Entity Hierarchy Windows, clicking over one of the text

identifiers shows a pop-up menu (Figure 1-4) in which you can choose an applicable action. The `Accept` and `Choose Another` menu choices allow you to accept or select another text identifier to be the one closest to the information required. For example, the Designations Window requires you to select the identifier closely matching your actual designation. There are also several menu choices in the pop-up menu shown on Figure 1-4 that work similarly for all Entity Hierarchy Windows. These are the `Show SubClasses`, `Hide SubClasses`, `Focus`, `Focus Parent`, and `Focus Ancestor` menu choices. `Show SubClasses` and `Hide SubClasses` allow you to show or hide, respectively, the sub-classes of a particular entity. `Focus` allows you to make a particular entity the leftmost entity displayed in Entity Hierarchy Windows. `Focus Parent` is similar to `Focus` with the exception that the leftmost entity displayed is the parent of a particular entity. For example, the parent of **Architectural Designers** is **Designers**, and the parent of **Designers** is **Designations**. `Focus Ancestor` displays the base entity or ancestor of a particular entity. The ancestor of both **Architectural Designers** and **Designers** is **Designations**.

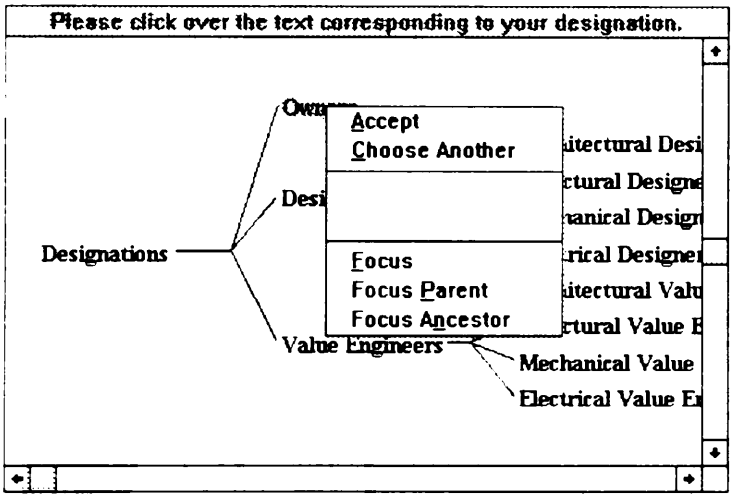


Figure 1-4. Pop-up Menu

Click over the text description closely resembling your particular designation and select `Accept` from the pop-up menu. Once you have completed the login process, the system shows the DRIVE Main Window.

Tutorial Chapter 2
Building Functional Description

Chapter 2 shows how to create a functional description of a building. In this chapter, you will learn how to work with DRIVE projects, define space types for the building, define functions at the building and space levels, and display a Functional Analysis Systems Technique (FAST) diagram used in value engineering studies.

2.1 Creating a New Project

Projects are groups of DRIVE data files that contain the design description and design rationale for a specific building project. Choose **Project|New** to create a new project. Figure 2-1 shows the New Project template. Type in **My Tutorial Project** or your preferred project name in the space provided. Project names can contain both upper and lower case letters, numbers, and spaces. DRIVE is capable of handling project names up to 200 characters long. Click the **OK** button to create the data files for the new project

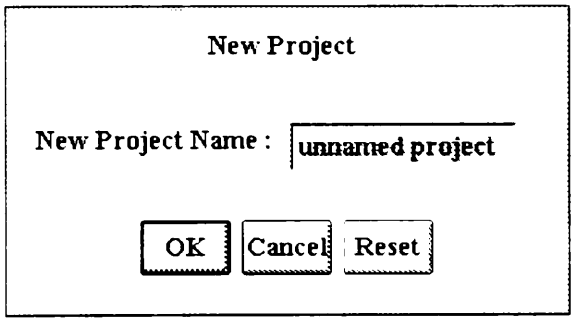


Figure 2-1. New Project Template

The next step in creating a project is selecting a building type. Figure 2-2 shows another kind of Entity Hierarchy Window called the Building Type Selection Window. Clicking over one of the text identifiers inside this window shows a pop-up menu (Figure 1-4). For this tutorial, you are creating a new senior high school building project. Search for and click over the text identifier **Senior High School Buildings** to display the pop-up menu. Select **Accept** to create a typical senior high school building with which you can build upon for your particular design.

2.2 Window Menu

The **Window** menu allows you to switch to the various windows used by the DRIVE application. This section gives a summary of the various windows. Figure 2-3 shows the **Window** menu. There are three "stage" windows, namely, **Concept Window**, **Design Window**, and **Value Engineering Window**. These windows

contain various interface objects that can manipulate the values of the attributes of the various objects as it goes through different stages. For example, the gross area of a classroom might have different values for the As Required, As Designed, and As Value Engineered stages. Each of these "stage" windows has its own `Task` menu containing the various tasks you may perform on a particular window.

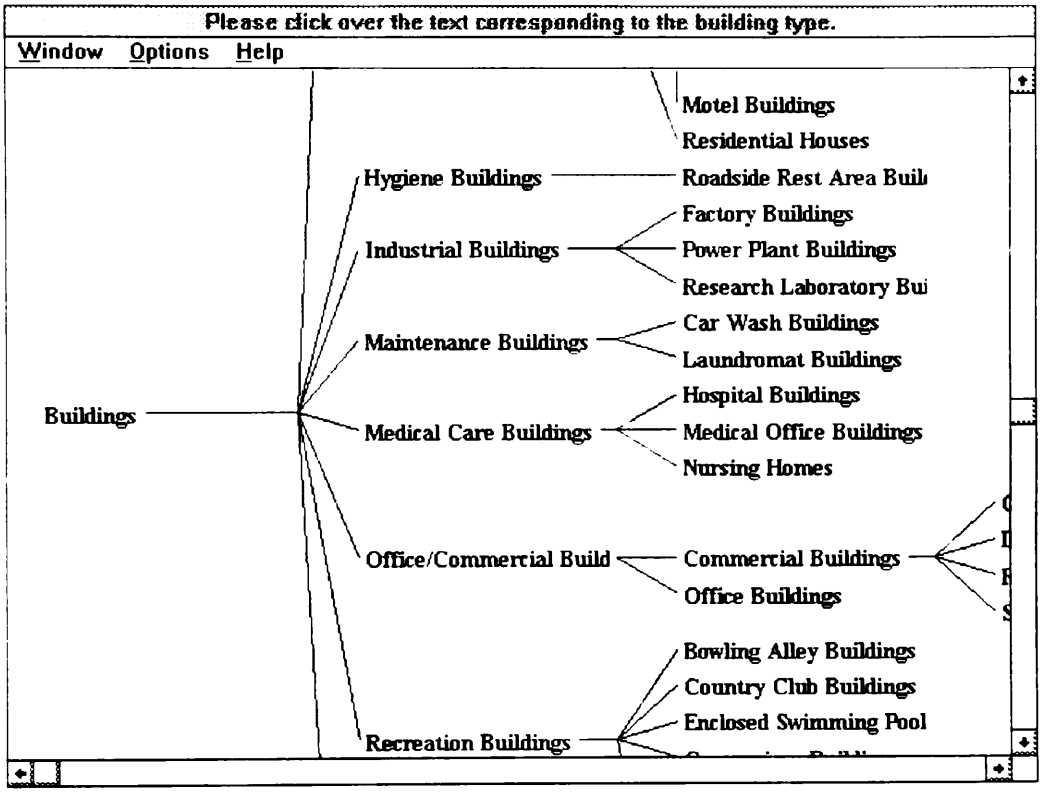


Figure 2-2 Building Type Selection Window

The `AutoCAD Window` is the DRIVE's graphical model design module. The `AutoCAD Window` menu choice allows you to quickly switch to AutoCAD. There are also three "secondary" window types, namely, `Object-Attribute Windows`, `Rationale Windows`, and `Conflict Windows`. These window types allow you to input values, capture rationale, and resolve conflicts for the different attributes of the various objects in the design model. For a more thorough discussion of "secondary" windows, please refer to Sections 2.7.3, 3.3, and 8.2 for `Object-Attribute Windows`, `Rationale Windows`, and `Conflict Windows`, respectively.

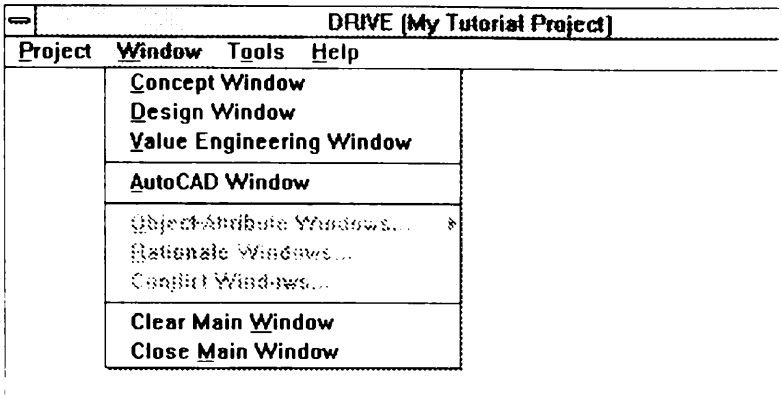


Figure 2-3. Window Menu

2.3 Adding a Text Description

Select the `Window|Concept Window` menu item from the DRIVE `Main Window`. DRIVE now shows a new window called the `Concept Window`. Select the `Task|Building Requirements` from the `Concept Window` to display the Building Requirements Task Form (Figure 2-4). This task form allows you to view and change the text description and the performance requirements of the building. This section discusses the `Text Description` box. Section 2.7 discusses the `Performance Types` and `Performance Parameters` boxes.

The Text Description box allows you to attach text descriptions to the various objects in the building model. Please bear with the current limitation of the DRIVE prototype regarding word wrapping. Once you reach the right edge of the Text Description box, please hit the Enter key to move to the next line. A future version of DRIVE may include automatic word wrapping capability. For the tutorial project, please input the following text (or, if you prefer, your own text description) into the Text Description box:

The Anytown Planning Division estimates that the existing facilities for Anytown High School will be inadequate for the expected enrollment by 1995. The addition of twelve classrooms allows the school to comply with building code standards up to 2005. The new building will have seven classrooms, a reading room, and some administrative offices. The relocation of the administrative offices to the new building will provide space for five additional classrooms in the existing building.

2.4 Previewing a Typical Building

You may wish to check the default data contained in a typical building before starting the actual design process. From the `Window` menu, select the `Concept Window` item. DRIVE now shows a new window called the `Concept Window`.

Select the **Task|Building Spaces** menu item to display the Building Space Types Task Form (Figure 2-5). This task form lists both the current space types included in the building and all the available space types. Since you have not yet made any changes to the building data, this right list shows the default space types for a typical senior high school building.

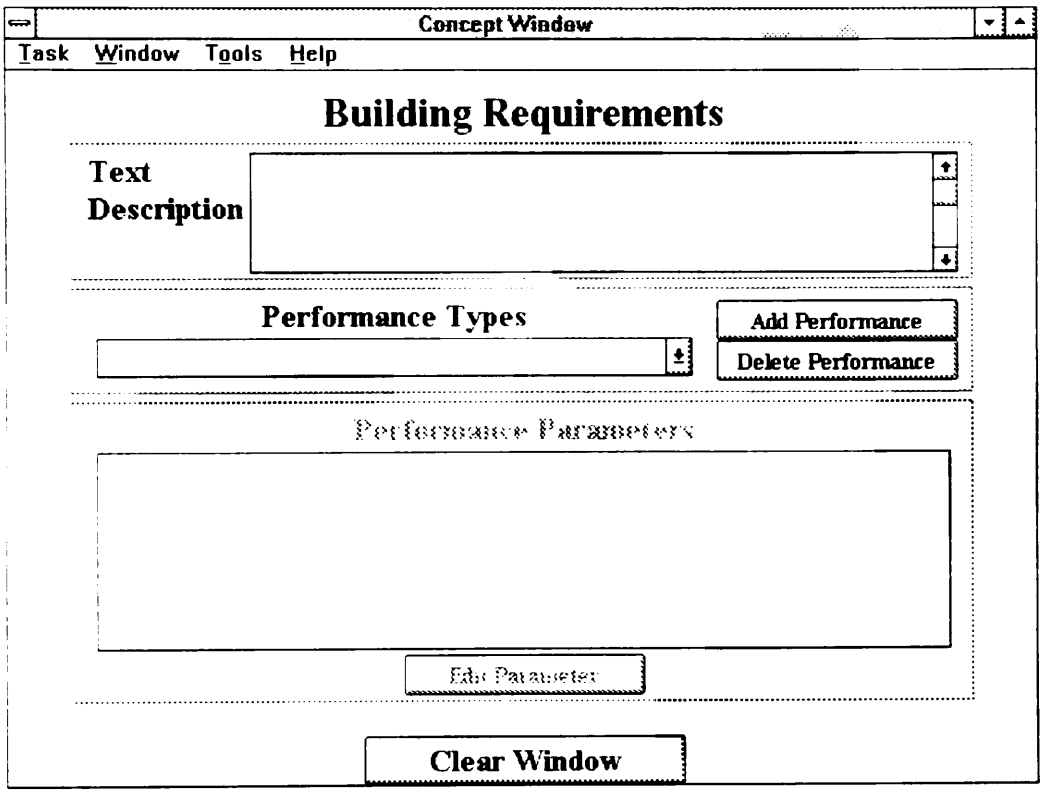


Figure 2-4. Building Requirements Task Form

Another type of default data you may wish to check is the FAST Diagram. Value engineers use the Functional Analysis Systems Technique (FAST) as an aid in determining unnecessary costs of the system. The FAST Diagram is a graphical representation of the functional breakdown of a system. From the **Window** menu, select the **Value Engineering Window** item. DRIVE shows another window called the **Value Engineering Window**. If you have a monitor having a resolution of at least 800x600, you may wish to arrange these windows such that one window will not totally cover another window. Select the **Task|Display FAST Diagram** menu item to display the FAST Diagram (Figure 2-6). Since you have not yet made any changes to the building data, Figure 2-6 show the default FAST Diagram for a typical senior high school building.

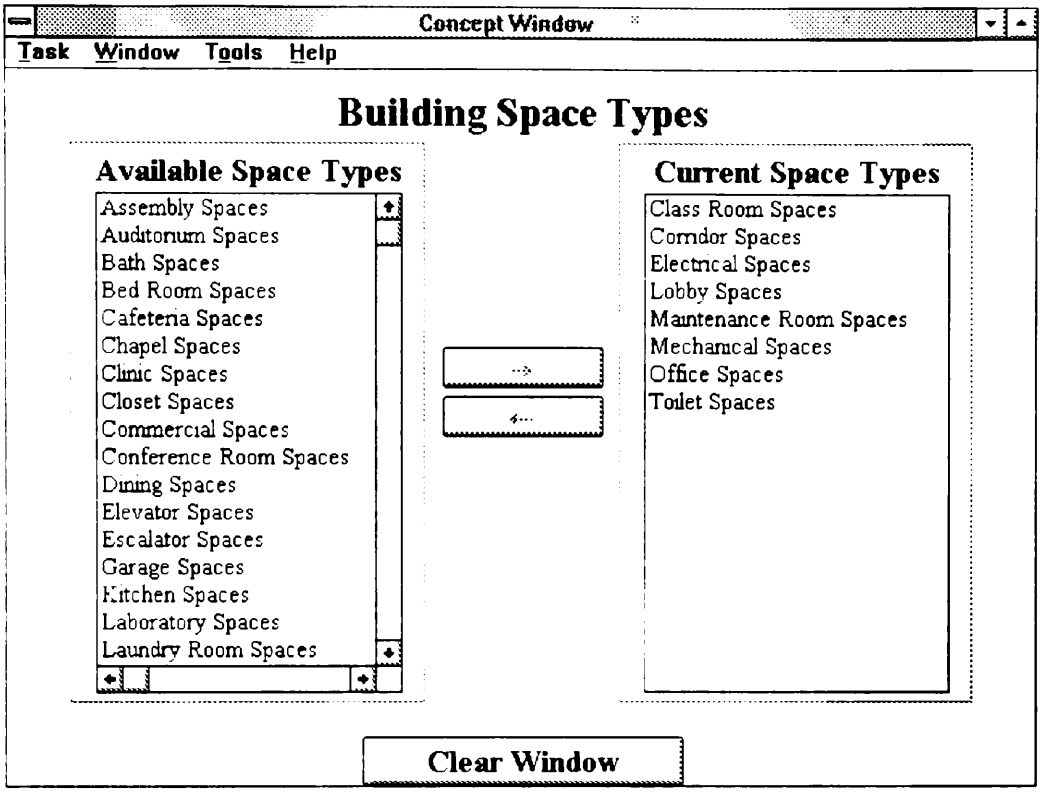


Figure 2-5 Building Space Types Task Form

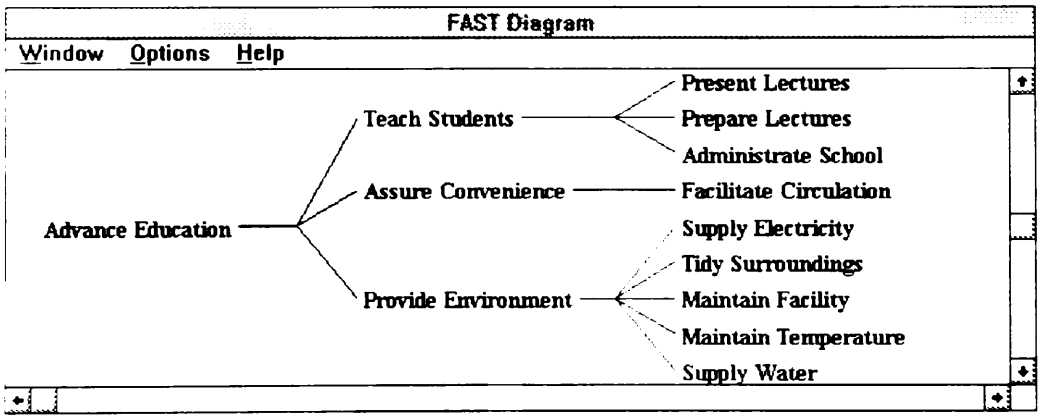


Figure 2-6. FAST Diagram

2.5 Entity Hierarchy Window Menus

Entity Hierarchy Windows usually have a menu bar allowing you to perform operations on the window. The menu bar in Figure 2-6 is the same as those found in most Entity Hierarchy Windows. The `Window` menu contains three items: `Print Window`, `Copy Window To Clipboard`, and `Close Window`. `Print Window` sends an image of the Entity Hierarchy Window to the default printer as specified in Windows' Control Panel. `Copy Window To Clipboard` sends an image to

Windows' Clipboard, thereby allowing you to paste this image into other applications such as word-processors. `Close Window` hides the window.

The `Options` menu contains three items: `Text Scale`, `Window Position`, and `Window Size`. `Text Scale` changes the size of the text identifiers in any Entity Hierarchy Window. `Text Scale` asks for the Text Size Parameters for both the horizontal and vertical scales. These parameters must be between 0.01 and 4. The larger the parameter, the longer or larger the space allocated for the text identifier. `Window Position` and `Window Size` specify the position and size where Entity Hierarchy Windows appear. The values for these parameters are in pixels. Please bear with the limitations of the DRIVE prototype regarding Entity Hierarchy Window positioning and sizing. A future version of DRIVE may allow dynamic positioning and resizing of these windows, thereby eliminating the need for the `Window Position` and `Window Size` menu items.

Another limitation of Entity Hierarchy Windows is that they can only be one such window displayed at any one time. This is a limitation of the prototyping environment, Kappa-PC. Until Kappa-PC overcomes this limitation, any version of DRIVE will have this limitation.

2.6 Building Space Types Customization

Usually, the default building space types do not correspond to the actual space types contained in the building. You can change the building space types through the `Task|Building Space Types` menu item in the `Concept Window`. Table 2-1 shows the building space types for this tutorial project.

Table 2-1. Tutorial Project Building Spaces

Current Space Types to be retained	Available Space Types to be added	Current Space Types to be deleted
Class Room Spaces	Clinic Spaces	Toilet Spaces
Corridor Spaces	Conference Room Spaces	
Electrical Spaces	Elevator Spaces	
Lobby Spaces	Garage Spaces	
Maintenance Room Spaces	Laboratory Spaces	
Mechanical Spaces	Library Spaces	
Office Spaces	Stairway Spaces	
	Storage Room Spaces	
	Workshop Spaces	

Figure 2-5 displays two buttons containing arrows. These denote the direction you wish to move a particular space type. For example, if you wish to move **Clinic Spaces** from the `Available Space Types` column to the `Current Space`

Types column, click over **Clinic Spaces** and click the → button. You can also highlight all the space types you wish to move before clicking the arrow buttons. Another method of moving space types is by double-clicking on a text inside a column.

For this tutorial project, try to move the different space types from one column to another such that the Current Space Types column in the Concept Window ends up with the 16 space types enumerated on the first two columns of Table 2-1. Click the Accept Changes button to accept the list contained in the Current Space Types column as the space types for the current building project. The system displays a confirmation message before deleting a space type (in this example, Toilet Spaces). Click the Yes button in the confirmation message to delete a space type.

2.7 FAST Diagram Customization

DRIVE generates the FAST Diagram from the **General Function** and **Particular Functions** attributes of the **Building** object and the various **Building Spaces** objects. The **General Function** and **Particular Functions** attributes are parameters of the performance type **Suitability Performance**. DRIVE groups related attributes into a performance type. These performance types help in presenting a manageable list of attributes or parameters for viewing and editing.

2.7.1 Viewing the Building Functions

As mentioned in Section 2.3, selecting the Task|Building Requirements menu item allows you to view and change the performance requirements of the building. Switch to the Concept Window and select the Task|Building Requirements menu item. Click the pull-down arrow corresponding to the Performance Types box and select Suitability Performance from the pull-down list. The Performance Parameters box shows the various parameters of **Suitability Performance** (Figure 2-7). Scroll the through the parameter list to view the **General Function** and **Particular Functions** of the building. Note how these functions correspond to the FAST diagram.

2.7.2 Viewing Building Space Functions

The each space type existing in the building has its own **General Function** and **Particular Functions**. You can access these functions through the Task|Show Building Spaces Hierarchy menu item in the Concept Window. Figure 2-8 shows the Building Spaces Hierarchy. Click over the text Office Spaces and select Edit Requirements from the pop-up menu. The Concept Window now displays something very similar to Figure 2-7. However, instead of accessing **Building** requirements, you are accessing **Office Spaces** requirements. You can now attach a text description to **Office Spaces**. The procedure to view attributes

of **Office Spaces** is similar to the procedure used in viewing **Building** attributes. Note again the correspondence between the **Office Spaces** functions to the FAST diagram. If you prefer, you can repeat this procedure for all the building space types to attach text descriptions and see the basis for the FAST diagram.

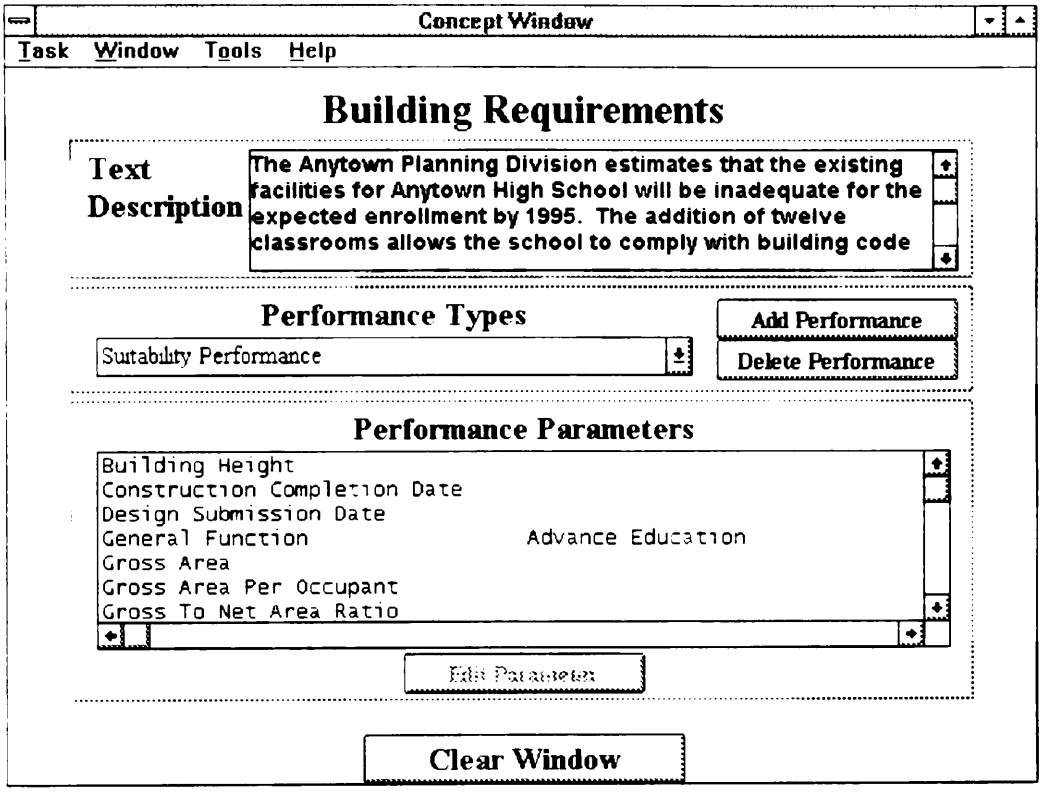


Figure 2-7. Accessing Building Performance Parameters

2.7.3 Using Object-Attribute Windows

Object-Attribute Windows allow you to change the current values of an attribute of any design object. The default functions taken from a typical building do not always hold true for the building being designed. For this tutorial project, there are differences in the default values and the project-specific values for **Particular Functions** of **Office Spaces**. If the `Concept Window` is not showing the attributes of **Office Spaces**, select the `Task|Show Building Spaces Hierarchy` menu item, click on `Office Spaces`, select `Edit Requirements` from the pop-up menu, and set the `Performance Type` to `Suitability Performance`. The `Concept Window` should now look very similar to Figure 2-7 with the exception that you are now accessing **Office Spaces** instead of **Building**. Scroll through the parameter list so that **Particular Functions** is visible. Click on the line containing **Particular Functions** to highlight this line and enable the `Edit Parameter` button. Click the `Edit Parameter` button to create a new object-attribute window (Figure 2-9). For the tutorial project, the **Particular Functions** of **Office Spaces** are **Counsel**

Students and **Administrate School**. Edit the object-attribute window such that it reflects the new values of Particular Functions (Figure 2-10). For attributes capable of having multiple values, such as **Particular Functions**, place each unique value on separate lines. DRIVE treats a single line of text as one particular value. Use the `Enter` key to separate multiple values. Click the `Accept Changes` button in the object-attribute window. Switch to the `Value Engineering Window` and select the `Task|Display FAST Diagram` menu item to display the FAST diagram. Note how both the parameter list in the `Concept Window` and the FAST diagram changed to reflect the changes in the **Particular Functions** attribute of **Office Spaces**.

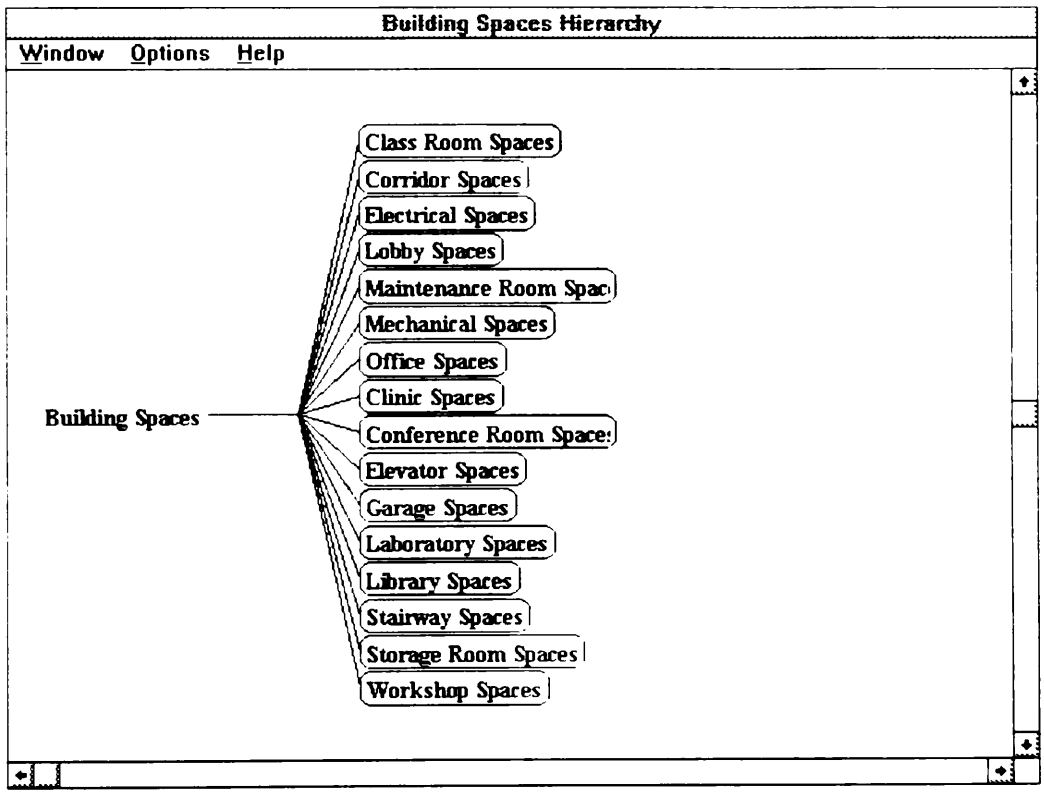


Figure 2-8. Building Spaces Hierarchy

The `Close Window` button is visible only when you have not yet made any changes to the values inside the object-attribute window. This button closes the object-attribute window. Once you made some changes to the values inside the object-attribute window, the `Close Window` button changes to the `Discard Changes` button. The `Discard Changes` button allows you to close a particular object-attribute window without committing any changes to the system.

The procedure to change the value of most object-attributes is similar to the procedure you used in changing the value of the attribute **Particular Functions** of

the object **Office Spaces**. However, there are a few object-attributes that have a slightly different procedure. An example is the **General Function** attribute of a space type. Edit the **General Function** attribute of the object **Clinic Spaces** to create another type of an object-attribute window (Figure 2-11). In these type of object-attribute windows, you do not need to type in the values, rather you select a value from a list of possible values. Click the pull-down arrow to display the list of possible values for **General Function** This list corresponds to the **Particular Functions** of **Building**. For Clinic Spaces, select `Assure Convenience` as its **General Function**. Click the `Accept Changes` button in the object-attribute window to formally change the value of the attribute **General Function** of the object **Clinic Spaces**.

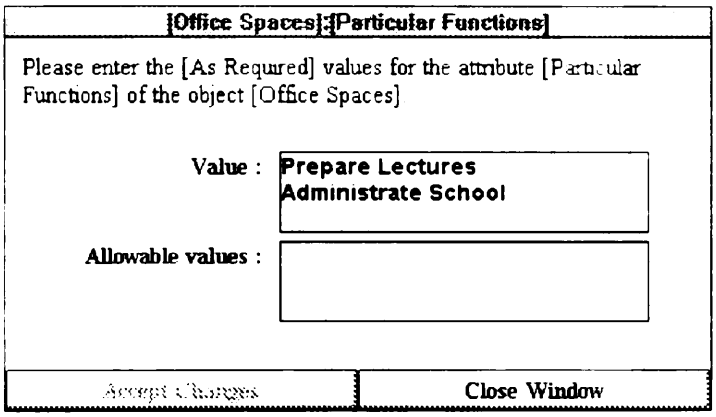


Figure 2-9. Object-Attribute Window

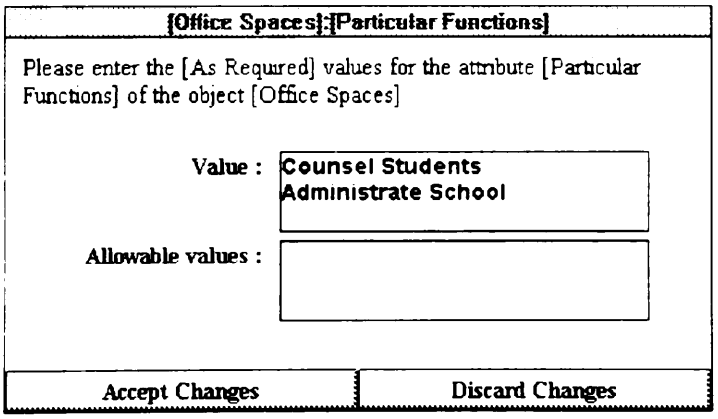


Figure 2-10. Revised Particular Functions of Office Spaces

Using the procedures shown in this section, please complete the customization of the FAST diagram for this tutorial project by changing, as necessary, the values of the **General Function** and **Particular Functions** attributes of the various space types. Table 2-2 shows these values. The DRIVE system imposes the restriction that a space type must have a **General Function** defined before you can add

Particular Functions Although the DRIVE does not require that you define the **General Function** and **Particular Function** attributes for the various space types, these values are necessary for generating FAST and Functional Cost Breakdown Diagrams. Chapter 5 discusses Cost Breakdown Diagrams.

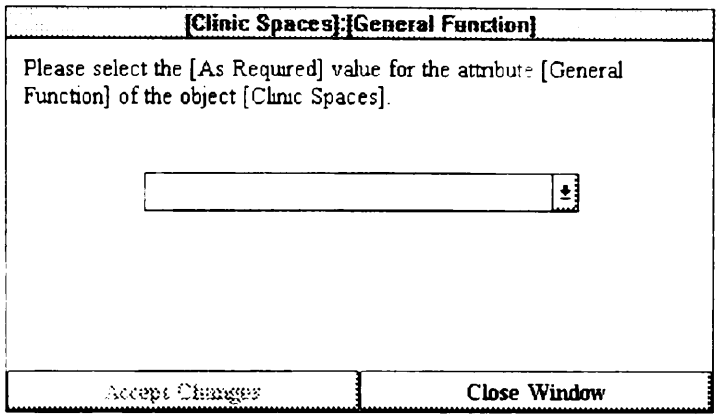


Figure 2-11. Selection Type Object-Attribute Window

2.8 Project Menu

Choose the `Project|Save` menu item from DRIVE's `Main Window` to write the changes you made to the hard disk. DRIVE displays a message asking you for confirmation of your `Save` request. Click the `Yes` button to save your project to the disk.

Choose the `Project|Close` menu item to remove all the project specific data from DRIVE's working memory. If there are any unsaved information when you issue this command, DRIVE displays a message asking if you wish to save your current project to disk before removing the project from the working memory.

The `Project|Logout` menu item closes the current project, returns the system to the `Login Window`, and allows another user to use the DRIVE application. The `Project|Exit` menu item exits the DRIVE application.

Table 2-2 Tutorial Project Space Type Functions

Space Type	General Function	Particular Functions
Class Room Spaces	Teach Students	Present Lectures
Clinic Spaces	Assure Convenience	Treat Injuries
Conference Room Spaces	Assure Convenience	Facilitate Meetings
Corridor Spaces	Assure Convenience	Facilitate Circulation
Electrical Spaces	Provide Environment	Supply Electricity
Elevator Spaces	Assure Convenience	Facilitate Circulation
Garage Spaces	Provide Environment	Transport Supplies
Laboratory Spaces	Teach Students	Prepare Experiments
Library Spaces	Teach Students	Study Lessons
Lobby Spaces	Assure Convenience	Facilitate Circulation
Maintenance Room Spaces	Provide Environment	Tidy Surroundings Maintain Facility
Mechanical Spaces	Provide Environment	Maintain Temperature Supply Water
Office Spaces	Teach Students	Counsel Students Administrate School
Stairway Spaces	Assure Convenience	Facilitate Circulation
Storage Room Spaces	Provide Environment	Store Supplies
Workshop Spaces	Provide Environment	Maintain Facility

Tutorial Chapter 3
Rationale Capture Module

Chapter 3 shows how to use the rationale capture module. In this chapter, you will learn how to work with existing DRIVE projects, make copies of projects, use the various modes of rationale capture, and use the rationale capture windows.

3.1 Opening and Copying an Existing Project

The DRIVE system allows you to work with a set of related data files through a graphical point and click interface. Choose **Project|Open** to select an existing project to work with. Figure 3-1 shows the graphical interface for opening project files. Clicking over one of the project text descriptions loads the various files associated with that project. For this tutorial, you may click on **My Tutorial Project** (or the name of the project you created in Chapter 2), or you may click on **Tutorial Chapter 3** (if you skipped portions of Chapter 2 or if you simply prefer using the original tutorial project).

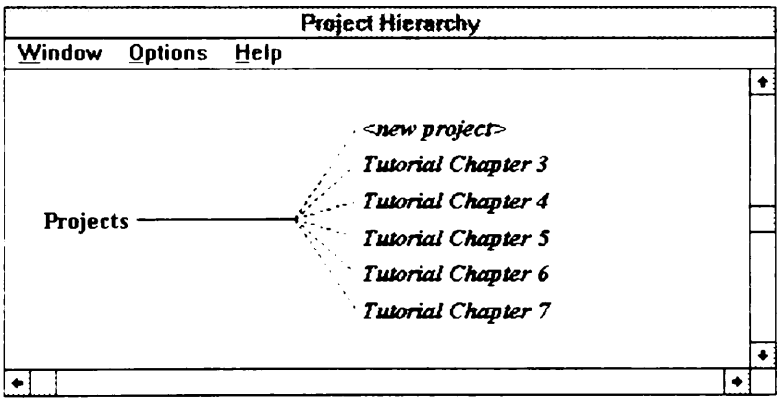


Figure 3-1. Opening Projects

If you chose to use **Tutorial Chapter 3**, DRIVE recommends to make a copy of this project rather than work with the original. Working with a copy of the tutorial project allows users (both you and others) to later access the original tutorial project in its unmodified state. To copy a project, open the project you wish to copy by using the **Project|Open** menu item. After loading all the project files, use the **Project|Save As** menu item to copy the entire contents of the original project to a new project. Figure 3-2 shows the Save As Project template. Type in **My Tutorial Project - Chapter 3** or the name of your preference in the space provided. Click the **OK** button to copy the original project's data files into your new project.

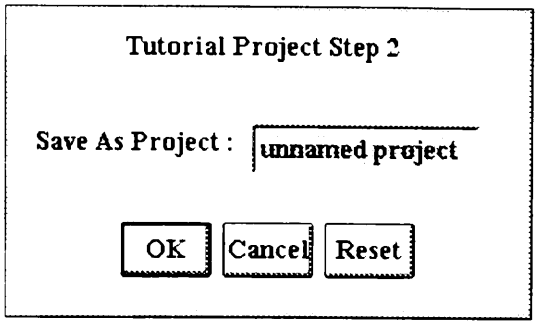


Figure 3-2. Save As Project Template

3.2 Rationale Capture Module

The Rationale Capture Module allows you to express the reasoning or rationale behind a particular value of an object-attribute. The `Tools Menu` contains three items: `Rationale Capture`, `Data Verification`, and `Knowledge Editor`. This section describes the rationale capture module. Chapters 8 and 9 describe in detail the data verification and knowledge editor modules, respectively.

Figure 3-3 shows the various modes of operation of the rationale capture module. `Fully Automatic` triggers the rationale capture module every time a value changes for an object-attribute. `Next Assertion Only` captures rationale only for any changes made to next object-attribute window (Section 2.7.3) you process. After an object-attribute window initiated rationale capture, this module switches `Off`. If the Rationale Capture Module is `Off` and a value changes in an object-attribute, this value change creates a pending rationale. `Process Pending Rationale` allows you to select a pending rationale to process. This section points out the various differences between the `Fully Automatic`, `Next Assertion Only`, and `Off` modes of operation. Section 6.1 discusses the procedure for processing pending rationale. The `Unload Rationale Capture Module` menu item removes this module from the working memory of DRIVE.

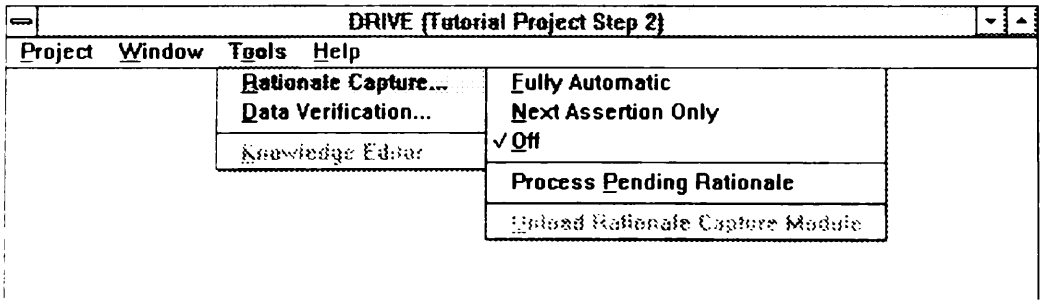


Figure 3-3. Rationale Capture Menu

The four modes of operation (`Fully Automatic`, `Next Assertion Only`, `Off`, and `Process Pending Rationale`) allow for maximum flexibility in capturing rationale

for the project. You have control over when DRIVE asks you for rationale information. This helps in making the rationale capture process as unintrusive as possible.

3.2.1 Fully Automatic

Switch the rationale capture module to `Fully Automatic` mode by selecting the `Tools|Rationale Capture|Fully Automatic` menu item. Display the `Concept Window` by selecting the `Window|Concept Window` menu item. Edit the building requirements by selecting the `Task|Building Requirements` menu item. Set the performance type to **Suitability Performance** by clicking on the pull-down arrow corresponding to the `Performance Type` box and selecting `Suitability Performance` from the pull-down list. Edit the **Location** attribute of the **Building** by scrolling through the `Performance Parameters` list box, highlighting the `Location` attribute, and clicking the `Edit Parameter` button.

Figure 3-4 shows the object-attribute window created by following the instructions of the above paragraph. Type in the value `1002 Main Street, Anytown, USA` in the space provided. When rationale capture is on `Fully Automatic` mode, the `Accept Changes` button changes to `Initiate Rationale Capture` once changes occur in any of the text fields in the object-attribute window. Click the `Initiate Rationale Capture` button to start the rationale capture process. Figure 3-5 shows the rationale window created after clicking the `Initiate Rationale Capture` button.

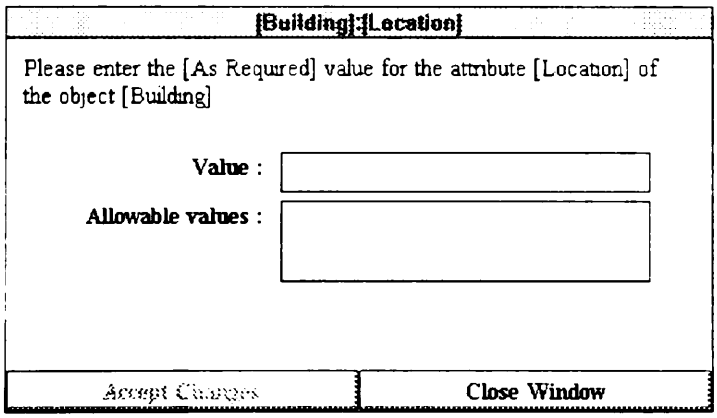


Figure 3-4. Object-Attribute Window

Return to the `Concept Window` by either clicking over it, using the `Alt-Tab` key combination, or by using the `Window` menu from the DRIVE `Main Window`. Scroll through the parameters list and edit the **Construction Completion Date** attribute. Change its value to `August 31, 1995`. Start the rationale capture process again

by clicking the `Initiate Rationale Capture` button. DRIVE now displays a new Rationale Window.

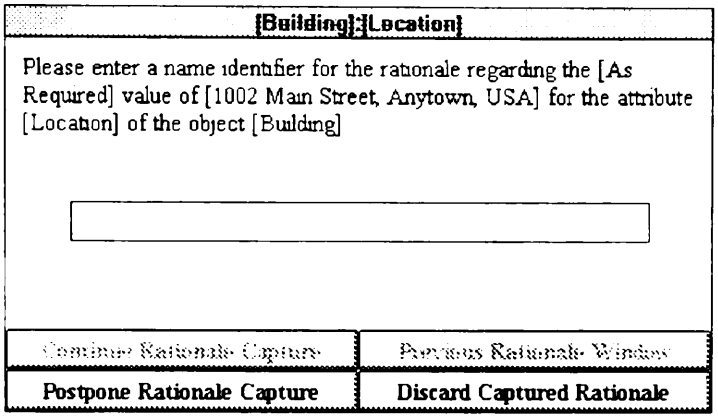


Figure 3-5. Example of a Rationale Capture Window

Figure 3-6 shows the Rationale Windows Menu. This menu contains all the rationale windows currently open. For this example, two rationale windows are currently open: the `[Building]:[Location]` and the `[Building]:[Construction Completion Date]` windows. Selecting any one of these items displays that particular rationale window. This principle applies to the other "secondary" windows (Object-Attribute Windows and Conflict Windows). The ability to have many "secondary" windows open further enhances the flexibility and unintrusiveness of the application. There is no need to close an existing "secondary" window if the need arises to work on another object-attribute.

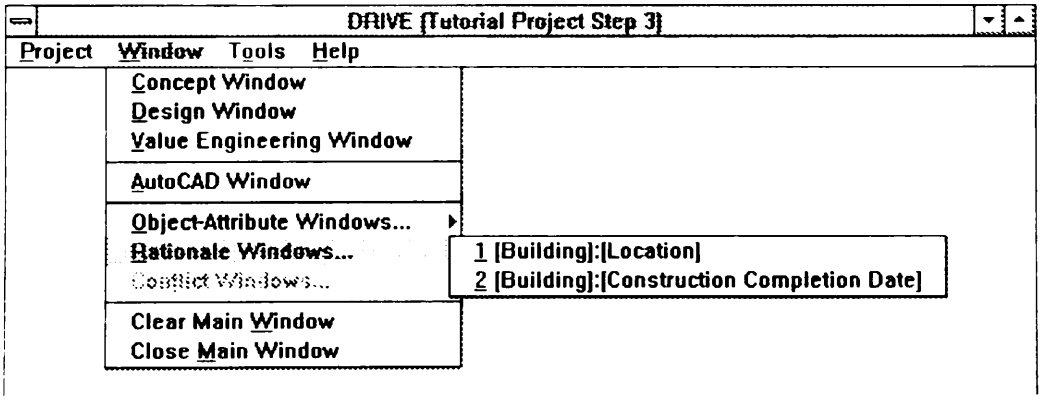


Figure 3-6. Rationale Windows Menu

3.2.2 Next Assertion Only

Switch the rationale capture module to `Next Assertion Only` mode by selecting the `Tools|Rationale Capture|Next Assertion Only` menu item. This mode needs to close all open rationale windows. DRIVE asks for confirmation before

proceeding with this command. Click the Yes button to confirm the Next Assertion Only command. Change the Performance Type to Economic Performance and edit the Construction Cost attribute. Change its maximum value to 1500000. Click the Initiate Rationale Capture button to create a new rationale window. Select the Tools|Rationale Capture menu item. Note that the rationale capture module is now Off.

3.2.3 Off

Verify if the rationale capture module is Off by selecting the Tools|Rationale Capture menu item. If it is not yet in the Off position, switch it to the Off mode. Change the Performance Type to Suitability Performance, edit the Number of Stories attribute, and change its value to 2. When the rationale capture module is in the Fully Automatic or Next Assertion Only modes, the Accept Changes button changes to Initiate Rationale Capture. Notice that when the rationale capture module is Off, the Accept Changes button becomes enabled and does not change to Initiate Rationale Capture. Click the Accept Changes button in this object-attribute window to formally change the value of the attribute **Number of Stories** of the object **Building** and to create a pending rationale for this particular object-attribute.

Click the Accept Changes button on all open object-attribute windows. This commits all changes in the object-attributes to the system.

3.3 Rationale Windows

The rationale capture process consists of a series of "question and answer" rationale windows. The use of a series of rationale windows breaks down the rationale capture process into smaller tasks. This is necessary so as not to overwhelm you with the complex task of expressing rationale. There are several kinds of rationale windows and some of the various sub-sections in this section discuss each of these window types.

3.3.1 Control Buttons

Figure 3-5 shows a kind of rationale window. Notice the four control buttons at the lower part of a rationale window. These control buttons are always present for all kinds of rationale windows. The names of these buttons are: Continue Rationale Capture, Previous Rationale Window, Postpone Rationale Capture, and Discard Captured Rationale. The only exception to this button naming convention occurs in the Rationale Comment windows (Section 3.3.5) where Conclude Rationale Capture replaces Continue Rationale Capture.

The Continue Rationale Capture button processes the rationale information on the current window and displays the next valid rationale window in the series.

DRIVE disables or grays-out this button when there is any missing information in the current window. The `Previous Rationale Window` button discards any new rationale information captured in the current window and displays a rationale window corresponding to a prior step in the series. DRIVE disables this button when there is no prior rationale window. The `Postpone Rationale Capture` button saves all the captured rationale up to the current window. This button relieves you of having to fully complete a rationale capture process once you start such a process. Section 6.1 describes how you can process postponed or pending rationale. The `Discard Captured Rationale` button discards all rationale captured up to the current window. The `Conclude Rationale Capture` button completes the rationale capture process.

3.3.2 Rationale Example 1: Building Occupants Minimum Value Requirement

Switch the rationale capture module to `Fully Automatic` mode. Edit the building requirements through the `Task|Building Requirements` menu item of the `Concept Window`. Change the minimum value of the attribute **Number of Occupants** to `200` and click the `Initiate Rationale Capture` button to start the rationale capture process.

3.3.3 Rationale Name

Figure 3-7 shows the Rationale Name Window. This type of rationale window allows you to define a descriptive name identifier for the rationale you are creating for this particular object-attribute. This name identifier can be any text up to 200 characters in length. Type in `Building Occupants Minimum Value Requirement` or your preferred name identifier in the space provided. Typing a text identifier enables the `Continue Rationale Capture` button. Click this button to proceed to the next rationale window.

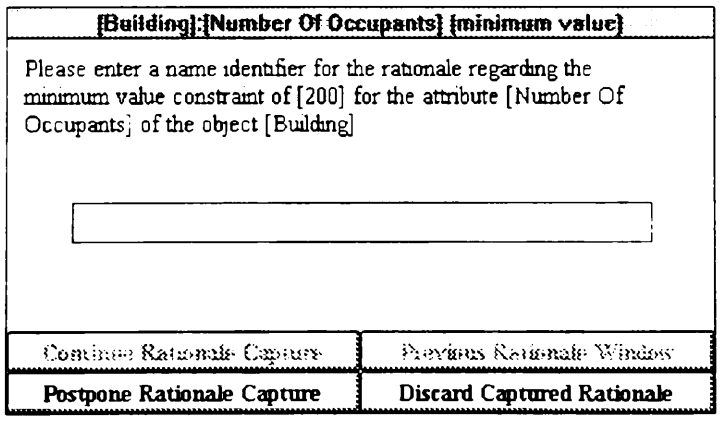


Figure 3-7. Rationale Name Window

3.3.4 Dependency Type

Figure 3-8 shows the Dependency Type Window. This type of rationale window allows you to define the type of dependency for this particular rationale. DRIVE currently recognizes three types of dependencies: `other object-attributes`, `owner requirements`, and `code requirements`. For this example, select `owner requirements` and click the `Continue Rationale Capture` button.

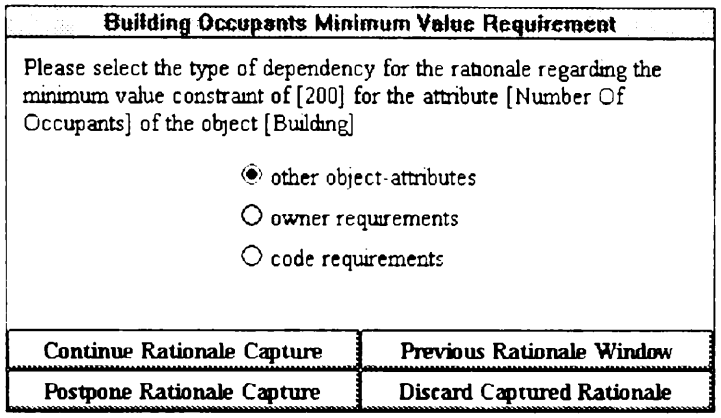


Figure 3-8. Dependency Type Window

3.3.5 Rationale Comment

Figure 3-9 shows the Rationale Comment Window. This type of rationale window allows you to place a text description for this particular rationale. DRIVE does not process this text in any manner and it is up to the users to interpret and use this text description. You can type your own text description or use the following as the text description for this rationale:

> *The Anytown Planning Division estimates that the provision of classroom spaces for at least 200 students would allow the school to comply with building code standards up to academic year 2004-2005.*

Complete the rationale capture process by clicking the `Conclude Rationale Capture` button.

3.3.6 Rationale Example 2: Building Height-Location Relationship

Edit the building requirements and change the maximum value of the attribute **Building Height** to `55`. Start the rationale capture process. DRIVE displays a new Rationale Name Window. Type in `Building Height-Location Relationship` or your preferred name identifier in the space provided in this window. Proceed to the Dependency Type Window and select `other object-attributes` as the dependency for this rationale. Click the `Continue Rationale Capture` button to proceed to the next rationale window.

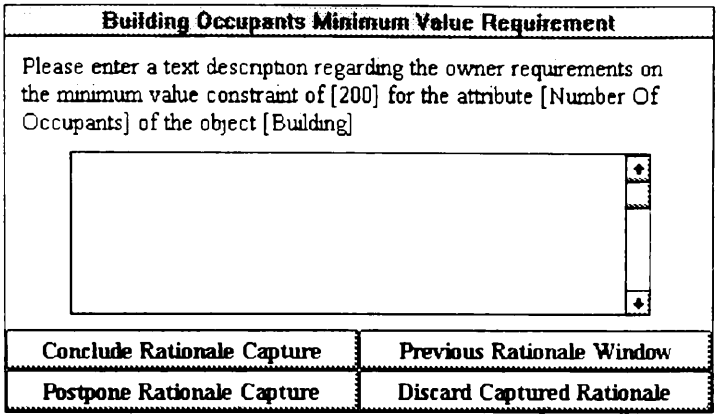


Figure 3-9. Rationale Comment Window

3.3.7 Object Selection

Figure 3-10 shows the Object Selection Window. This type of rationale window allows you to select the objects affecting this particular rationale. The object list contains all the objects defined in the database. You can select one or more objects that affect this rationale. DRIVE disables the `Continue Rationale Capture` button when there are no selected objects and enables it when there is at least one selected object. For this example, select `Building` as the only object affecting this rationale and click the `Continue Rationale Capture` button.

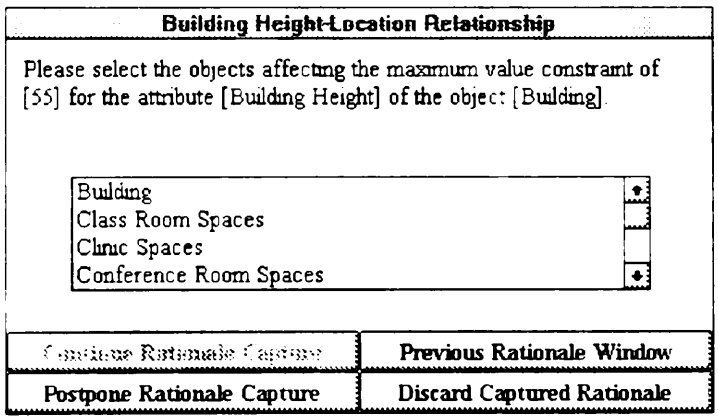


Figure 3-10. Object Selection Window

3.3.8 Attribute Selection

Figure 3-11 shows the Attribute Selection Window. This type of rationale window allows you to select the attributes of the selected objects affecting this particular rationale. The attribute list contains all the attributes of a particular selected object. Since there is one object affecting this rationale, the current rationale window has only one attribute list. The attribute selection window can create up to four lists of attributes had there been several objects affecting this rationale. If

there are more than four objects affecting this rationale, this window also creates a `More` button to access the attributes of these various objects. For this example, select `Location` as the only attribute affecting this rationale and click the `Continue Rationale Capture` button.

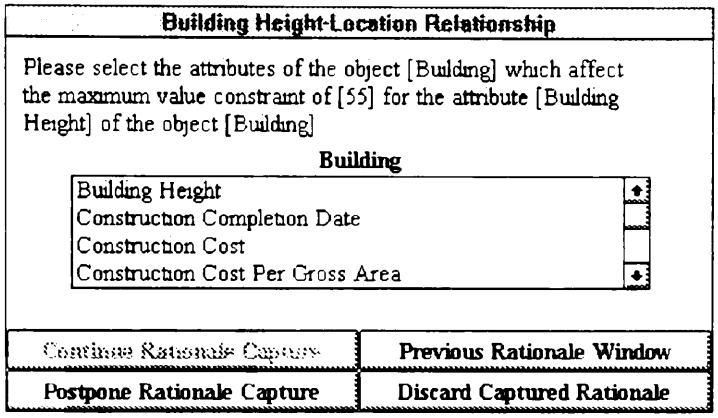


Figure 3-11. Attribute Selection Window

3.3.9 Relationship Type

Figure 3-12 shows the Relationship Type Window. This type of rationale window allows you to select how your selected object-attributes relate to the original object-attribute. DRIVE currently recognizes two kinds of relationships: `Logical` and `Mathematical`. Choose `Logical` if the rationale is in the form of a cause and effect relationship (`If-Then` or `If-Then-Else`). Choose `Mathematical` if the rationale is in the form of a mathematical equation. Sections 3.3.10 and 3.3.11 describe in more detail the Logical and Mathematical relationships, respectively.

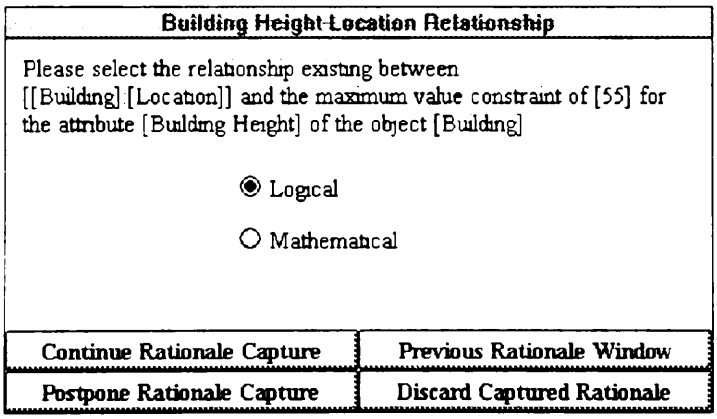


Figure 3-12. Relationship Type Window

3.3.10 Logical Relationship Definition

Figure 3-13 shows the Logical Relationship Window. This type of rationale window allows you to define how your selected object-attributes logically relate to the original object-attribute. DRIVE breaks the logical relationship into three sections: `If`, `Then`, and `Else`. Each of these sections has its own separate statement window where you can build a portion of the logical relationship. Only one of these windows can be active at any one time. Clicking on one of the buttons beside these windows makes the corresponding statement window active. The color of the button texts determine the state of these windows. Black text indicates an active window, while red text indicates an inactive window. The active statement window receives all the relationship statements created by the six statement buttons (`ObjAttrs`, `Constraints`, `Relations`, `Values`, `Logical`, and `Math`) at the lower portion of the Logical Relationship Window.

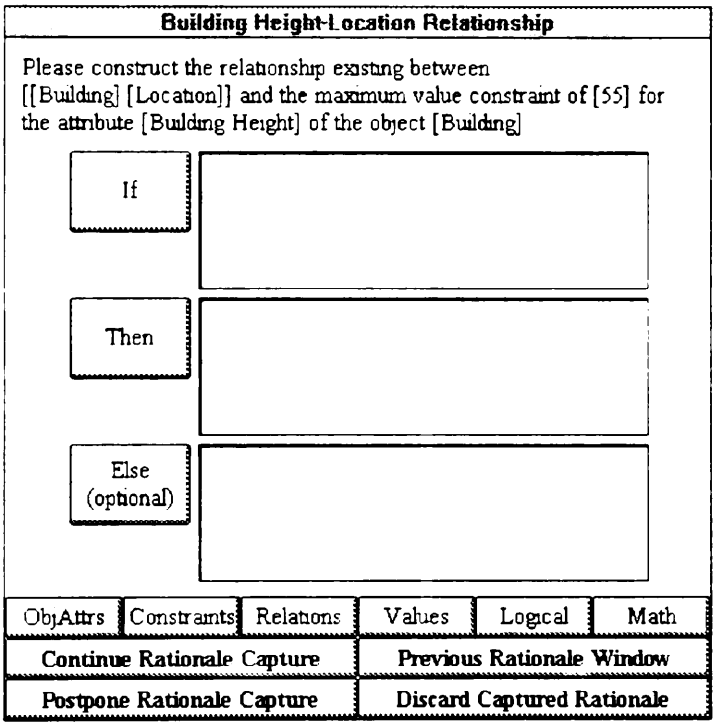


Figure 3-13 Logical Relationship Window

The statement buttons help in building the rationale relationship. Each of these buttons submit relationship statements to the active statement window. The `ObjAttrs` (short for object-attributes) and `Constraints` buttons both display a list of object-attributes relevant to this rationale. The `ObjAttr` button creates a statement reference to a particular object-attribute; while the `Constraints` button creates one to a particular constraint of a particular object-attribute. The `Relations`, `Logical`, and `Math` buttons creates a statement corresponding to a particular relational, logical, or mathematical operation. The `Values` button allows

you to create a statement containing a value from either a value already existing in an object-attribute or a user-specified value.

Set the active statement window to the `If` portion by clicking on the `If` button. The color of the text in the `If` section becomes black, while the other sections become red. Click the `ObjAttrs` button and select `[Building]:[Location]` from the list of object-attributes. This statement appears in the active statement window. Click the `Relations` button, select `=`, and note how DRIVE creates the statement in the active window. Next, click the `Values` button, select `object-attributes`, select `[Building]:[Location]`, and finally select `1002 Main Street, Anytown, USA`. This completes the `If` portion of this logical relationship.

Set the active statement window to the `Then` portion by clicking on the `Then` button. Click the ObjAttrs button and select `[Building]:[Building Height] (maximum value)`. Click the `Relations` button and select `<`. Figure 3-14 shows the Statement Window Editing Menu. Clicking over a text statement in the active statement window displays this menu. Currently, only `Delete Line` works in this menu. The final released version of DRIVE may enable the other two editing functions. Click over the text `is smaller than` in the active statement window and select `Delete Line` from the menu to remove this line from the window. Click the `Relations` button and select `=`. Click the `Values` button, select `specific value`, type `55`, and finally click the `OK` button to complete the `Then` portion of this logical relationship.

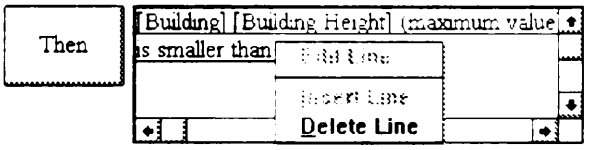


Figure 3-14. Statement Window Editing Menu

Click the `Continue Rationale Capture` button to accept the logical relationship and switch to the next rationale window (Rationale Comment Window). You can type your own text description or use the following as the text description for this rationale:

Anytown zoning regulations limit the height of buildings to 55 feet.

Complete the rationale capture process by clicking the `Conclude Rationale Capture` button.

3.3.11 Mathematical Relationship Definition

Figure 3-15 shows the Mathematical Relationship Window. DRIVE displays this window whenever you choose **Mathematical** as the relationship type (Figure 3-12). The definition of the mathematical relationship is similar to the procedure for defining a logical relationship. While the Logical Relationship Window allows you to relate object-attributes using an If-Then-Else condition, the Mathematical Relationship Window allows you to mathematically relate various object-attributes. Figure 3-15 shows an example of a mathematical relation between the attributes **Building Height** and the **Number Of Stories** of the object **Building**. It shows the **Building Height** should be greater than the **Number Of Stories** multiplied by 8. This reflects the requirement that the minimum story height is 8 feet.

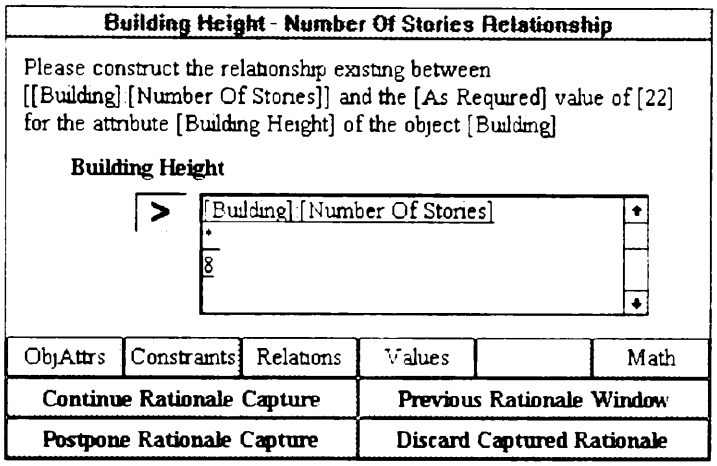


Figure 3-15. Mathematical Relationship Window

3.3.12 Multiple Value Rationale Capture

Figure 3-16 shows the Multiple Value Window. The system displays this window when capturing rationale for an attribute having two or more values. Figure 3-16 captures rationale for the values (**Administrate School** and **Counsel Students**) of the attribute **Particular Functions** of the object **Office Spaces**. This window allows you to define one rationale for all these values (Single Group) or to define different rationale for each value (Individual elements). Clicking the `Continue Rationale Capture` button displays the Rationale Name Window (Figure 3-7).

Click the `Accept Changes` button on all open object-attribute windows. Choose `Project|Save` to save all changes to the disk. Choose `Project|Close` to clear the working memory of the DRIVE application.

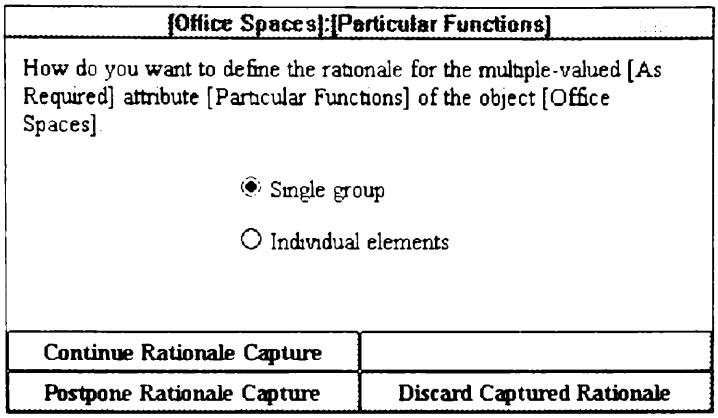


Figure 3-16. Multiple Value Window

Tutorial Chapter 4
Creating Building Spaces

Chapter 4 shows how to further customize a typical building by creating graphical representations of specific building spaces. In this chapter, you will learn how to define the building story parameters and use the Design and AutoCAD Windows.

4.1 Project Initialization

Using the procedure described in Section 3.1, open either `Tutorial Chapter 4` or the project you created in Chapter 3. If you are using the original DRIVE tutorial files (`Tutorial Chapter 4`), please select `Project|Save As` and use `My Tutorial Project - Chapter 4` (or your preferred project name) as the new project name.

4.2 Initializing AutoCAD

Figure 4-1 shows the various parameters DRIVE needs to initialize the AutoCAD Window. The `Length Units` parameter specifies the unit of measure for all objects in a project. The `Drawing Limits`, `Grid Spacing`, and `Snap Spacing` check boxes allow you to enable or disable these capabilities. The various `Drawing Limits` values specify the extents of your model. These limit values also determine the extents of AutoCAD's reference grid. AutoCAD displays a grid of reference dots according to the value of the `Grid Spacing` box. This reference grid helps you in the placement of objects in to the graphical model. `Snap Spacing` refers to the accuracy you wish for the cross-hairs in the graphical model to move. This feature allows you to perfectly align objects inside the graphical model. A future version of the DRIVE application may include the capability of generating column lines.

Initialize the AutoCAD window by selecting the `Window|AutoCAD Window` menu item. For this example, select `Feet` as the `Length Units`, use `-10` for both the `Lower X Limit` and `Lower Y Limit`, `175` as the `Upper X Limit`, `75` as the `Upper Y Limit`, `5` as the `Grid Spacing`, and `1` as the `Snap Spacing`. Click the `Load AutoCAD` button to complete the initialization and load AutoCAD in to the working memory of the system. Once this initialization is complete, selecting the `Window|AutoCAD Window` menu item switches focus to the AutoCAD application.

4.3 Building Story Parameters

You can input requirements for the building story parameters through the Concept Window. Display this window by choosing the `Window|Concept Window` menu item. Select the `Task|Show Building Stories Hierarchy` menu item to display the various building stories (Figure 4-2). The number of story objects displayed in

this window corresponds to the value of the attribute **Number Of Stories** of the **Building** object (Section 3.2.3). Click over **[unnamed story 1]** and choose `Rename Story` from the pop-up menu. Rename it to **First Floor**. Repeat the process to rename **[unnamed story 2]** to **Second Floor**.

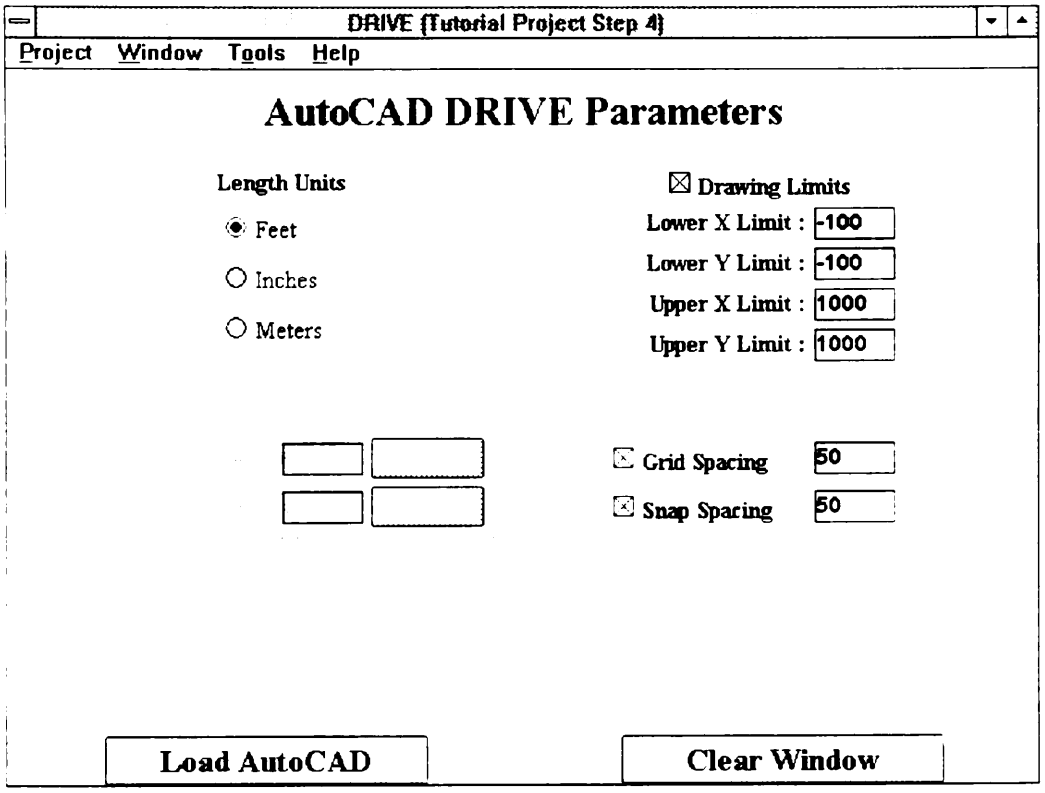


Figure 4-1. AutoCAD DRIVE Parameters

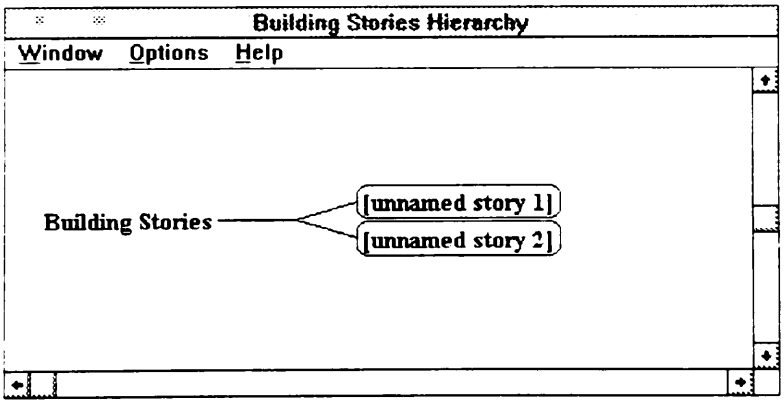


Figure 4-2. Building Stories Hierarchy

Customize the parameters of First Floor by choosing `Edit Requirements` from the pop-up menu. Figure 4-3 shows the Building Story Requirements Task Form.

This task form is a variation of the Building Requirements Task Form (Section 2.3). If you prefer, you can add your own text description in the space provided. Highlight the parameter **Floor Elevation** and click the `Edit Parameter` button. DRIVE then displays a new object-attribute window. Type in **0** as the value of this object-attribute and click the `Accept Changes` button.

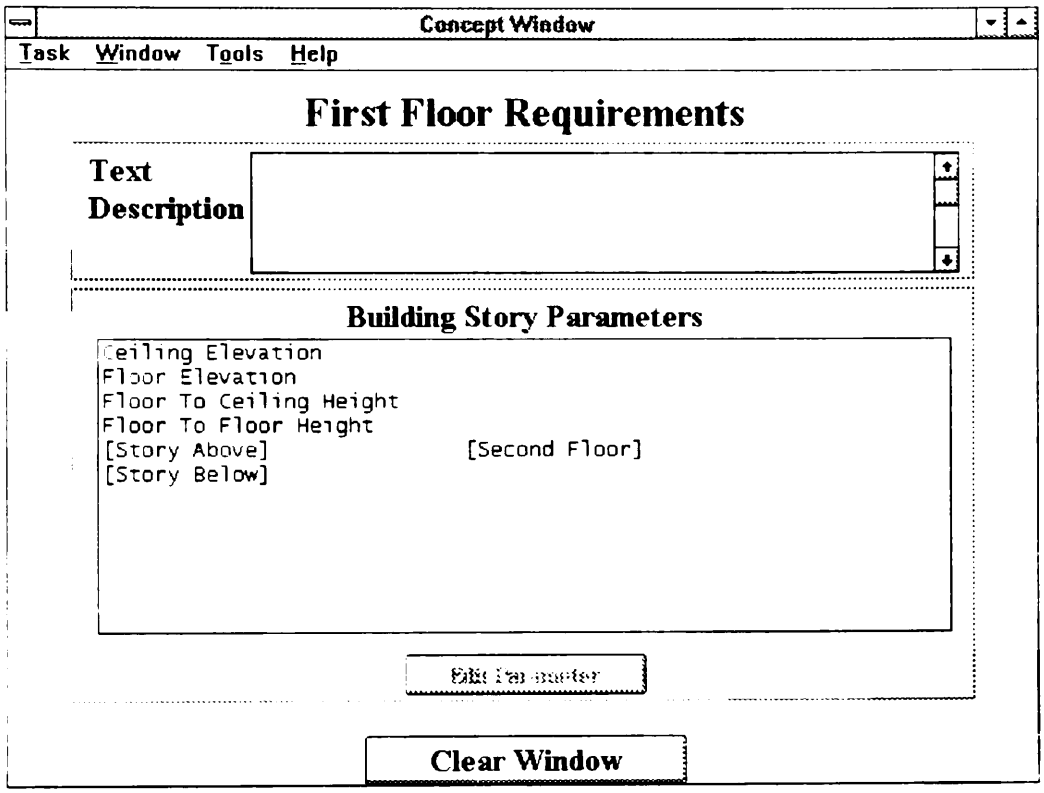


Figure 4-3. Building Story Requirements Task Form

Change the value of the attribute **Floor To Floor Height** of the object **First Floor** to **12**. This value represents a 12-foot difference between the **First Floor Elevation** and the **Second Floor Elevation**, since feet is the specified unit of measure. Click the `Accept Changes` button in the object-attribute window to update the system. Switch to the object **Second Floor** by selecting the `Task|Show Building Stories Hierarchy` menu item in the `Concept Window`, clicking over the **Second Floor** object and selecting `Edit Requirements` from the pop-up menu. Notice that DRIVE calculated the value of the attribute **Floor Elevation** of the object **Second Floor** from the **First Floor** attribute values. Finally, set the value of **Floor To Floor Height** of **Second Floor** to **10**.

The current version of DRIVE has limited capabilities regarding the manipulation of stories. The disabled items (`Delete Story`, `Add New Stories`, and `Insert New Stories`) in the Building Story Pop-up Menu attest to this fact. A future version of DRIVE may enable these disabled menu items.

4.4 Design Window

The `Concept Window` allows you to define the requirements of the attributes or parameters of the various objects in a building project. The `Design Window` allows you to specify the design values of these attributes. The value you set as a requirement for an attribute is separate from the design value of the same attribute. This separation allows the DRIVE system to check the design values against the project requirement values. A future version of DRIVE will have this data verification capability.

Display the `Design Window` by selecting the `Window|Design Window` menu item. From the `Task` menu of the `Design Window`, choose the `Show Building Project Hierarchy` item. DRIVE displays the entire Building Project hierarchy. This hierarchy contains all the objects for the current project. Click over `First Floor` and select `Edit Design Variables` from the pop-up menu. Notice that there are no values for the **Floor Elevation** and **Floor To Floor Height** parameters. This is because, previously, you were accessing the As Required values, and now you are accessing the As Designed values of the very same parameters. The procedure to enter As Designed values are similar to the procedure you used in entering the As Required values. For this example, set the **Floor Elevation** and **Floor To Floor Height** of **First Floor** to **0** and **12**, respectively. Also set the **Floor To Floor Height** of **Second Floor** to **10**. DRIVE uses these values to create the three-dimensional model object representations in AutoCAD.

4.5 Using AutoCAD DRIVE Extension

The AutoCAD DRIVE Extension adds more functionality to the already rich set of AutoCAD commands. The AutoCAD DRIVE Extension adds a DRIVE menu to the standard AutoCAD menus. This DRIVE menu contains allows you to create and manipulate graphical representations of the objects in a building project. DRIVE does not modify nor disable any of AutoCAD's built-in commands, so you still can use these AutoCAD commands. However, using some AutoCAD commands can cause inconsistencies between the graphical model and the object hierarchies. Do not use the `SAVE`, `SAVEAS`, `EXIT`, and `QUIT` commands in AutoCAD. Instead, use the Project menu in the DRIVE Main Window. The `Project|Save` menu item performs the saving operation on both the object hierarchies and the graphical model. Similarly, the `Project|Exit` performs the exit operation on both the DRIVE application and AutoCAD. Another set of AutoCAD commands that can cause inconsistencies is the "entity" duplication and modification commands (`COPY`, `MIRROR`, `ARRAY`, `SCALE`, `STRETCH`, etc.). Using these commands will either create graphical entities having no corresponding object in the object hierarchy or change the graphical representation of an object without a corresponding change in the properties of an object. Therefore, please exercise

care and judgment on using some of the AutoCAD commands. A future version of DRIVE may take into account all these limitations.

Figure 4-4 shows the layout of the first floor of the tutorial building project. The DRIVE menu in AutoCAD allows you to create AutoCAD Objects. AutoCAD Objects are objects that represent a real physical entity. Select the `DRIVE|Building Objects|Access Object Database|Architectural| Spaces` to display the Building Spaces Hierarchy. Currently, DRIVE can only create space and wall objects. Future versions of DRIVE will include more object types. Click over the `Library Spaces` text to display a pop-up menu. Select `Add AutoCAD Object` from the pop-up menu. Figure 4-6 shows the Add AutoCAD Objects Task Form. For this tutorial, type in `Reading Room` as the `New AutoCAD Object Name`. If you prefer, you may type in a short text description regarding the **Reading Room** object. Click the pull-down arrow corresponding to the `Topological Layout` and select `Rectangular Layout` from the list. The `Layout Parameters` box displays the various parameters corresponding to the selected layout. DRIVE also displays a diagram of these parameters. The procedure for placing values for these parameters is similar to what you already did for `Performance Parameters` in the Concept Window. The `Edit All Parameters` button allows you to type in values for all the various parameters without individually accessing these parameters through the `Edit Parameter` button. Click the `Edit All Parameters` button and type in `71` and `60` as the `a` and `b` parameters, respectively. The `Building Story` selection box changes the story level to create the new object. The `Performance Parameters` button allows access to the various non-graphical performance parameters of the object.

DRIVE only enables the `Create Object` button once the object name and all the layout parameters have values. Click the `Create Object` button to accept these parameters. DRIVE now switches the focus from the Design Window to the AutoCAD Window and asks for the corner position of the new object. Type in `92,5` as the corner coordinates. If you prefer using the mouse, position the AutoCAD cross-hairs to the point `92,5` and click the left mouse button. DRIVE now asks for the rotation angle of the new object. Type in `0` as the rotation angle. If you prefer using the mouse, position the cross-hairs such that there is no rotation angle imparted on the object and click the left mouse button. DRIVE now creates a new object based on the given parameters.

Clicking the right mouse button in the AutoCAD Window repeats the last command issued. If you have not issued another AutoCAD command since selecting the `DRIVE|Building Objects|Access Object Database| Architectural|Spaces` menu item, then clicking the right mouse button redisplays the Building Spaces Hierarchy. If clicking the right mouse button

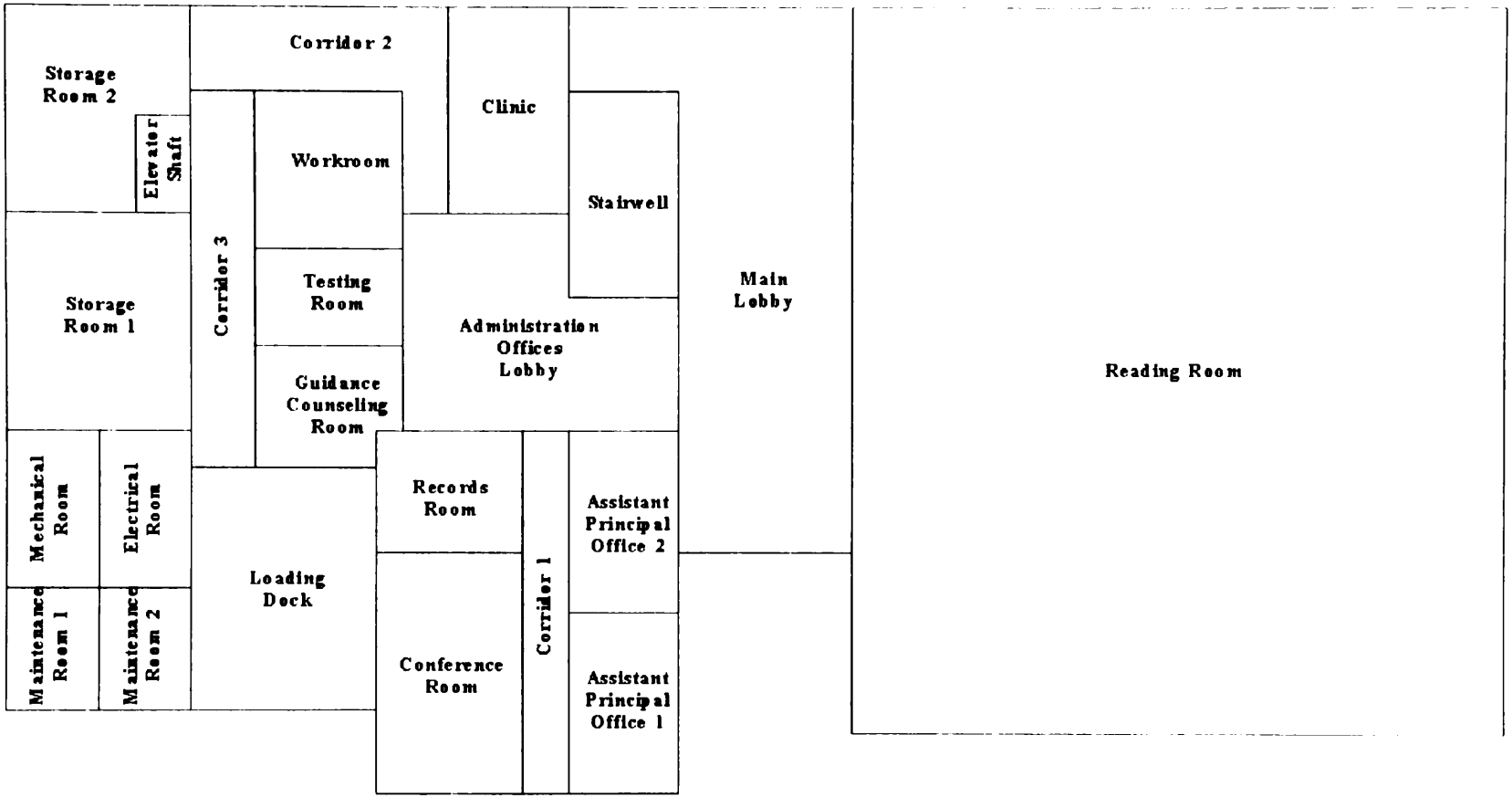


Figure 4-4. First Floor Layout

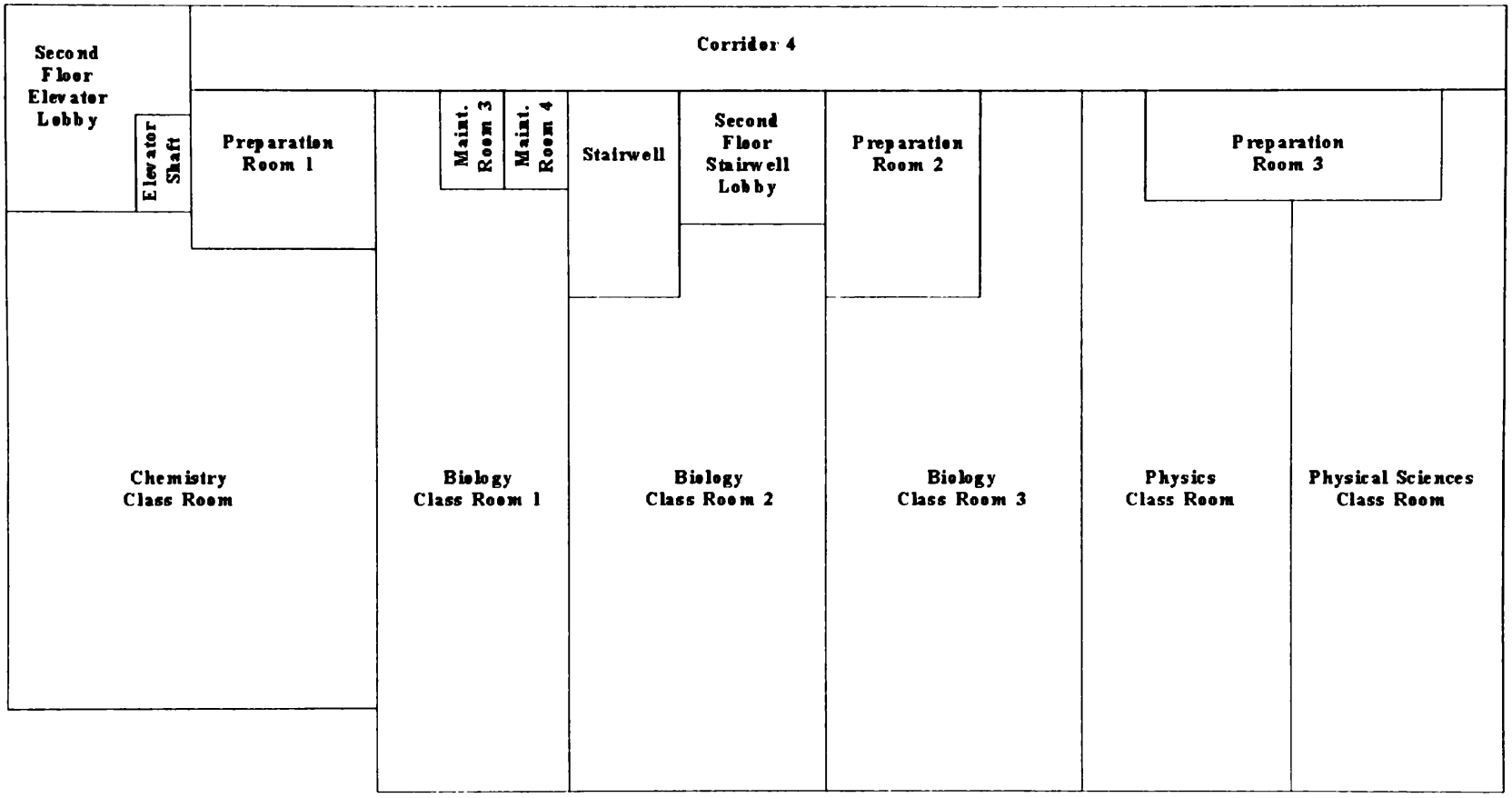


Figure 4-5. Second Floor Layout

invokes a different command, cancel that command by pressing `Ctrl-C` three times, and select the `DRIVE|Building Objects|Access Object Database|Architectural|Spaces` menu item. Click over the `Lobby Spaces` text and select `Add AutoCAD Object` from the pop-up menu. Type in `Main Lobby` as the object name. Set the `Topological Layout` to `L-Shaped Layout` and use the following parameters to create the object: `a` is `31`; `b` is `45`; `a1` is `19`; `b1` is `7`; corner coordinates are `92,65`; and rotation angle is `180`.

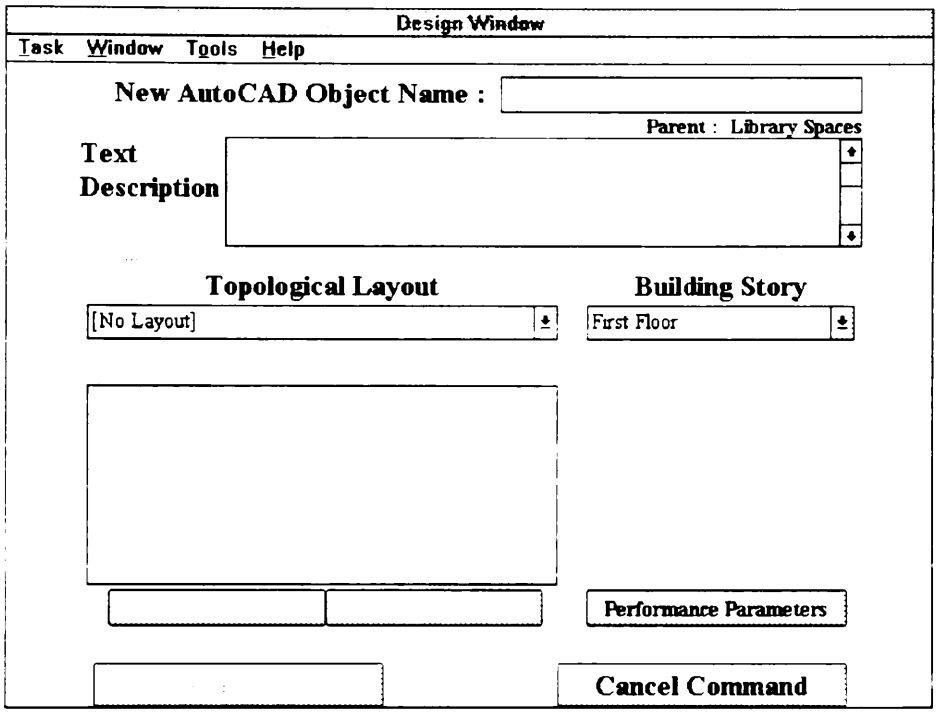


Figure 4-6. Add AutoCAD Objects Task Form

The two **Assistant Principal Offices** have similar graphical and non-graphical performance attributes. Rather than inputting similar data for these objects twice, it seems better to define the similar data once for a "parent" object and create "children" objects from this "parent" object. Access the Building Spaces Hierarchy, click over the `Office Spaces` text, and select the `Add SubClass` pop-up menu item. Type in `Assistant Principal Office Rooms` as the new sub-class title and hit the `Enter` key or click the `OK` button. Click over the newly created `Assistant Principal Office Rooms` text and select the `Edit Design Variables` pop-up menu item. Click the `Layout Parameters` button to define the graphical parameters for this sub-class. Use a `Rectangular Layout` with the following parameters: `a` is `12` and `b` is `15`. Click the `Accept Changes` button once to accept the changes made to the graphical parameters. Click again the `Accept Changes` button this time to accept any changes made to the non-graphical performance parameters.

Switch back to the AutoCAD Window and access the Building Spaces Hierarchy again. Click over the `Assistant Principal Office Rooms` text and select **Add AutoCAD Object** from the pop-up menu. Notice that the graphical parameters for the new object have values already. These values come from the values defined in the "parent" object. Thus, you only need to type in the name of the new object. Use `Assistant Principal Office 1` as the new object's name and click the `Create Object` button. Use `61,0` as the corner coordinates and `0` as the object rotation. Repeat the same process for `Assistant Principal Office 2` having corner coordinates `61,15` and `0` rotation.

A space can also have a different height than the height of its corresponding floor. One example is the **Stairwell**. Add an AutoCAD object named `Stairwell` to `Stairway Spaces`. Use a Rectangular Layout with the following parameters: `a` is 12, `b` is 17, `height` is 22, corner coordinates are `61,41` and rotation is `0`.

4.6 Object Manipulation Limitations

The object manipulation capabilities of DRIVE has numerous limitations. DRIVE is currently unable to modify the AutoCAD representation of an object if the user changes a graphical parameter. For example, you want to change the dimensions of a particular space that has an existing AutoCAD representation. DRIVE allows you to change its various graphical layout attributes parametrically. However, DRIVE does not change the AutoCAD representation to reflect the new layout attributes.

The inability of DRIVE to monitor any graphical change in an AutoCAD representation of an object is another limitation. AutoCAD has several existing commands and procedures that allow users to graphically modify an object. Thus, AutoCAD allows you to change the graphical dimensions of the AutoCAD representation of an object. However, various attributes of the object do not reflect these new graphical dimensions. Therefore, the responsibility of maintaining consistency between the graphical parameters of the DRIVE object hierarchy and the AutoCAD graphical model lies with the user.

Similar limitations exist for deleting objects. Both DRIVE and AutoCAD allow users to delete objects. However, deleting an object or object hierarchy in DRIVE does not delete the corresponding AutoCAD representation of these objects. Conversely, deleting an AutoCAD representation through the use of AutoCAD's ERASE command does not delete the corresponding object in the DRIVE object hierarchy. Therefore, it is once again left to the user to ensure consistency between the DRIVE object hierarchy and the AutoCAD graphical model when deleting objects.

4.7 Using Existing AutoCAD Drawings

DRIVE can also embed design rationale information onto existing two- or three-dimensional AutoCAD drawings. Initialize the AutoCAD DRIVE Extension using the procedure described in Section 4.2. Use the AutoCAD command **XREF** to link your existing AutoCAD drawings with the DRIVE model. Please consult the AutoCAD manuals regarding the subtleties of the AutoCAD command **XREF**.

The procedure for creating DRIVE objects from existing AutoCAD drawings is very similar to the procedure outlined in Section 4.5. The difference begins when you select **Pick AutoCAD Points** from the **Topological Layout** pull-down list (see Figure 4-6). Selecting **Pick AutoCAD Points** takes you to the AutoCAD window. The first value AutoCAD asks for is the height of the object you are creating. The default value is the height of the story you specified in the **Building Story** pull-down list. Press the **Enter** key to accept the default or type in the value of the height of the object. AutoCAD then asks for the bounding points for the object. Keep clicking the left mouse button on points inside the AutoCAD drawing or repeatedly use AutoCAD's snap capabilities (**ENDPoint**, **INTersection**, **MIDpoint**, **NEARest**, **PERpendicular**) to define the shape of the object. Please consult the AutoCAD manuals for information on these various snap capabilities. Use the **Close** command to instruct AutoCAD to create a line from the last point to the first point, thereby creating the object. The **Close** command only becomes available after defining at least three points for the object. **Pick AutoCAD Points** allows the definition of spaces having shapes other than the currently available topological layout shapes (Rectangular and L-shaped).

4.8 Completing the Building Spaces

The procedures in Section 4.5 describe how to create graphical representations of objects in the DRIVE system. Use this set of procedures to create all the remaining spaces in the building. Table 4-1 shows the graphical parameters for all the spaces on the **First Floor** including those detailed in Section 4.5. Table 4-1 includes information regarding the spaces detailed in Section 4.5 for completeness. Table 4-2 shows the parameters for all the **Second Floor** spaces. Please remember to use the **Add SubClass** instead of the **Add AutoCAD Object** pop-up menu item when creating sub-classes (denoted with an asterisk in the following two tables). Also, please change the default space height for objects spanning the entire height of the building (denoted with double asterisks). Spaces having no values for the **a1** and **b1** parameters have **Rectangular Layouts**, while those with values for these parameters have **L-Shaped Layouts**. Please remember to change the **Building Story** to **Second Floor** when you are creating spaces on the second floor. Choose **Project|Save** to save all the changes to the project after creating all the spaces in the building model. Finally, choose **Project|Close** to clear the working memory of the DRIVE application.

Table 4-1 First Floor Building Space Parameters

Space	Parent	a	b	a1	b1	Corner	Rotation
Reading Room	Library Spaces	71	60			92,5	0
Main Lobby	Lobby Spaces	31	45	19	7	92,65	180
Assistant Principal Office Rooms*	Office Spaces	12	15				
Assistant Principal Office 1	Assistant Principal Office Rooms	12	15			61,0	0
Assistant Principal Office 2	Assistant Principal Office Rooms	12	15			61,15	0
Stairwell**	Stairway Spaces	12	17			61,41	0
First Floor Maintenance Rooms*	Maintenance Room Spaces	10	10				
Maintenance Room 1	First Floor Maintenance Rooms	10	10			0,7	0
Maintenance Room 2	First Floor Maintenance Rooms	10	10			10,7	0
Loading Dock	Garage Spaces	20	20			20,7	0
Mechanical Room	Mechanical Spaces	10	13			0,17	0
Electrical Room	Electrical Spaces	10	13			10,17	0
Storage Room 1	Storage Room Spaces	20	18			0,30	0
Storage Room 2	Storage Room Spaces	17	20	9	14	0,65	-90
Elevator Shaft**	Elevator Spaces	6	8			14,48	0
Conference Room	Conference Room Spaces	16	20			40,0	0
Records Room	Storage Room Spaces	16	10			40,20	0
Guidance Counseling Room	Office Spaces	10	16	7	13	27,37	-90
Testing Room	Office Spaces	16	8			27,37	0
Workroom	Workshop Spaces	16	13			27,45	0
Clinic	Clinic Spaces	13	17			48,48	0
Administration Offices Lobby	Lobby Spaces	30	18	18	11	43,30	0
Corridor 1	Corridor Spaces	5	30			56,0	0
Corridor 2	Corridor Spaces	28	17	5	7	48,65	180
Corridor 3	Corridor Spaces	7	31			20,27	0

** sub-class* *** height is 22*

Table 4-2. Second Floor Building Space Parameters

Space	Parent	a	b	a1	b1	Corner	Rotation
Second Floor Maintenance Rooms*	Maintenance Room Spaces	7	8				
Maintenance Room 3	Second Floor Maintenance Rooms	7	8			47,50	0
Maintenance Room 4	Second Floor Maintenance Rooms	7	8			54,50	0
Chemistry Class Room	Class Room Spaces	40	41	20	38	0,7	0
Biology Class Room 1	Class Room Spaces	21	58	7	50	40,0	0
Biology Class Room 2	Class Room Spaces	47	28	41	16	89,0	90
Biology Class Room 3	Class Room Spaces	58	28	41	11	117,0	90
Physics Class Room	Class Room Spaces	23	58	7	49	117,0	0
Physical Science Class Room	Class Room Spaces	58	23	49	7	163,0	90
Corridor 4	Corridor Spaces	143	7			20,58	0
Preparation Room 1	Laboratory Spaces	20	13			20,45	0
Preparation Room 2	Laboratory Spaces	17	17			89,41	0
Preparation Room 3	Laboratory Spaces	32	9			124,49	0
Second Floor Elevator Lobby	Lobby Spaces	17	20	9	14	0,65	-90
Second Floor Stairway Lobby	Lobby Spaces	16	11			73,47	0

* *sub-class*

Chapter 5
Cost Breakdown Diagrams

Chapter 5 discusses DRIVE's cost breakdown module. In this chapter, you will learn how to develop a life cycle cost budget as well as a preliminary life cycle cost estimate for the project. This chapter also explains how you can use the cost breakdown diagrams as a navigation tool to aid in developing a budget or a preliminary estimate.

5.1 Cost Module Limitations

Value engineering aims to provide alternatives with lower life-cycle cost without sacrificing the functionality of a design. Life-cycle costs include all the costs associated with the various phases of a project (e.g., design costs, construction costs, operation costs, maintenance costs, and demolition costs). As such, a computer tool to support value engineering should be able to conduct life-cycle cost estimating. DRIVE provides a limited life cycle cost estimating capability. DRIVE handles design and construction costs as initial costs. DRIVE also has two types of operation and maintenance costs, namely, annual and single year. Annual costs occur yearly for the entire economic life period of the building. Single year costs occur only once during the entire building life. A major limitation of DRIVE's life cycle costing module is its inability to present graphically the life cycle cost breakdown according to the various life cycle stages.

The inability of DRIVE's cost module to specify a construction cost estimating parameter for the entire building is another limitation. An example of this limitation is that users cannot specify the parameter **Construction Cost Per Gross Area** at the building level. Rather, users can only specify this parameter at the space type level. Section **5.5** further illustrates this limitation. Future versions of DRIVE may take into account the above limitations.

5.2 Life Cycle Cost Budgeting - Space Types

Open the project you created in Chapter 4 or `Tutorial Chapter 5`. If you are using `Tutorial Chapter 5`, please copy the project into a new project. See Section 3.1 for the procedure for copying projects.

This section shows how to assign life cycle cost budgets to the various space types in the building. A life cycle cost budget is nothing more than the **As Required** value of the attribute **Life Cycle Cost** of the various space objects. As such, only the `Concept Window` allows access to these values. Switch to the `Concept Window` and select `Task|Life Cycle Budget Allocation` from its menu bar. DRIVE then displays the entire object hierarchy for the project. Clicking over one of the

text identifiers in the object hierarchy displays the Cost Budgeting Menu (Figure 5-1). Click over the text identifier for the **Building** object and select `Estimate Costs` from the pop-up menu.

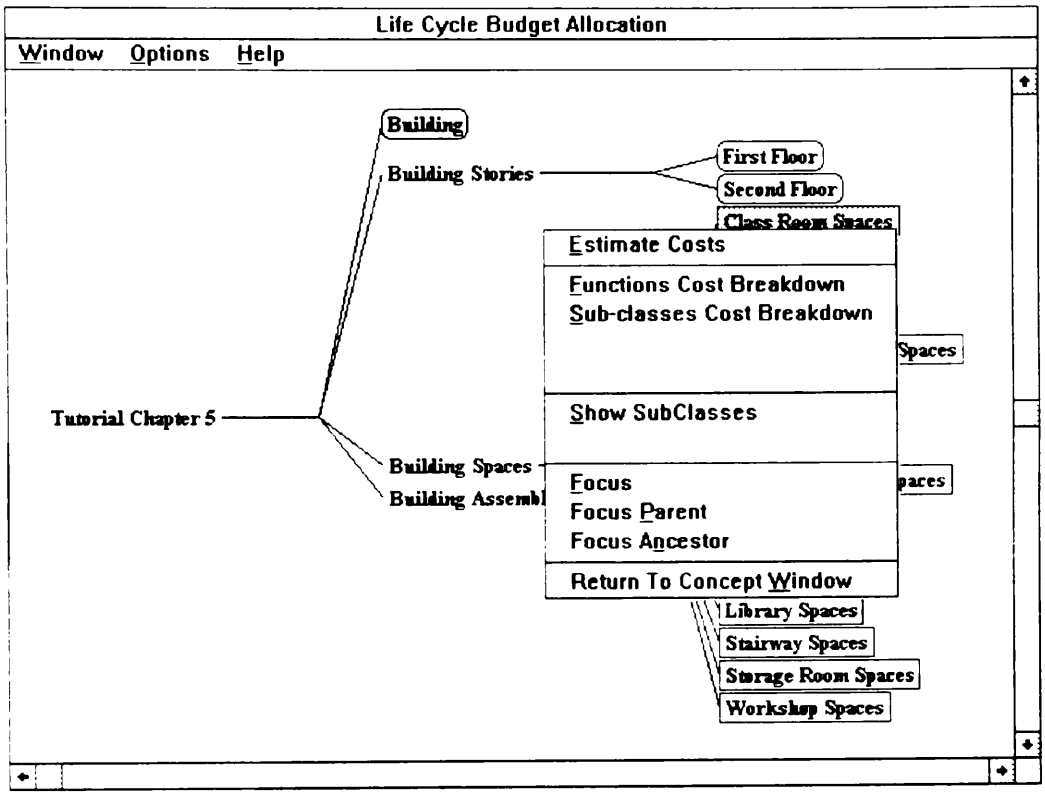


Figure 5-1. Cost Budgeting Menu

Figure 5-2 shows the Life Cycle Cost Budgeting Form. This form allows users to enter budgeted values for the various life cycle phases of a design object. For the **Building** object shown in Figure 5-2, use the `Allocation Parameters` list box and the `Edit Parameter` button to set values of the `Construction Cost` and `Design Cost` parameters to `1500000` and `45000`, respectively. Notice that the system does not calculate the value of the `Total Present Value Life Cycle Cost` parameters from the other parameters. This is to give freedom to the users to specify values for the various parameters that are not dependent on each other. However, only the `Concept Window` allows this flexibility. The `Design Window`, on the other hand, calculates the `Total Present Value Life Cycle Cost` parameters from the other parameters as described in Section **5.5**. For this tutorial, set the value of the `Total Present Value Life Cycle Cost` parameter to `1545000`.

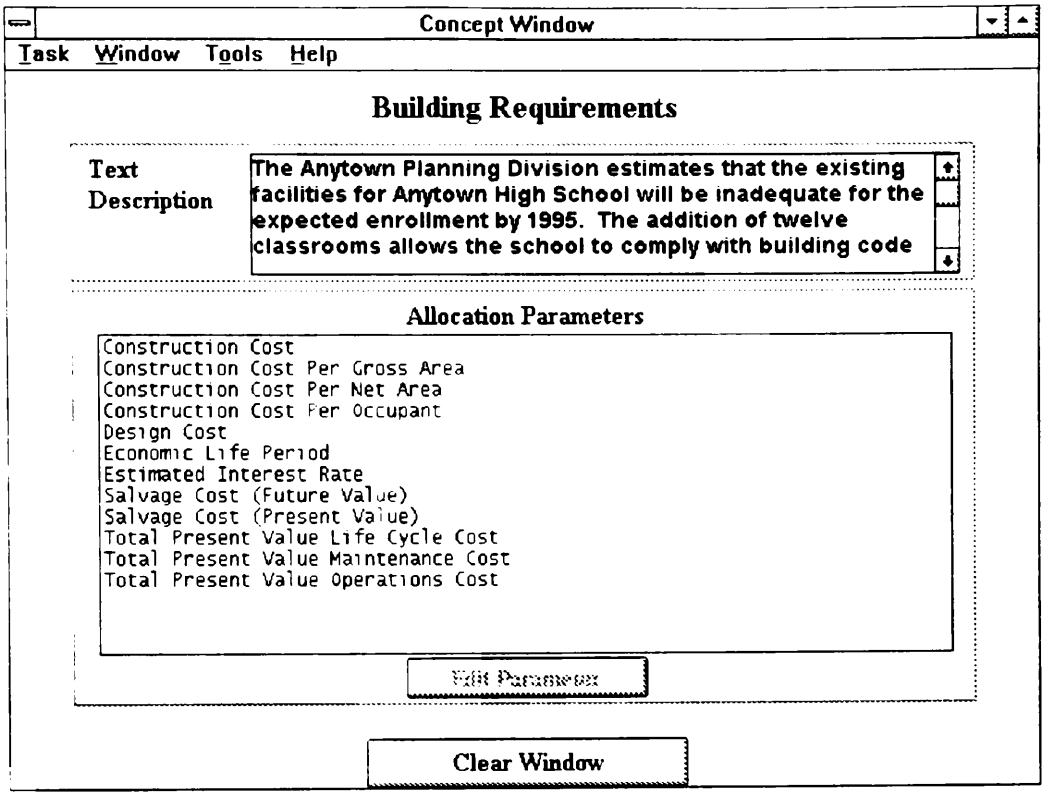


Figure 5-2 Life Cycle Cost Budgeting Form

Figure 5-3 shows the Cost Breakdown Color Table. This color table changes the current color assignments of the various breakdown categories, in this particular case, space types. Click over the pull-down arrow corresponding to any space type. The pop-up list shows all the breakdown categories that are the space types in the building. This list also contains a <none> choice to exclude a particular color from the breakdown diagram. The color table is useful in cases where a particular color must represent a specific breakdown category. The check box below the color table forces the system to always use its default color assignment scheme for the various breakdown categories. Click over this check box such that there is an x mark inside the small square. Having the system assign the color assignments is useful in cases where there is no particular need to customize these assignments. Click the Show Cost Breakdown button at the bottom of the Concept Window to accept the color table assignments and bring up the Cost Breakdown Task Form (Figure 5-4).

Use the slider bar or the edit box associated with **Class Room Spaces** to set its budgeted amount to 550000. Notice how the cost breakdown pie chart changes to reflect this value. Next, set the budget for the **Corridor Spaces** to 60000. The color of the slider bars and the cost pie sections correspond to the colors specified in the color table. Table 5-1 shows a possible set of construction budgets for the

various space types. If you feel that the budgeted values shown in Table 5-1 are incorrect, please modify them as necessary when entering the budget data into the system. Since there are more than 12 space types for this building, then some of these space types are initially inaccessible. For space types that are not already accessible (e.g., **Library Spaces**), use the `other categories` button at the bottom of the `Concept Window`. Clicking this button allows access to the other breakdown categories. Click the `other categories` button and select `Library Spaces` from the pop-up menu. DRIVE then displays a new pop-up menu asking for the new location of **Library Spaces**. Click over the pop-up menu item `Corridor Spaces`. Notice that **Library Spaces** now occupies the position and color previously assigned to **Corridor Spaces**. Notice also that the cost pie chart now contains a slice with a black color. The black pie slice corresponds to the total amount allocated to the various breakdown categories that have no assigned color as shown in the Cost Breakdown Task Form.

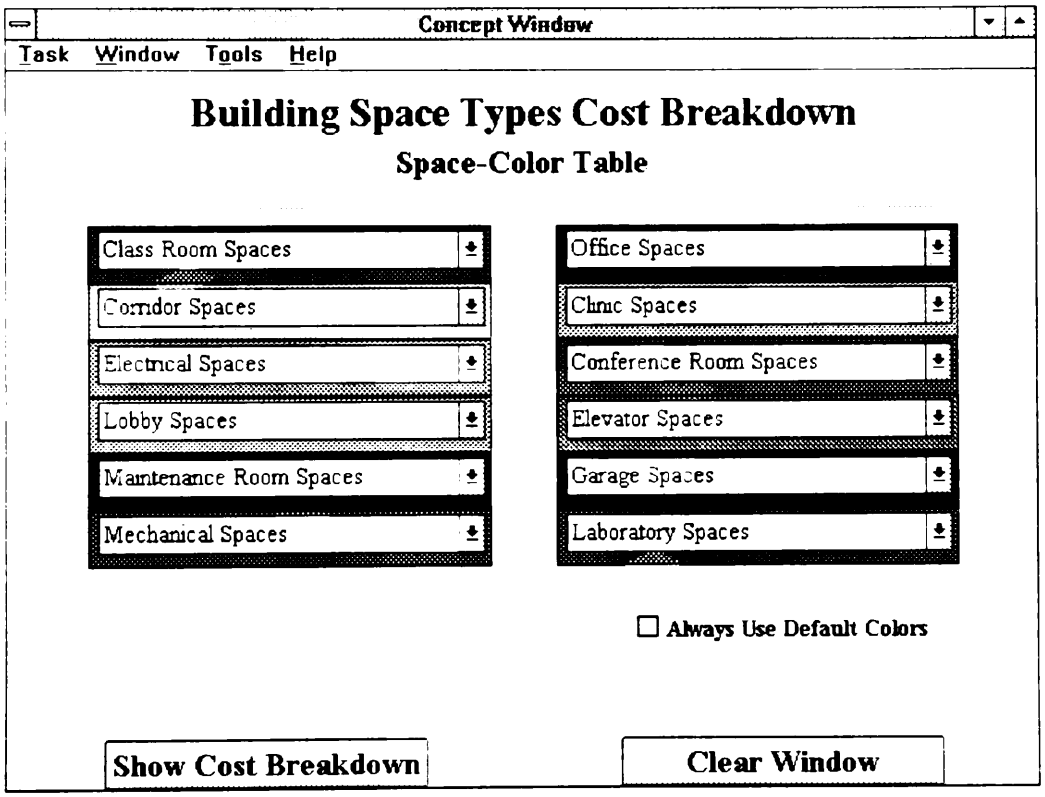


Figure 5-3. Cost Breakdown Color Table

5.3 Using the Breakdown Pie Chart

The cost breakdown pie chart not only depicts the relative proportions of the different breakdown categories but is also a navigation tool to aid in developing cost breakdowns. Move the mouse cursor over the various pie slice portions. Notice how the text above the pie chart changes to reflect the current breakdown

category. If **Office Spaces** is not a currently accessible breakdown category (i.e., grouped under `other categories`), click over the `other categories` button, select `Office Spaces` from the pop-up menu, and choose a new display position for **Office Spaces**. Click over the pie slice corresponding to **Office Spaces**.

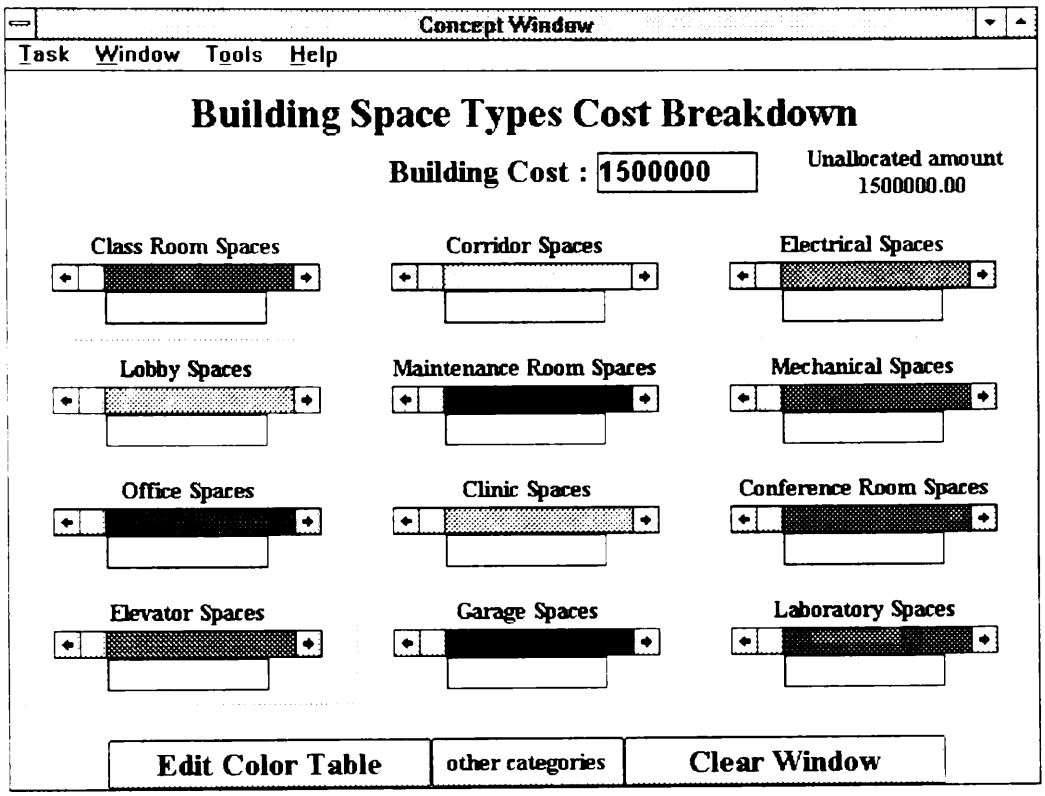


Figure 5-4. Cost Breakdown Task Form

Figure 5-5 shows the Cost Breakdown Navigation Menu. If a breakdown category has further sub-categories, then this pop-up menu allows quick access to these sub-categories. For example, the Office Spaces object has two kinds of sub-categories namely a sub-class breakdown and a function breakdown. Select the pop-up menu item `Office Spaces SubClass Breakdown` to show the breakdown task form for the sub-classes and AutoCAD instances of **Office Spaces**.

Assign arbitrary values for these sub-categories of **Office Spaces**. Click over the various pie slices corresponding to these sub-categories. Notice that only the **Assistant Principal Office Rooms** category has further space type sub-categories. Move the mouse cursor to a point outside the circular pie but still inside the square enclosing the pie chart. Notice that the text over the pie chart changes to show **Building**. Click over this area and select `Building Spaces SubClass Breakdown` from the pop-up menu. The system then returns to the earlier pie chart. The area outside the pie chart switches to the next higher level just as any pie slice switches

to the next lower level. Please attempt to assign arbitrary construction budget values to the various space objects in the building.

Table 5-1. Construction Budgets for Space Types

Space Type	Budget	Space Type	Budget
Class Room Spaces	$540,000	Library Spaces	$325,000
Clinic Spaces	$30,000	Lobby Spaces	$100,000
Conference Room Spaces	$40,000	Maintenance Room Spaces	$10,000
Corridor Spaces	$60,000	Mechanical Spaces	$60,000
Electrical Spaces	$30,000	Office Spaces	$80,000
Elevator Spaces	$50,000	Stairway Spaces	$10,000
Garage Spaces	$15,000	Storage Room Spaces	$50,000
Laboratory Spaces	$90,000	Workshop Spaces	$10,000

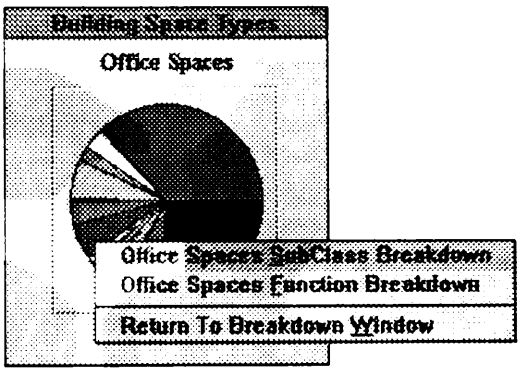


Figure 5-5 Cost Breakdown Navigation Menu

The white pie slice in these cost breakdown diagrams represents unallocated amounts. Clicking over a white pie slice displays a message showing the unallocated amount. A black pie slice represents the total of all the breakdown categories relegated to the other categories group. DRIVE is currently only able to assign colors to twelve categories. If there are more than twelve categories, then the `other categories` group takes the extra categories. Clicking over a black pie slice displays a message showing the amounts allocated to the various categories lumped together in the `other categories` group. Another feature of the cost breakdown diagrams is the over-allocation warning message. If the sum of all the breakdown categories for a particular item exceeds the budget for that item, then the breakdown diagram displays a warning message showing the percentage amount of over-allocation.

5.4 Construction Budgeting - Functions

Assigning construction budgets for the various functions of the building and its spaces is similar to the procedure to assign construction budgets to the various space types. Select the `Task|Life Cycle Budget Allocation` from the `Concept Window` menu bar. Click over the text identifier `Building` and select `Functions Cost Breakdown` from the pop-up menu. Instead of space types, the breakdown categories are the various particular functions of the building. Table 5-2 shows the budgets for the various particular functions of the building. Enter these values to generate a cost breakdown pie chart.

Table 5-2 Construction Budgets for Functions

Function	Budget	Function	Budget
Teach Students	$1,035,000	Present Lectures	$540,000
		Administrate School	$40,000
		Counsel Students	$40,000
		Prepare Experiments	$90,000
		Study Lessons	$325,000
Assure Convenience	$290,000	Facilitate Circulation	$220,000
		Treat Injuries	$30,000
		Facilitate Meetings	$40,000
Provide Environment	$175,000	Supply Electricity	$30,000
		Tidy Surroundings	$5,000
		Maintain Facility	$30,000
		Maintain Temperature	$30,000
		Supply Water	$15,000
		Transport Supplies	$15,000
		Store Supplies	$50,000

Clicking over one of the pie slices corresponding to a function displays a pop-up menu. This menu allows access to the sub-functions associated with the selected function. Click over the `Teach Students` pie slice and select the `Teach Students Sub-Function Breakdown` pop-up menu item. DRIVE then displays a number of sub-functions associated with the **Teach Students** function. The **Particular Functions** attribute of space type objects having a **General Function** attribute of **Teach Students** form this list of sub-functions. Enter the appropriate values shown in Table 5-2 for the sub-functions of **Teach Students** function. Click outside the pie chart but inside its bounding square and select `Building Function Breakdown` to return to the previous breakdown diagram. Repeat this procedure for the **Assure Convenience** and **Provide Environment** functions.

5.5 Preliminary Cost Estimating

DRIVE's Design Window can perform a preliminary construction estimate. This estimate uses a historical unit area construction cost for similar buildings. Given a unit area construction cost (dollars per square foot), DRIVE takes the areas of the various AutoCAD space objects and multiplies these area values with the unit area cost to come up with a preliminary estimate. Typical estimating handbooks, such as Means, contain these unit area costs for various building types. The unit area cost values contained in these handbooks are only for the entire building. DRIVE currently cannot specify a unit area cost at the building level. However, DRIVE can specify unit area costs at the space type level. Thus, one way to make use of the existing data contained in estimating handbooks is to enter one building unit area cost for all the space types. Collecting data for unit area costs at the space type level allows users to specify different unit area costs for each particular space type. This ultimately leads to a significant improvement to the preliminary construction estimating capability of DRIVE.

Switch to the Design Window to perform a preliminary construction estimate. Select `Task|Preliminary Cost Estimates` from the menu bar. Click over the `Building` text identifier. Select `Space Types Cost Breakdown` from the pop-up menu. Unless you prefer not to use DRIVE's default color assignment scheme, switch on the `Always Use Default Colors` check box. Click the `Show Cost Breakdown` button. The Design Window is now something similar to Figure 5-4. However, the big difference between this window and Figure 5-4 is that this new window lost all its cost editing capabilities. This is because the cost values at this level are aggregations of the various cost estimates of the various AutoCAD space objects.

Select `Task|Show Building Project Hierarchy` from the menu bar of the Design Window. Click over the `Office Spaces` text identifier and select `Edit Design Variables` from the pop-up menu. Edit the parameter `Construction Cost Per Gross Area for AutoCAD Objects`. Assuming a unit area construction cost of $60 per square foot for the entire building, type in `60` as the value of this parameter for **Office Spaces**. Click the `Accept Changes` button to formally submit this value to the DRIVE system for processing. Notice that the parameters **Construction Cost Per Gross Area** and **Construction Cost Per Gross Area For AutoCAD Objects** now have the value of **60** and that the parameter **Construction Cost** also has a value. The value for **Construction Cost** comes from the values of **Construction Cost** of the various AutoCAD objects in the **Office Spaces** hierarchy.

The concept of aggregation of sub-class costs becomes clearer by using the cost breakdown diagrams. Select the `Task|Preliminary Cost Estimates` menu item,

click over the `Building` text identifier, select the `Space Types Cost Breakdown` menu item, click over the pie slice corresponding to Office Spaces, and select the `Office Spaces SubClass Breakdown`. Figure 5-5 shows a situation where some but not all cost editing interface objects are inoperable.

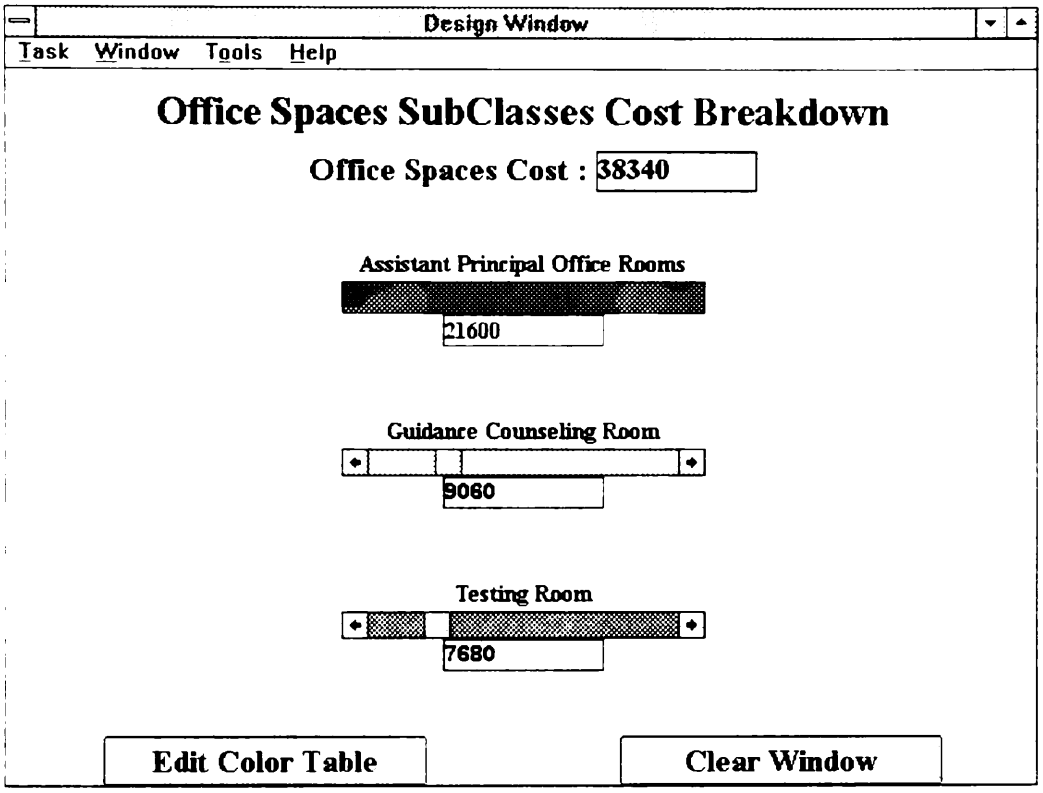


Figure 5-5 Design Window Cost Breakdown

In the Design Window, only AutoCAD Objects have their cost values changeable. The value for **Construction Cost Per Gross Area for AutoCAD Objects** and the gross area of the object taken from the AutoCAD model determined the value for the **Construction Cost** of this object. Taking the **Testing Room** as an example, this object has a gross area of 128 square feet. At $60 per square foot, **Testing Room** has a **Construction Cost** of $7680. A similar procedure determined the **Construction Cost** values for the other AutoCAD Objects in the **Office Spaces** hierarchy. Having no source for unit area construction costs for space types, use $60 per square foot as the unit area costs for all space types.

The Design Window also limits the parameters that users can directly modify. **Construction Cost Per Gross Area for AutoCAD Objects** is an example of a directly changeable parameter. **Total Present Value Life Cycle Cost** is an example of a derived parameter. Derived parameters are not directly modifiable by the user. The system calculates these parameters from the other parameters the

user can modify. To illustrate, select the **Task|Preliminary Cost Estimates** menu item, click over the **Building** text identifier, and select the **Estimate Costs** pop-up menu item. The Design Window shows an interface similar to the one shown in Figure 5-2. Use the **Allocation Parameters** list box and the **Edit Parameter** button to set values of the **Economic Life Period** and **Estimated Interest Rate** parameters to **50** and **0.10**, respectively. These values correspond to a 50 year building life and a 10 percent interest rate. Select again the **Task|Preliminary Cost Estimates** menu item, click over the **Office Spaces** text identifier, and select the **Estimate Costs** pop-up menu item. Assign **General Cleaning** as the value of the **Maintenance (Annual A) Description** parameter. The estimated cost of the annual cleaning maintenance on all office spaces is $1000. Thus, use the **Allocation Parameters** list box and the **Edit Parameter** button to set value of the **Maintenance (Annual A)** parameter to **1000**. Scroll down the **Allocation Parameters** list box and look at the life cycle parameters. The system calculated the values for these parameters from the user specified information on the economic life period, interest rate, and annual costs.

Save the current project after completing the unit area costs for all space types. Finally, close the current project to clear the working memory of the DRIVE application.

Chapter 6
Processing Pending Rationale

Chapter 6 discusses the pending rationale module. In this chapter, you will learn how to process any postponed rationale, as well as how to add and delete performances of various objects. This chapter also reviews the rationale capture concepts.

6.1 Processing Pending Rationale

Open the project you created in Chapter 5 or `Tutorial Chapter 6`. If you are using `Tutorial Chapter 6`, please copy the project into a new project. See Section 3.1 for the procedure for copying projects.

Chapter 3 showed the capturing and postponing mechanisms of the rationale capture module. DRIVE allows the resumption of postponed or pending rationale. Selecting the `Tools|Rationale Capture|Process Pending Rationale` displays the Pending Rationale Task Form (Figure 6-1) in the DRIVE Main Window. Clicking over one or more lines in the `Pending Rationale` list selects which object-attributes to resume rationale capture. This alphabetically arranged list contains the various object-attributes that have a value defined but have not yet fully completed its rationale definition process.

Select `[Building]:[Construction Cost] (maximum value)` and `[Building]:[Number Of Stories] (value)` from the list. Click the `Process Rationale` button. DRIVE displays the object-attribute and rationale capture windows for both these selections. DRIVE also removes these items from the `Pending Rationale` list. Please refer to Sections 2.7.3 and 3.3 for a more thorough description of the object-attribute and rationale capture windows, respectively.

Display the rationale capture window for the attribute **Construction Cost** of the object **Building** by selecting `Window|RationaleWindows|[Building]:[Construction Cost] (maximum value)`. Type in `Owner's Construction Budget Description` as the name identifier for this rationale. Click the `Continue Rationale Capture` button in this rationale window. DRIVE displays a new rationale window asking for the dependency type for this particular rationale. Select `owner requirements` and click the `Continue Rationale Capture` button. DRIVE now displays a new rationale window asking for a text description. For illustration purposes, click the `Postpone Rationale Capture` button in this new rationale window. DRIVE places this newly postponed rationale back into the `Pending Rationale` list. Switch back to the Pending Rationale Task Form to

verify that DRIVE placed this rationale back into the list. Note that DRIVE now uses the name identifier you typed in for this rationale.

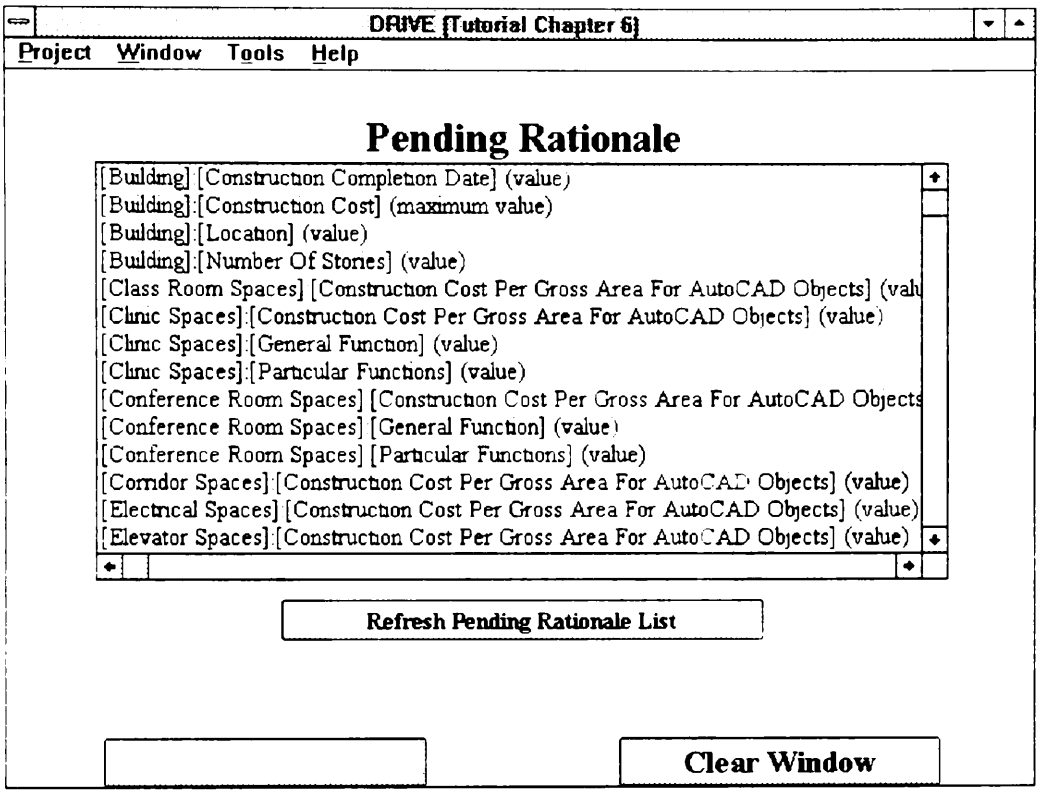


Figure 6-1. Pending Rationale Task Form

Resume the rationale capture process for `Owner's Construction Budget Description` by selecting it from the list and clicking the `Process Rationale` button. DRIVE now resumes the rationale capture process where it was last postponed, namely, in the `Rationale Comment Window`. You can type your own comment or use the following as the text description for this rationale:

> *The town council approved a $1.5 million budget for the new school building.*

Complete the rationale capture process by clicking the `Conclude Rationale Capture` button. Finally, click the `Close Window` button on the object-attribute window corresponding to `[Building]:[Construction Cost]`.

Switch to the rationale window corresponding to `[Building]:[Location]`. Type in `Building Location Owner Requirement` as the name identifier for this rationale. Continue the rationale capture process and select `owner requirements` as the dependency of this rationale. Complete the rationale capture process using the following rationale comment:

The new building will occupy the vacant lot adjacent to the existing high school building.

Finally, click the `Close Window` button on the object-attribute window corresponding to `[Building]:[Location]`.

6.2 Adding and Deleting Performances

The attributes of the various objects depend upon its defined performance types. Any task form containing the Performance Types Control Box (Figure 6-2) allows the addition or the deletion of performance types for an object. Switch to the `Concept Window` and select `Task|Show Building Spaces Hierarchy` from its menu bar. Click over `Class Room Spaces` and select `Edit Requirements` from the pop-up menu. The Edit Space Requirements Task Form contains a Performance Types Control Box.

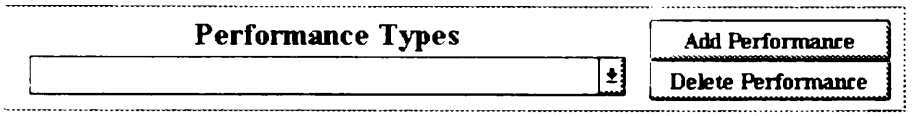


Figure 6-2 Performance Types Control Box

Click the `Add Performances` button to show the Performance Types Task Form (Figure 6-3). This task form performs in a manner similar to the Building Space Types Task Form (Figure 2-5). The arrow pointing to the `Current Performances` list box adds performance types to an object while the other arrow deletes them. For the Class Room Spaces Object, highlight the `Acoustical Performance`, `Air Purity Performance`, and `Visual Performance` in the `Other Performances` list and click the arrow pointing to the `Current Performances` list. Click the `Accept Changes` button to add these performance types to the **Class Room Spaces** object. Click the pull-down arrow corresponding to the `Performance Types` box. The pull-down list now contains also the new performances just added.

There are two methods for deleting performance types for the various objects. One method uses the Performance Types Task Form. When there is no performance selected in the `Performance Type` box, clicking the `Delete Performance` button brings up this task form. To use this task form, highlight the current performances you wish to remove from the `Current Performances` list box, and then click the arrow button pointing away from this list box. The other method for removing performance types involves selecting a current performance in the `Performance Type` box and then clicking the `Delete Performance` button. This removes the currently selected performance from the object. The first method is suitable when there are numerous performance types to remove from an object. Conversely, the latter method is suitable when there is only one performance type to remove.

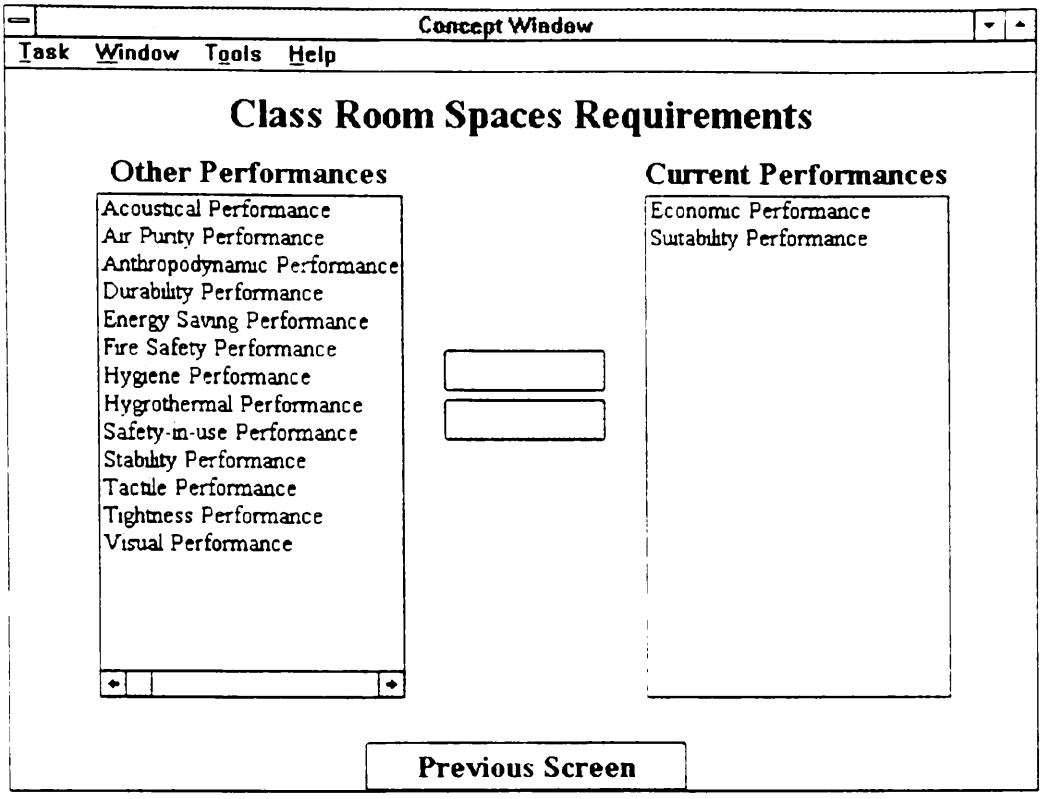


Figure 6-3. Performance Types Task Form

6.3 Rationale Capture Revisited

This section uses the concepts described in Chapter 3 regarding the various modes of rationale capture and the various rationale windows. This section does not explicitly state specific keyboard and mouse actions. Rather, it states only a general description of tasks. If the need arises, please refer to Chapter 3 for a description on how to perform a particular task.

Switch to the `Fully Automatic` rationale capture mode. Edit the requirements for **Class Room Spaces**. Search for the `Performance Type` having the `Illuminance` parameter. Set the value and minimum value of the attribute **Illuminance** of the object **Class Room Spaces** to `500` and `11`, respectively.

Name the rationale for the value of `[Class Room Spaces]:[Illuminance]` as `Illuminance - Function Relationship`. This rationale depends solely on the attribute **Particular Functions** of the object **Class Room Spaces**. The illuminance-function relationship is as follows:

If [Class Room Spaces]:[Particular Functions] is equal to Present Lectures

Then [Class Room Spaces]:[Illuminance] is equal to 500.

Use the following as the comment for this rationale:

Presenting lectures qualifies as a illuminance category D which requires a 500 lux illuminance according to the Illuminating Engineering Society of North America.

Use `Illuminance Minimum Value Requirement` as the name identifier for the minimum value rationale of `[Class Room Spaces]:[Illuminance]`. This rationale depends on a code requirement with the following comment:

The minimum value for illuminance of Class Room Spaces is 11 lux based on the safety considerations of ANSI/IES RP3-1988.

Switch the rationale capture module `Off` and bring up the Pending Rationale Task Form. Resume the rationale capture process for `[Building]:[Construction Completion Date] (value)`. Name this rationale as `Construction Completion Requirement`. This rationale depends on an owner requirement with the following comment:

The new building needs to be functional at the start of the 1995 academic year.

6.4 Completing Building Space Illuminance Values

Add the **Acoustical**, **Air Purity**, and **Visual Performances** to all the various space types. Set the required and minimum values of the illuminance for these space types as specified in Table 6-1. Please feel free to modify any of the values shown in Table 6-1 if the need arises.

Close up all object-attribute and rationale windows. Save the current project after closing all these windows. Finally, close the current project to clear the working memory of the DRIVE application.

Table 6-1. Illuminance Values (lux) for Various Space Types

Space Type	Required Value	Minimum Value
Class Room Spaces	500	11
Clinic Spaces	500	11
Conference Room Spaces	300	11
Corridor Spaces	150	11
Electrical Spaces	150	5.4
Elevator Spaces	150	5.4
Garage Spaces	200	11
Laboratory Spaces	500	5.4
Library Spaces	750	22
Lobby Spaces	200	22
Maintenance Room Spaces	150	5.4
Mechanical Spaces	150	5.4
Office Spaces	300	11
Stairway Spaces	150	22
Storage Room Spaces	150	5.4
Workshop Spaces	500	5.4

Chapter 7
Value Engineering Module

Chapter 7 discusses the value engineering module. In this chapter, you will learn how to use DRIVE to support the value engineering process. This chapter explains how to retrieve the rationale captured for the various object-attributes, as well as how to traverse the relationships existing between these object-attributes. This chapter also discusses how to perform queries on the AutoCAD model

7.1 Value Engineering Overview

The US Code of Federal Regulations defines value engineering (VE) as "an organized effort to analyze the functions of systems, equipment, facilities, services, and supplies for the purpose of achieving the lowest life-cycle cost consistent with required performance, reliability, quality, and safety". Although the same individuals who performed the design of a system can conduct the VE study, the usual case is that other persons, specializing in VE, work together with the above mentioned individuals in conducting VE studies.

There are many variations of value engineering phases. One variation divides value engineering into the following six phases:

- Information Gathering - gathering the details of the design as well as the reasoning leading to the creation of this design
- Speculation - brain-storming for VE suggestions to reduce the life-cycle cost of a system without sacrificing functionality
- Analysis - selection of more feasible VE suggestions for further analysis and development
- Development - further examination and design development of VE suggestions
- Presentation - presentation of feasible VE suggestions to the owner for final approval regarding implementation
- Post-Occupancy Evaluation - checking whether the implemented VE suggestions actually resulted in life-cycle cost savings.

DRIVE concentrates on the Information Gathering phase of a VE study. Since not all the individuals in a VE study team are part of the design development team, these individuals must gather both the details and reasoning leading to the creation of a design. DRIVE assists in this process by allowing designers to store the rationale behind their design together with the computer model of the design. Value engineers can then use DRIVE to retrieve the captured rationale to help them understand the design. The captured design rationale allows the value engineers to come better prepared to the VE study meetings, therefore enhancing

the Information phase by allowing the study team to spend more time discussing design issues in depth, instead of spending valuable meeting time describing the design.

7.2 Retrieving Rationale Information

Open the project you created in Chapter 6 or `Tutorial Chapter 7`. If you are using `Tutorial Chapter 7`, please copy the project into a new project. See Section 3.1 for the procedure for copying projects.

One kind of rationale information captured by the DRIVE application is the textual descriptions of the various model objects. Display the Value Engineering Window by selecting the `Window|Value Engineering Window` menu item. Select the `Task|Access Object Network` menu item from the Value Engineering Window. DRIVE displays the entire object network for the current project. Search for and click over the text identifier `Building`. DRIVE displays a menu containing the allowable operations for this object. Select the `Show Description` item from this pop-up menu. Figure 7-1 shows the Object Description Task Form. The text displayed in this task form corresponds to the text typed into the Text Description box for a particular object. Please refer to section 2.3 for a detailed explanation of the Text Description box.

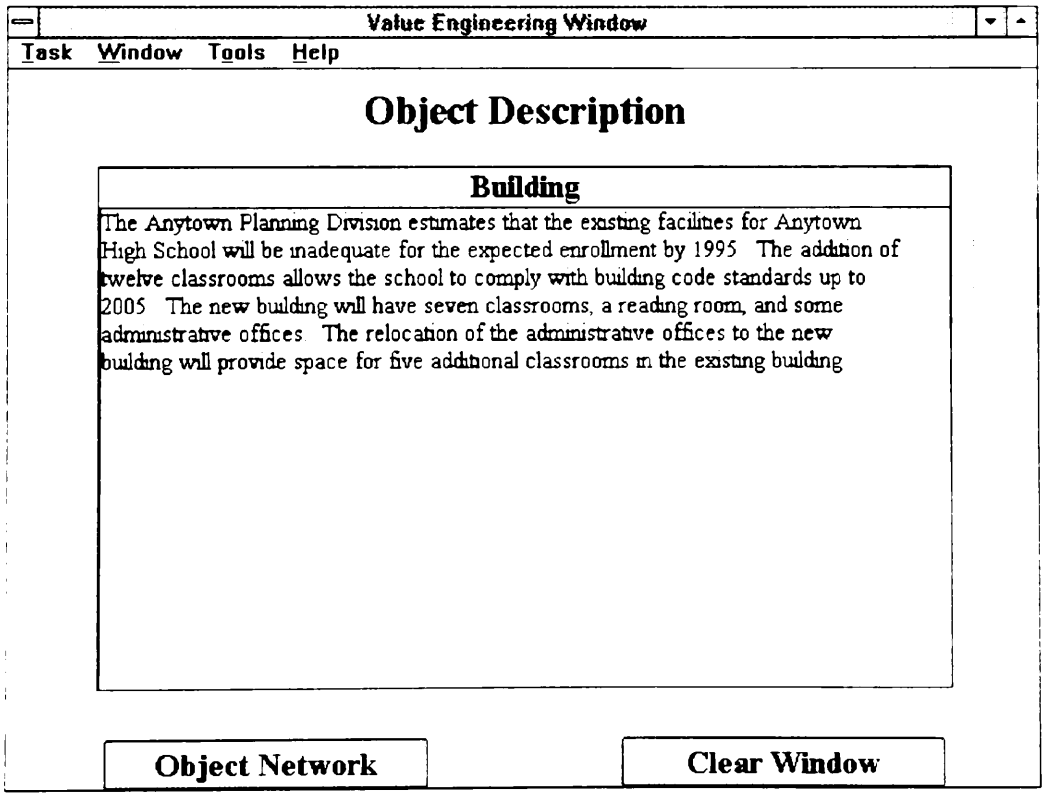


Figure 7-1. Object Description Task Form

Another type of rationale information is the description of the intent behind the values of the various object-attributes. DRIVE allows two methods of accessing this type of rationale information. One method displays a list of all the object-attribute assertions having completed the rationale capture process. Select the `Task|Retrieve Rationale Information|Display All Rationale` menu item to access all these assertions. Click over the various items in the `Rationale List` box of the Display All Captured Rationale Information Task Form (Figure 7-2). Note how DRIVE changes the information displayed in the lower portion of this task form to reflect the captured rationale information corresponding to the selected item. This method of accessing captured rationale information is useful for browsing through all the rationale captured for a particular project.

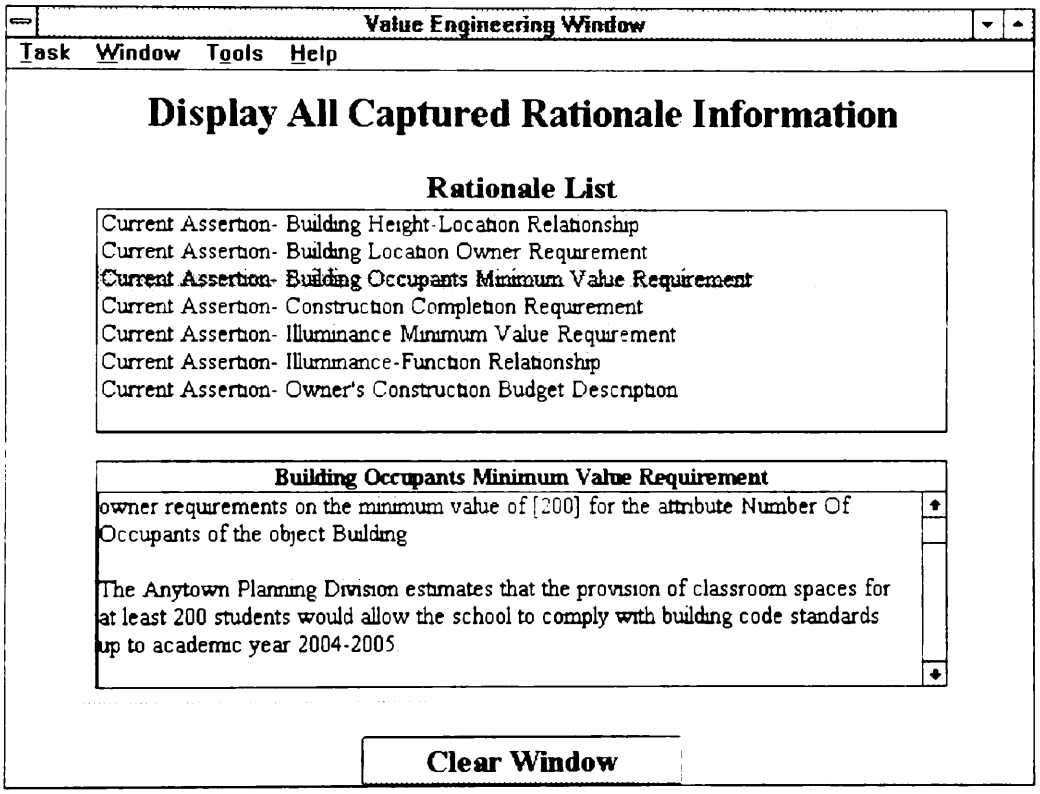


Figure 7-2 Display All Captured Rationale Information Task Form

Another method of accessing the description of the intent behind the values of the various object-attributes involves query searches. This method of accessing captured rationale information is useful for searching for specific rationale captured for a particular project. Select the `Task|Retrieve Rationale Information|Query Search on Capture Rationale` to formulate a query to process. Figure 7-3 shows the Query Search on Captured Rationale Task Form. Click the pull-down arrow corresponding to the `Object` selection box. DRIVE displays a list containing all the objects currently defined in the project. Select the

object `Building` from this list. DRIVE retrieves all the attributes of the object **Building** and places it in the pull-down list corresponding to the `Attribute` selection box. Select the attribute `Building Height` from the pull-down list of the Attribute selection box. Click the ➔ button to submit the particular object-attribute pair to the query definition box. Click the `Process Query` button to have DRIVE search for all captured rationale satisfying the defined query. DRIVE displays a task form similar to Figure 7-2. This task form displays only a list of rationale satisfying the current query.

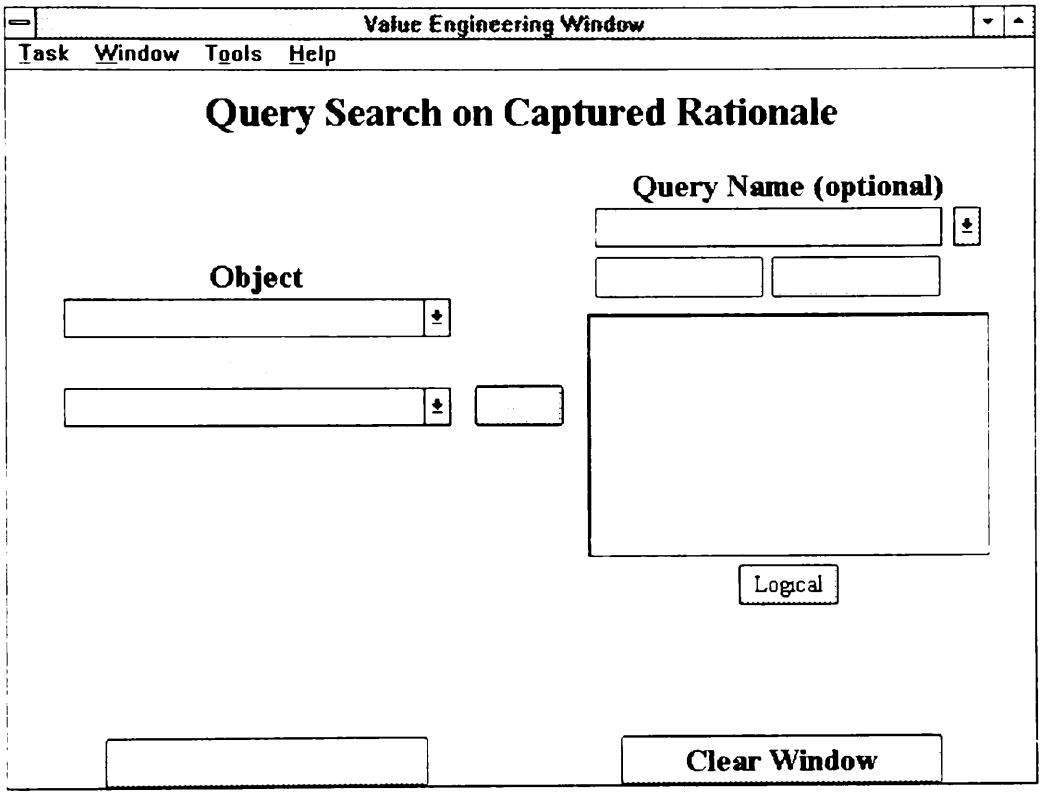


Figure 7-3. Query Search on Captured Rationale Task Form

DRIVE can also store query definitions. This capability eliminates the need to redefine again and again a particular query. Click the `Change Query` button to return to the Query Search on Captured Rationale Task Form. To illustrate the query definition concept, click over the `Query Name` edit/selection box. Type in `Building Height Query` inside this edit/selection box. Click the `Update Query` button to save the defined query as `Building Height Query`. Click the `Clear Window` button and select the `Task|Retrieve Rationale Information|Query Search on Captured Rationale` to reset this task form. Click over the pull-down arrow corresponding to the `Query Name` edit/selection box. DRIVE displays a pull-down list containing all the defined queries. Select the item `Building Height Query` from the pull-down list. DRIVE places the query definition of

Building Height Query in the query definition box. Click the Process Query button to perform the search for this query.

The Logical button allows you to perform boolean searches on the captured rationale. A boolean search involves combining two or more object-attribute pairs using And/Or relationships. An And relationship searches for captured rationale having both object-attribute operands. An Or relationship searches for captured rationale having at least one of its object-attribute operands. Click the Change Query button to return the previous task form. Click the Logical button and select the And item from the pop-up menu. Add the object-attribute pair [Building]:[Construction Completion Date] to the query definition box. Click the Process Query button to have DRIVE find a captured rationale relating the **Building Height** and the **Construction Completion Date**. Since the query definition changed, DRIVE asks for guidance on how to proceed. For this example, click the No button to proceed with the search without changing the definition of **Building Height Query**. Since there is no rationale satisfying this query, DRIVE displays a message stating this fact.

It is also possible to edit the statements contained in the query definition box. Click over one of the statements inside the query definition box to display a pop-up menu used for editing. Currently, DRIVE can only delete lines. Future versions of DRIVE may include the Edit Line and Insert Line capabilities. Thus, to change the previous query's And relationship to an Or relationship, delete the last two statements of the previous query by clicking over these statements and selecting Delete Line from the pop-up menu. Next, click the Logical button and select the Or menu item. Finally, use the Object and Attribute selection boxes and the ➔ button to place [Building]:[Construction Completion Date] into the query definition box. Click the Process Query button to have DRIVE search for all captured rationale containing **Building Height** and **Construction Completion Date**.

7.3 Displaying Object-Attribute Relationships

DRIVE helps users gain a better understanding of the relationships between the attributes of the various objects in the model through its Object Relationships Task Form (Figure 7-4). Select the Task|Access Object Network menu item from the Value Engineering Window. Search for and click over the text identifier for Building. Select the Show Relationships item from the pop-up menu.

The selection box at the top center of Figure 7-4 shows the object-attribute pair having the current focus. Click the pull-down arrow of this selection box. DRIVE displays a list of all the attributes of the focal object. Select the attribute Building Height from this pull-down list. This object-attribute pair becomes the current

focus of this task form. The **Affected By** and **Affects** list boxes show respectively, which object-attributes influence and depend on the current focus. For the current example, the object-attribute pair **[Building]:[Location]** influences the value of **Building Height**. Click over the text **[Building]:[Location]** to display how this particular object-attribute influences the current focal object-attribute pair. DRIVE displays this relationship in the lower portion of this task form.

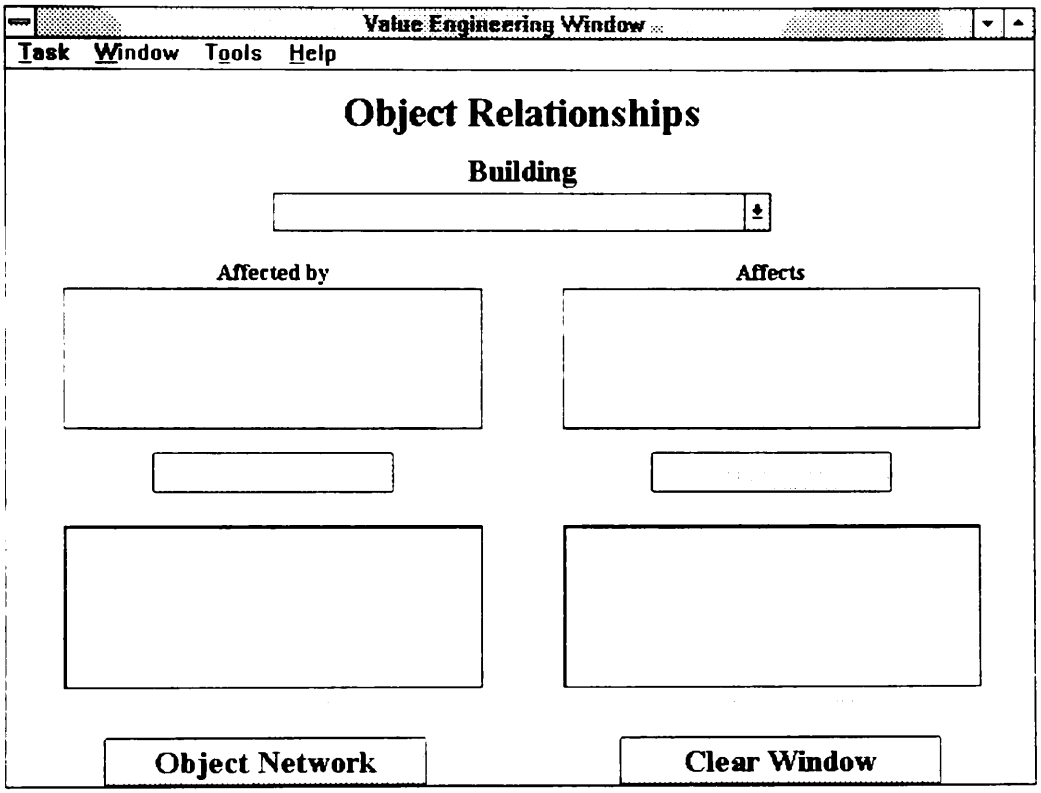


Figure 7-4. Object Relationships Task Form

DRIVE enables the **Change Focus** button when there is a selected object-attribute pair in the **Affected By** or **Affects** list boxes. Click over the **Change Focus** button below the **Affected By** list box to change the current focus from **[Building]:[Building Height]** to **[Building]:[Location]**. Notice how DRIVE changes the current focus selection box and the two list boxes to reflect the change in the focal object-attribute pair. Although this example only illustrates a one-step relationship, this capability is specially useful in understanding complex relationships in which one object-attribute affects another, which in turn, still affects another, and so on. The user can utilize this capability to "walk-through" the reasoning process leading to the specification of a particular value for an object-attribute.

7.4 AutoCAD Model Queries

DRIVE's AutoCAD model querying capability assists users in their design development and value engineering efforts. Select the `Task|Query AutoCAD Model` menu item from the Value Engineering Window. DRIVE displays the Query-Color Table Task Form (Figure 7-5). The various color strips in this task form correspond to the colors used in the AutoCAD model to denote objects satisfying a particular query. DRIVE can currently process up to six different queries for display in AutoCAD. Click the `New Query` button corresponding to the red color strip. DRIVE displays the Object-Attribute Query Definition Task Form (Figure 7-6). This task form is one of two Query Definition Task Forms used by DRIVE. The other type is the Object Query Definition Task Form (Figure 7-7). The `Query Type` selection box in the upper left hand corner of these task forms allows you to switch from one to the other.

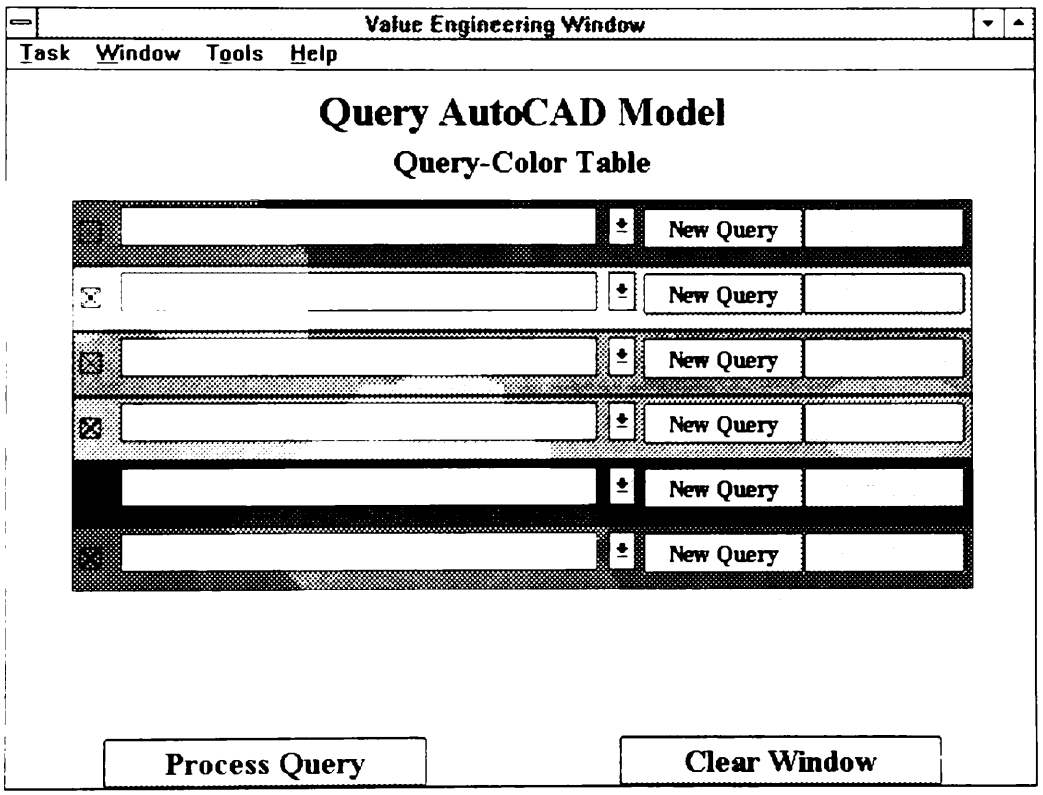


Figure 7-5. Query-Color Table Task Form

The Object Query Definition Task Form allows you to display an object or group of related objects. Select the `Object Query` item from the pull-down list of the `Query Type` selection box to display this task form. To have AutoCAD display the results of a query in a specific color, the query must have a name identifier. Click over the `Object Query` edit/selection box and type in `Class Room Spaces Query`

as the name identifier. The upper and lower list boxes in the left side of this task form contain a list of space type groups and a list of model spaces, respectively. The upper list box contains the following space type groupings: by hierarchy, by function, and by building story level. The right side list box defines the list of space type groups and/or model spaces to include in this particular query. Search for and select the `Class Room Spaces` identifier in the `Space Classes` list box. Click the ➔ button to place this space type group in the query definition. You can mix and add any number of space type groups and model spaces to the query definition box. Click the `Update Query` button to attach this query definition to **Class Room Spaces Query**. Click the `Query-Color Table` button to return to the Query-Color Table Task Form. Finally, click the `Process Query` button to have AutoCAD display the results of this particular query.

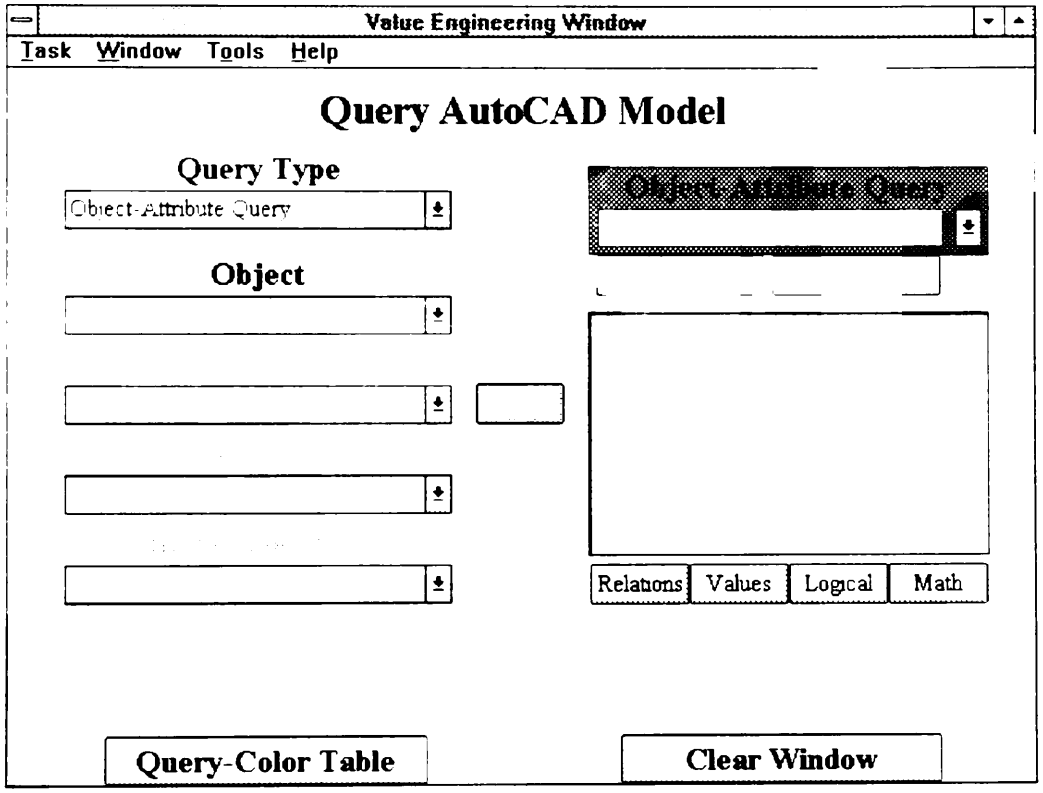


Figure 7-6. Object-Attribute Query Definition Task Form

DRIVE processes the Class Room Spaces Query by manipulating the AutoCAD model to change the color of all Class Room objects to red. Click the `New Query` button for the color yellow. Define an object query for **First Floor** objects and use `First Floor Objects` as its name identifier. The AutoCAD model now contains objects in three different colors. The red objects correspond to the **Class Room Spaces Query**, the yellow objects are the **First Floor Objects**, and the black objects are the objects not satisfying the first two queries. DRIVE can also turn

off these colors for clarity purposes. The check boxes to the extreme left of the various color strips in the Query-Color Table Task Form control the display of these colors in the AutoCAD model. Click the check box corresponding to the yellow color strip to hide all **First Floor Objects** in the model.

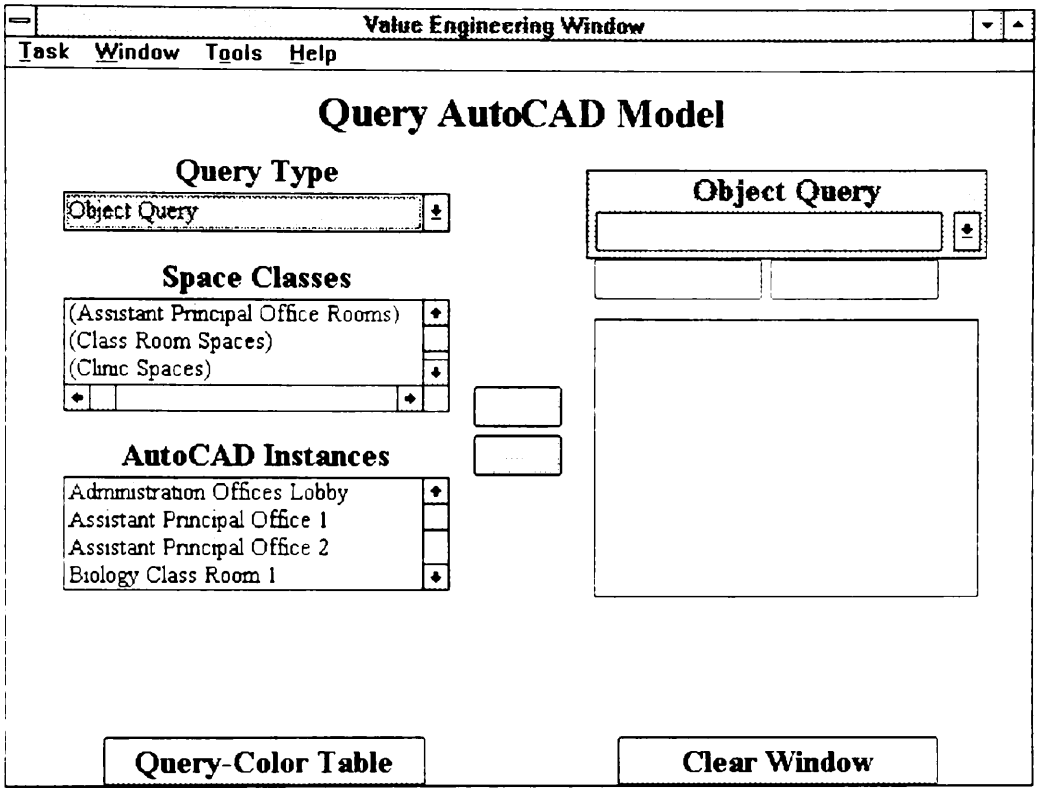


Figure 7-7. Object Query Definition Task Form

Redisplay the yellow objects by clicking again on the yellow check box. Click the pull-down arrow inside to the green color strip. DRIVE displays a list of currently defined queries. Select the `Class Room Spaces Query` item. DRIVE changes the color of these objects from red to green. Remove the queries attached to the yellow and green colors by erasing the texts inside the appropriate edit/selection boxes.

The Object-Attribute Query Definition Task Form (Figure 7-6) allows you to display particular objects having attribute values satisfying your defined criteria. Click the red `New Query` button to switch to this task form. Name this query as `Budgeted Cost Under $25,000`. Click the pull-down arrow for the `Object` selection box. Select the `Building Spaces` item from the pull-down list. DRIVE now enables the `Attribute` selection box. Select `Construction Cost` from this selection box. DRIVE now enables the `Constraint` and `Stage` selection boxes as well as the ➔ button. Allocated Budgets (As Required) and Preliminary Estimates

(As Designed) are two types of costs currently defined for the project. The **Stage** selection box specifies which stage to include in the query definition. This selection box is optional. An empty value for this selection box forces DRIVE to evaluate the defined query for all stages. Select **As Required** from this selection box. Click the ➔ button to submit this object-attribute-constraint-stage combination to the query definition box.

The four buttons below the query definition box function in a similar manner to the buttons in the Logical Relationship Rationale Windows (Section 3.3.10). Click the **Relations** button and select the < menu item. Click the **Values** button and type in **25000**. Click the **Update Query** and **Query-Color Table** buttons in sequence.

Define similar queries for budgeted costs between $25,000 and $50,000; between $50,000 and $75,000; between $75,000 and $100,000; and greater than $100,000 and attach these queries to different colors. A sample query definition is as follows:

[Building Spaces]:[Construction Cost] (As Required)
is not smaller than 25000 And
[Building Spaces]:[Construction Cost] (As Required)
is smaller than 50000.

Clear all queries attached to the various colors by removing the texts inside the various edit/selection boxes.

7.5 Value Engineering Diagrams

The Value Engineering Window allows quick access to the various cost breakdown diagrams. Select the **Task|Display Cost Breakdown Diagrams** to display the pie charts for the **Construction Budget** and **Preliminary Estimate Breakdowns**. These menu items switch you to either the Concept or Design Windows. Chapter 5 gives a detailed explanation of these Cost Breakdown Diagrams.

The Value Engineering Window generates and displays a FAST diagram. FAST stands for Functional Analysis Systems Technique. Value engineers use this diagram as a starting point in the analysis of the functions of a project. Select the **Task|Display FAST Diagram** to display Figure 7-8. Clicking over a function text identifier displays a pop-up menu. The two items on this menu are: **Cost Breakdown Diagrams** and **Highlight CAD Objects**. Chapter 5 explains the Cost Breakdown Diagrams.

The FAST Diagram allows you to have AutoCAD display the various objects fulfilling a given function. Click over the text identifier **Teach Students** and select **Highlight CAD Objects** from the pop-up menu. Select a color to display the

objects having the function **Teach Students**. Repeat the process for the other functions. Finally, save and close the current project to clear the working memory of the DRIVE application.

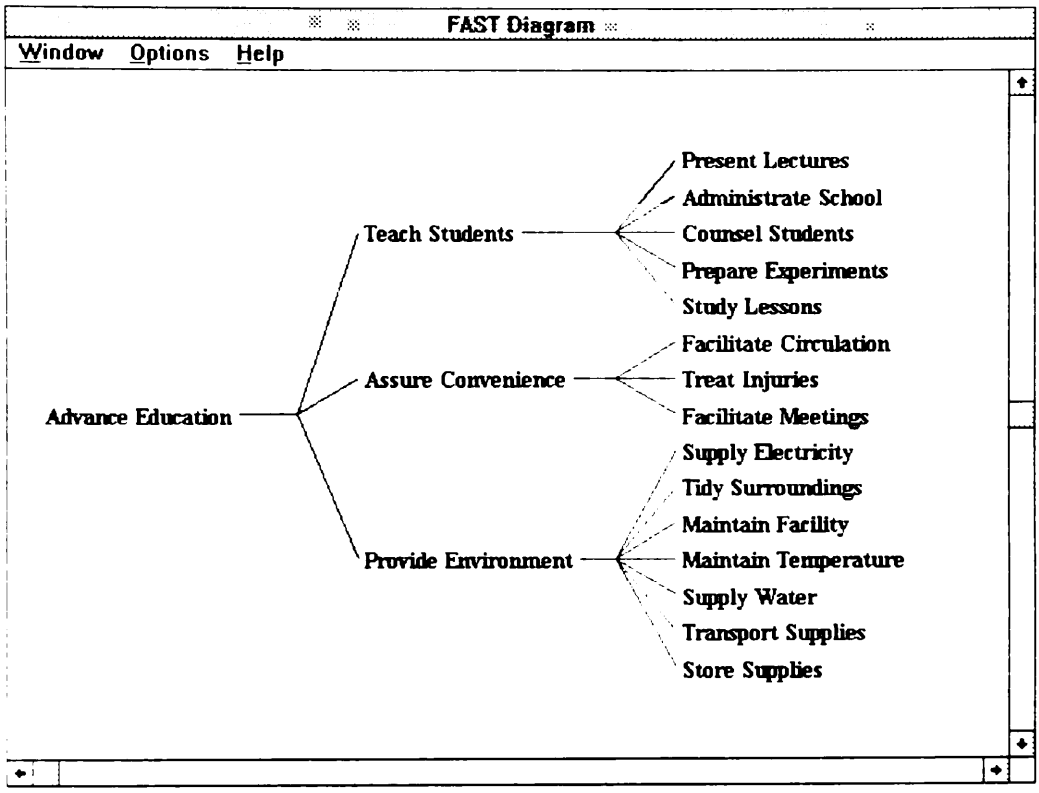


Figure 7-8. FAST Diagram

Tutorial Chapter 8
Data Verification Module

Chapter 8 shows how to use the data verification module. In this chapter, you will learn how to use the various modes of data verification, as well as the various conflict resolution windows.

8.1 Data Verification Menu

Open the project you created in Chapter 7 or `Tutorial Chapter 7`. If you are using `Tutorial Chapter 7`, please copy the project into a new project. See Section 3.1 for the procedure for copying projects.

Figure 8-1 shows the various modes of operation of the data verification module. `Verify All New Data` triggers the data verification module every time a value changes for an object-attribute. If the Data Verification Module is `Off` and a value changes in an object-attribute, the system accepts this value change without verification. `Verify Existing Data` allows you to perform a consistency check for the entire project database. `Process Known Conflicts` allows you to process conflicts detected during the last execution of the `Verify Existing Data` command. The `Unload Data Verification Module` menu item removes this module from the working memory of DRIVE.

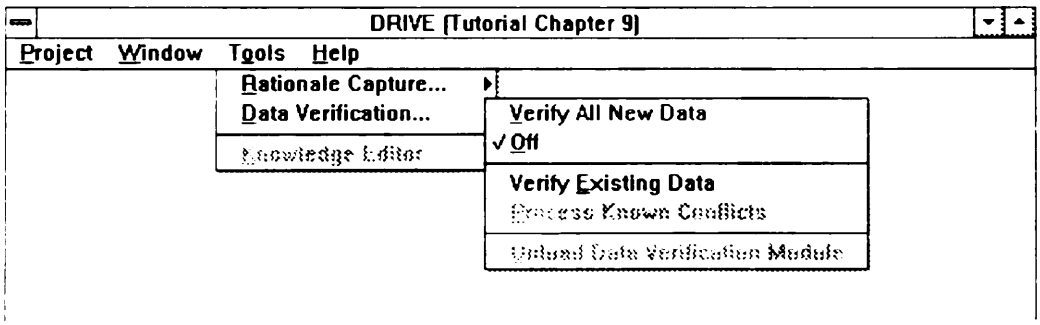


Figure 8-1. Data Verification Menu

8.2 Conflict Resolution Windows

The Conflict Resolution Windows allow users to resolve conflicts arising from inconsistencies between the current design decisions and previously captured design rationale. Switch the data verification module to `Verify All New Data` mode by selecting the `Tools|Data Verification|Verify All New Data` menu item. Display the `Concept Window` by selecting the `Window|Concept Window` menu item. Edit the building requirements by selecting the `Task|Building`

Requirements menu item. Set the performance type to **Suitability Performance** by clicking on the pull-down arrow corresponding to the Performance Type box and selecting Suitability Performance from the pull-down list. Edit the **Number of Occupants** attribute of the **Building** by scrolling through the Performance Parameters list box, highlighting the Number of Occupants attribute, and clicking the Edit Parameter button. Force a constraint conflict to occur by setting the value of this parameter to 150, which is less than its minimum value constraint value of 200. Although this constraint conflict example seems trivial because the user already sees that the minimum value is 200, the value of 150 might have been set by another relationship not just by direct user input. Click the Accept Changes button. The DRIVE system now checks the new data and detects a conflict. DRIVE then presents the window shown in Figure 8-2. This window informs the user of the constraint conflict and asks the user for guidance on how to resolve this conflict. Figure 8-2 shows how DRIVE notifies the user of a constraint violation conflict. Users can change either the value or the constraint to remove the constraint violation. Alternatively, users can click the Process Conflict Later button to defer the resolution of this constraint conflict. For this tutorial example, click the Process Conflict Later button.

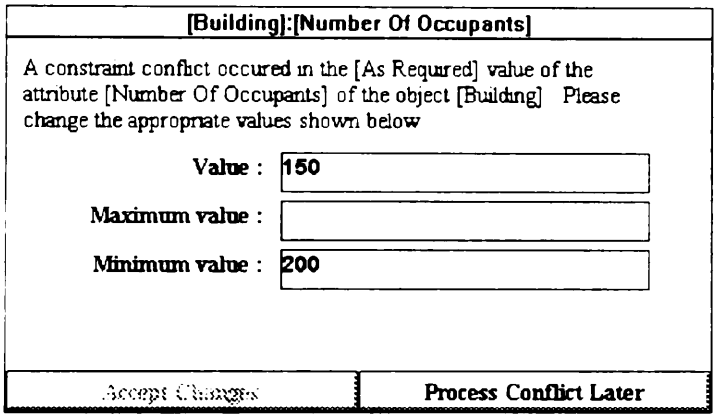


Figure 8-2. Constraint Conflict Resolution Window

Another type of conflicts DRIVE can detect are relationship conflicts. One of the relationships you created in Tutorial Chapter 3 is the **Building Height - Location Relationship** (Section 3.3.6). Violate this relationship by setting the value of the maximum value constraint of the **Building Height** parameter to **60** feet. Switch back to the Concept Window. Edit the **Building Height** attribute of the **Building** by scrolling through the Performance Parameters list box, highlighting the Building Height attribute, and clicking the Edit Parameter button. Force a conflict to occur by changing the maximum value of this parameter to 60.

Figure 8-3 shows how DRIVE notifies the user of a relationship inconsistency conflict. The four buttons in this window allows users to select one of four possible courses of action. These actions are `Modify Attribute Values`, `Modify Relationship`, `Invalidate Relationship`, and `Process Conflict Later`. The actions of these buttons are evident from their titles.

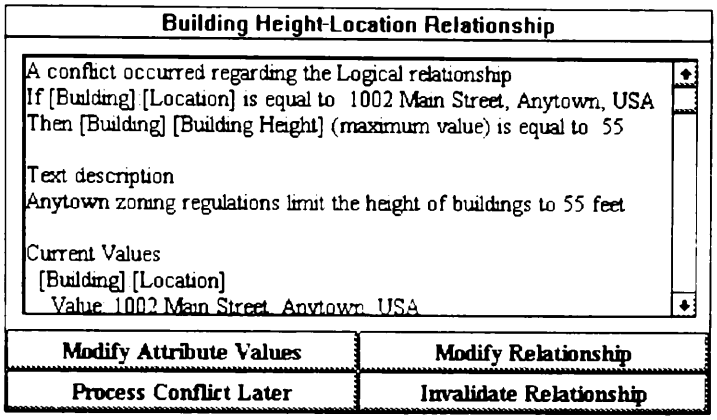


Figure 8-3. Relationship Conflict Resolution Window

Click the `Modify Attribute Values` button to display the window shown in Figure 8-4. Modifying attributes to resolve a relationship conflict is a two-step process. Figure 8-4 shows the first step, namely, selecting the attribute to modify. DRIVE displays a list of all the attributes relevant to the relationship that caused the conflict. For this example, the relevant attributes are **Building Height** and **Location**. Clicking the `Previous Conflict Window` button in this window takes you back to the Relationship Conflict Resolution Window shown in Figure 8-3.

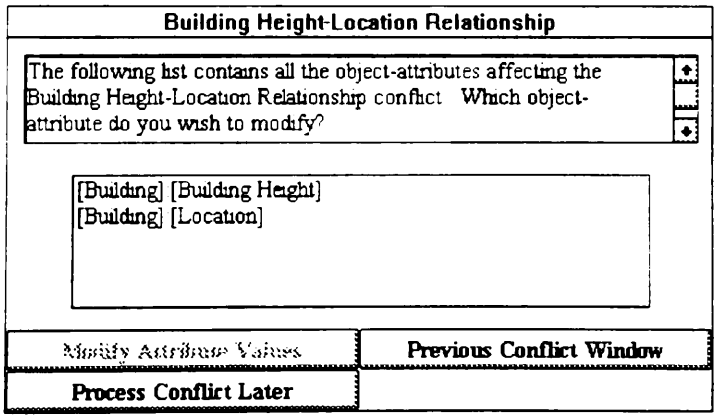


Figure 8-4. Conflict Resolution Window: Attribute Selection

Highlight the `Building Height` attribute and click the `Modify Attribute Values` button to display the window shown in Figure 8-5. This window shows the

second step in modifying attributes to resolve a relationship conflict. This step actually modifies the value or constraint of the selected attribute. Clicking the **Previous Conflict Window** button in this window takes you back to the previous window shown in Figure 8-4. For this tutorial example, click the **Previous Conflict Window** button twice to return to the Relationship Conflict Resolution Window shown in Figure 8-3.

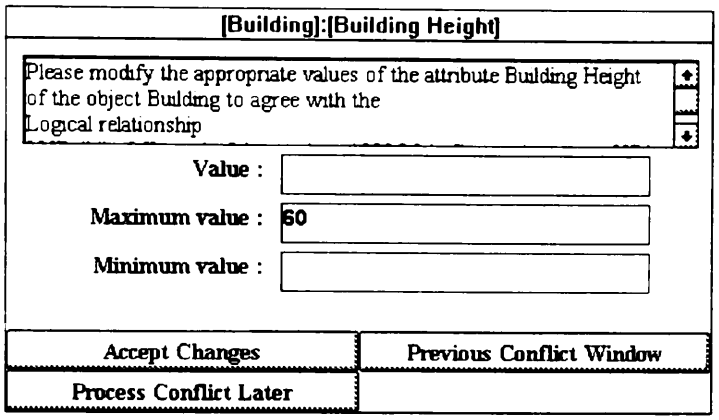


Figure 8-5. Conflict Resolution Window: Attribute Modification

Click the **Modify Relationship** button to display the window shown in Figure 8-6. This window shows a window similar to the relationship definition window explained in detail in Section 3.3.10. For this tutorial example, the relationship states the location of the building limits its maximum height to 55 feet. By setting the value to 60 feet, the user created a design inconsistency. Figure 8-6 shows how DRIVE allows users to modify the relationship causing the design inconsistency. Click the **Previous Conflict Window** button in this window to take you back to the window shown in Figure 8-3.

DRIVE also allows the user to totally remove a relationship. Click the **Invalidate Relationship** button to accomplish this task. DRIVE then displays a confirmation question if this is really what the user wants to do. For this tutorial example, click the **No** button in the confirmation window shown in Figure 8-7.

Finally, DRIVE also allows the user to defer the resolution of this conflict. Click the **Process Conflict Later** button to accomplish this task.

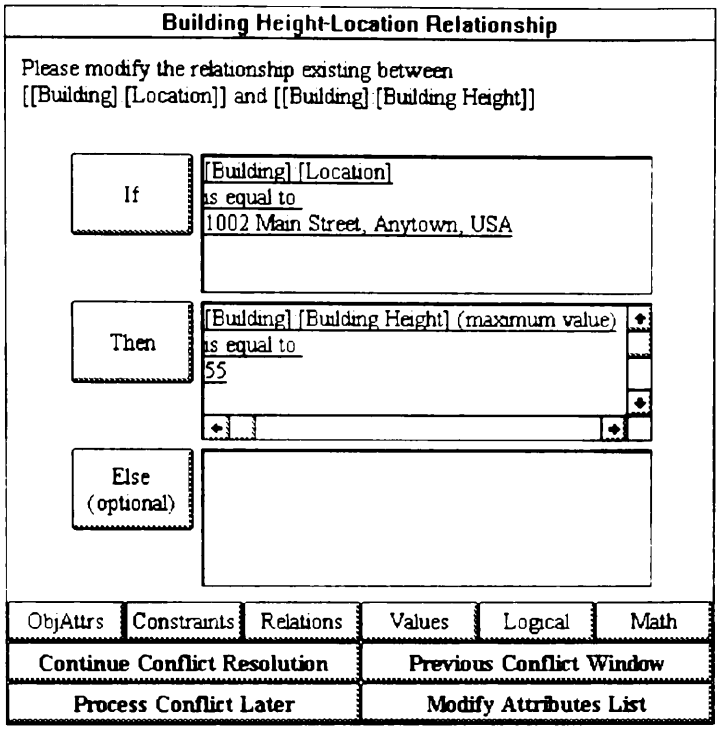


Figure 8-6. Conflict Resolution Window: Relationship Modification

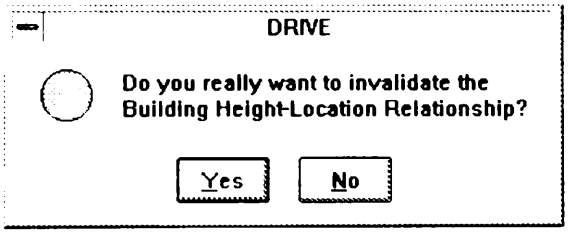


Figure 8-7. Conflict Resolution Window: Invalidating Relationships

8.3 Processing Existing Conflicts

Users can process the two unresolved conflicts in the previous section by selecting the `Tools|Data Verification|Verify Existing Data` menu item. DRIVE then checks the entire database for any constraint and relationship conflicts. After detecting data conflicts, DRIVE presents these conflicts in a list box as shown in Figure 8-8. Users can then select which conflicts they want to resolve. The list box in Figure 8-8 allows multiple selection. For this tutorial example, highlight both unresolved conflicts and click the `Process Conflict` button. DRIVE then creates two conflict windows allowing users to resolve the selected conflicts. These conflict windows are the same as those shown in Figures 8-2 and 8-3. Finally, DRIVE provides a mechanism to quickly switch between displayed conflict windows. Figure 8-9 shows this mechanism.

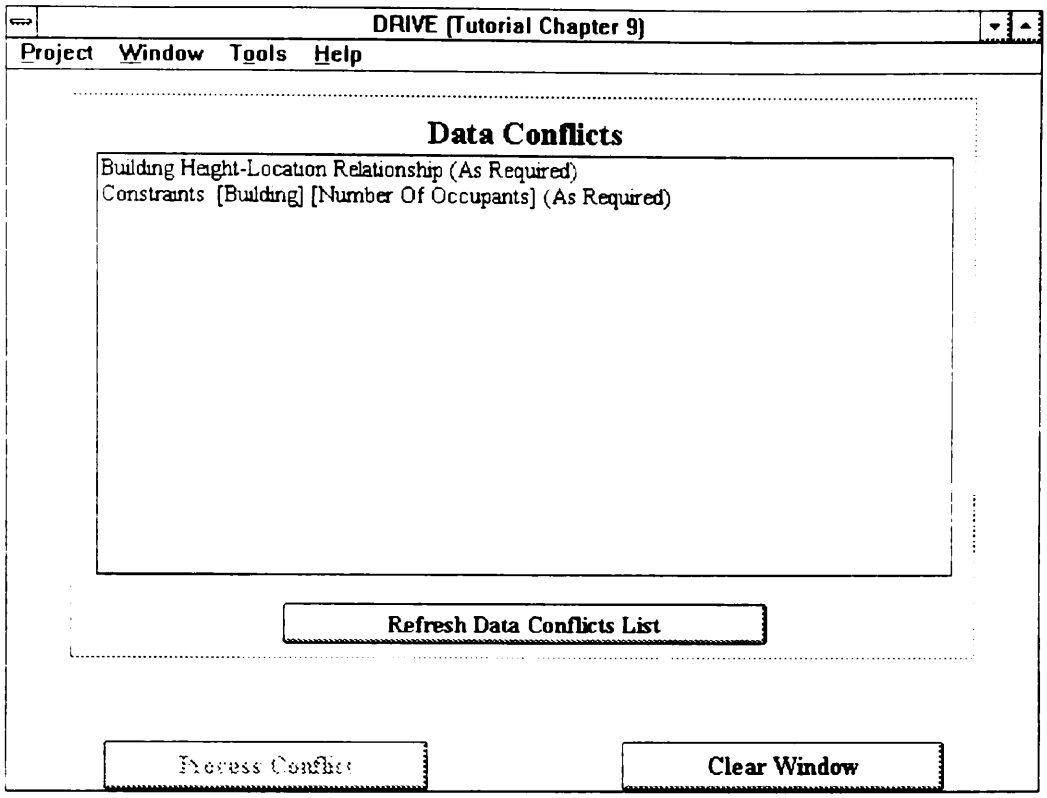


Figure 8-8. Processing Existing Conflicts

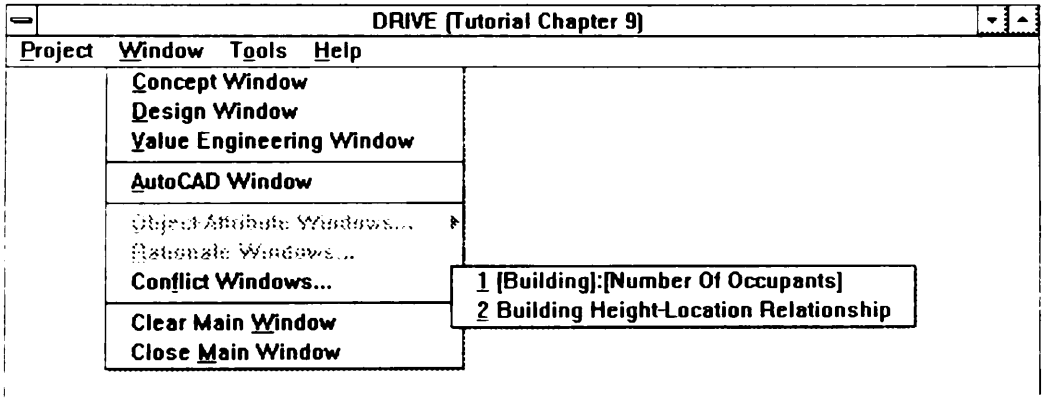


Figure 8-9. Accessing Active Conflict Windows

Tutorial Chapter 9
Domain Knowledge Editor

Chapter 9 discusses the domain knowledge editor. In this chapter, you will learn how to define default values for the domain knowledge libraries, manipulate the performances library, and modify the layouts library.

9.1 Overview

The tutorial on the Domain Knowledge Editor is different from the other tutorial chapters in this manual. The other chapters required you have to use a project for demonstration purposes. This chapter on the other hand, requires that no project is currently active for demonstration. If you have a project currently open, please use the `Project|Close` menu item in the main DRIVE window.

Select the `Tools|Knowledge Editor` menu item in the main DRIVE window to switch to the domain knowledge editor mode. Figure 9-1 shows the DRIVE interface for editing the domain knowledge. The `Tasks` menu allows access to all the domain knowledge hierarchies. For the building construction domain, there are three types of domain knowledge hierarchies, namely, the domain knowledge library, the performances library, and the layouts library. The succeeding sections in this chapter discuss each of these hierarchy types.

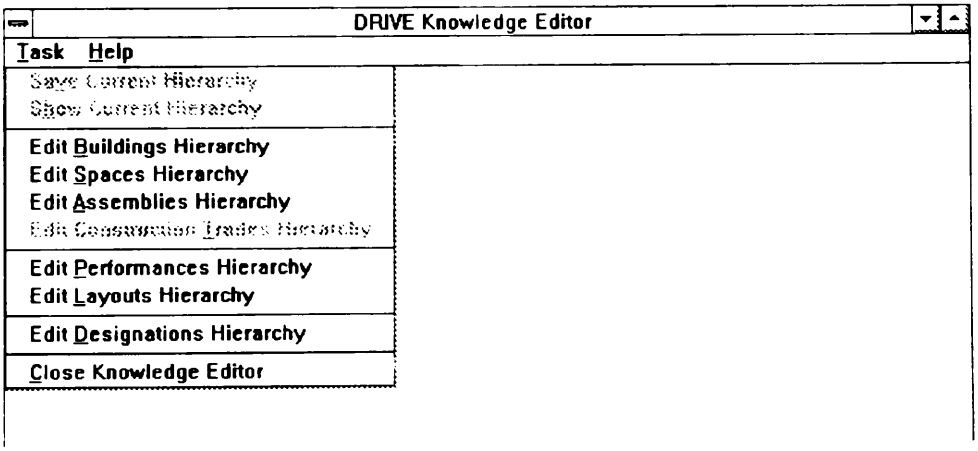


Figure 9-1 Domain Knowledge Editor Tasks

9.2 Building Construction Domain Libraries

Selecting the `Task|Edit Buildings Hierarchy`, `Task|Edit Spaces Hierarchy`, or `Task|Edit Assemblies Hierarchy` menu items in the knowledge editor window takes the user to a window similar to the one shown in Figure 9-2. These

windows allow the user to select a specific class in a knowledge hierarchy to modify. Figure 9-2 shows the window for selecting a **Spaces Hierarchy** object to modify. Clicking on one of these text identifiers displays the pop-up menu shown in Figure 9-2. In this menu, users can edit, delete, rename, or add sub-classes to the selected object.

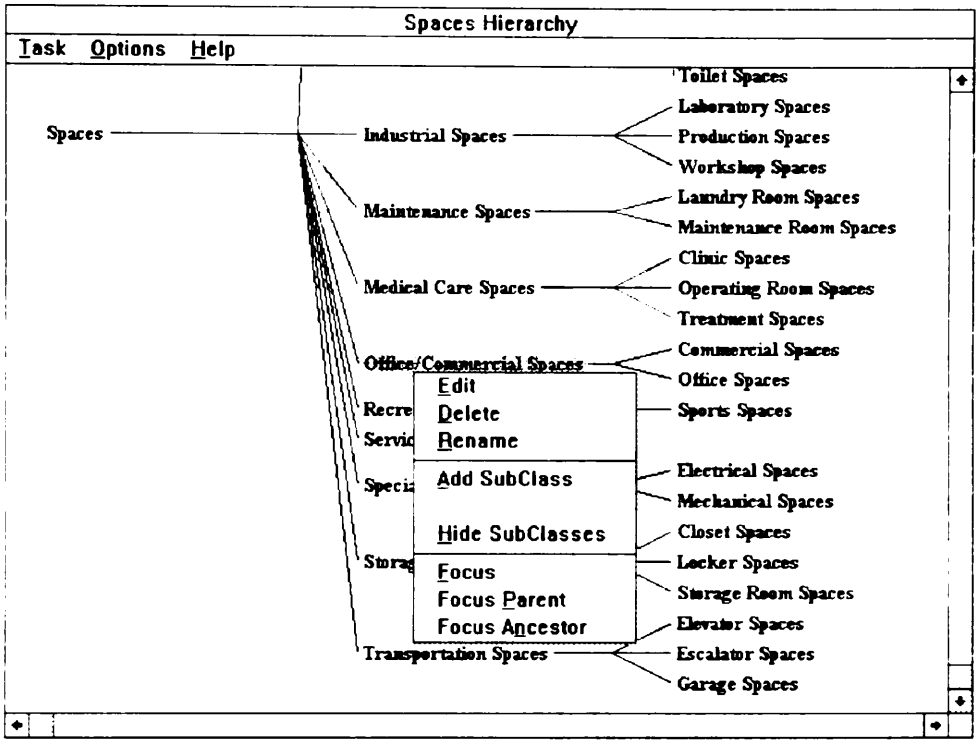


Figure 9-2. Modifying the Spaces Hierarchy

Selecting the `Edit` pop-up menu item takes the user to the window shown in Figure 9-3. Users can define the valid performances for the selected object class through this window. Users can also set default values for the various performance parameters. These default values form the basis for the initial values given to the design objects created for a specific project. For example, in Figure 9-2, if the user types in `Support Business Activities` as the default value for the `General Function` parameter of the `Office/Commercial Spaces` object, then all new spaces created having the `Office/Commercial Spaces` object as its parent will have `Support Business Activities` as the value for its `General Function` parameter. Figure 9-3 also shows two tabs near the top of the window. These tabs allow users to either edit the performance parameters or the layout parameters. Clicking on the `Layout` tab takes the user to the window shown on Figure 9-4. The large blank space on the right portion of this window acts as a display area for the selected default graphical layout selected on the pull-down list box on the left side.

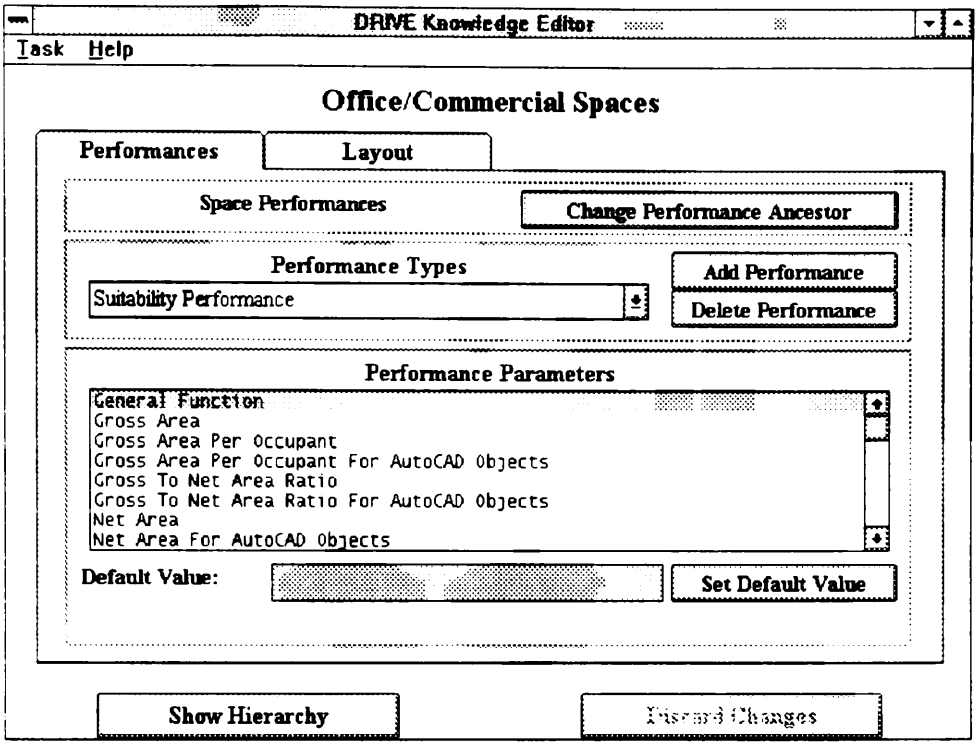


Figure 9-3. Setting Default Values

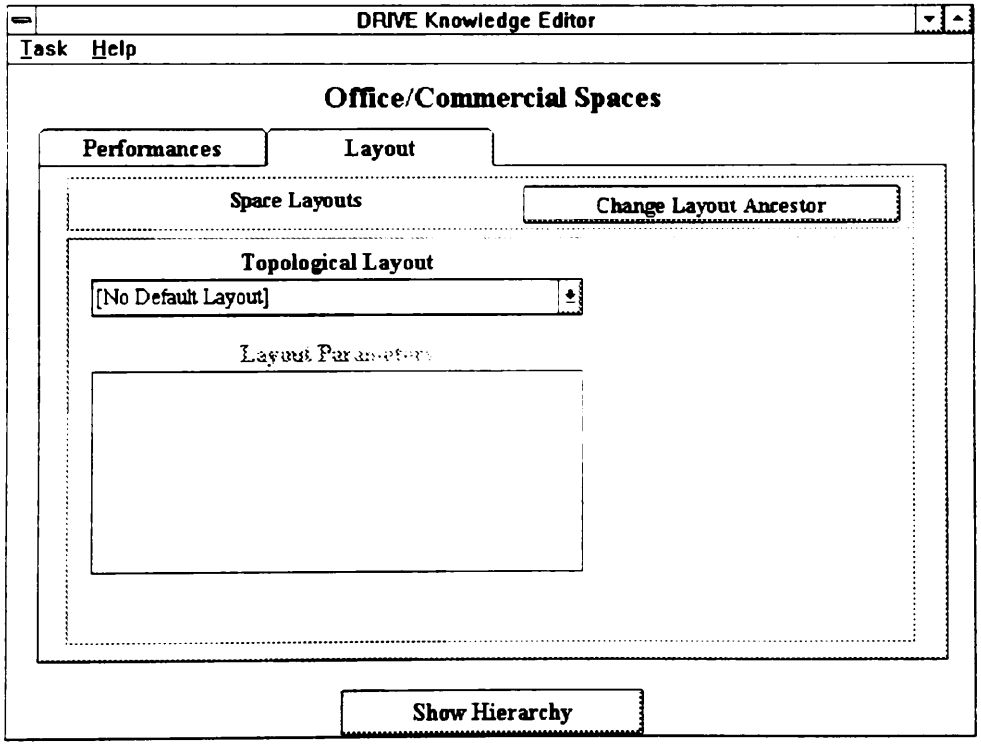


Figure 9-4 Setting Default Layouts

9.3 Performances Library

Selecting the `Task|Edit Performances Hierarchy` menu item in the knowledge editor window takes the user to the window shown in Figure 9-5. This window allows the user to select a specific performance to modify. Clicking on one of the text identifiers shown in Figure 9-5 displays a pop-up menu. In this menu, users can edit, delete, rename, or add sub-classes to the selected performance. These performances contain the parameter definitions used in generating the parameter lists in windows such as the one shown in Figure 9-3.

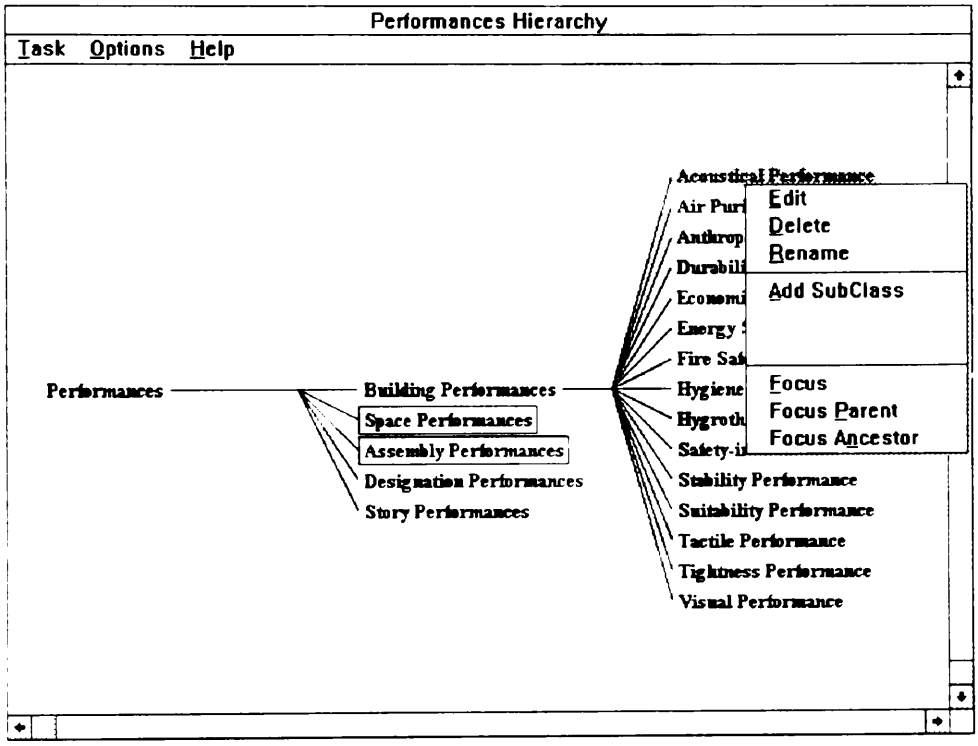


Figure 9-5. Modifying the Performances Library

Selecting the `Edit` pop-up menu item takes the user to the window similar to the one shown in Figure 9-6. Clicking on the `New Parameter` button displays the interface objects at the lower portion of Figure 9-6. These interface objects allow a user to define the characteristics of new parameters for the selected performance. These characteristics currently consist of `Cardinality` and `Value Type`. Future versions of DRIVE may include other parameter characteristics. For example, typing the text Resonance Frequency in the `Parameter Name` text box, selecting `Single` and `Numeric` as the `Cardinality` and `Value Type` characteristics, and clicking the `Create Parameter` button creates a new parameter for the `Acoustical Performance`. Highlighting a parameter shown in the list allows users to modify its characteristics as shown in Figure 9-7. Notice that the `Create Parameter` button changes to the `Update Parameter` button.

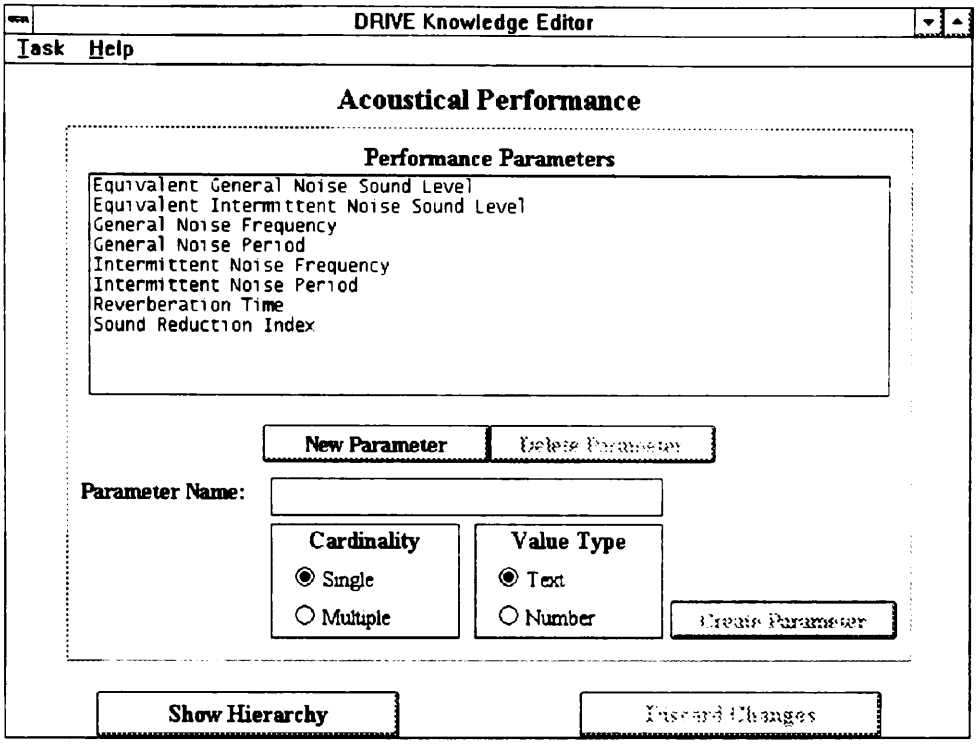


Figure 9-6. Creating New Performance Parameters

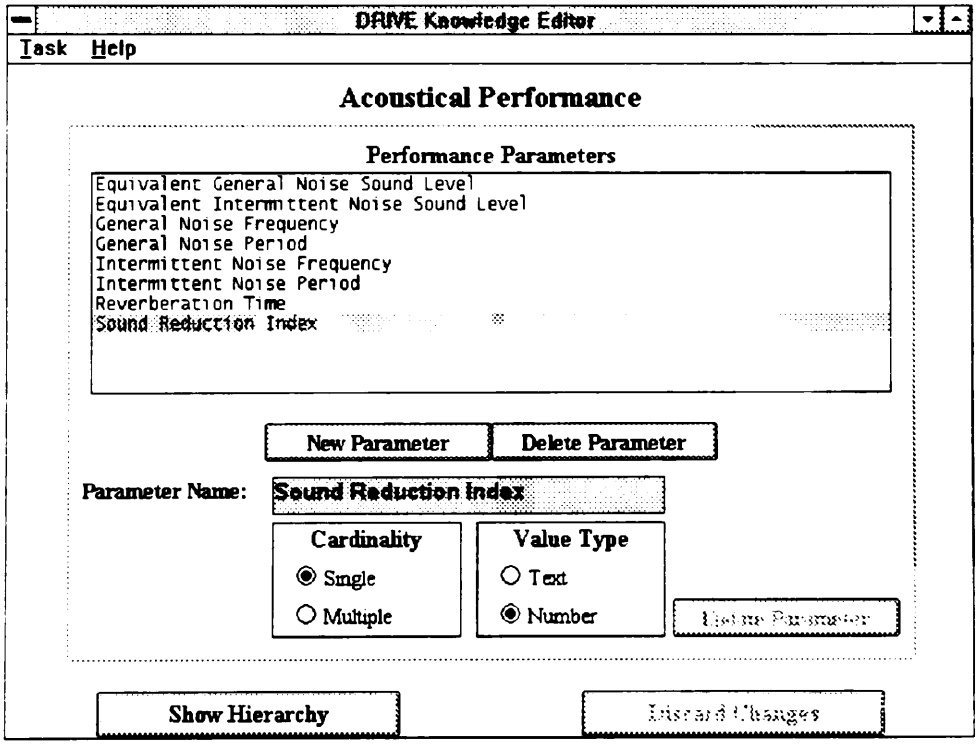


Figure 9-7. Modifying Existing Performance Parameters

9.4 Layouts Library

Selecting the `Task|Edit Layouts Hierarchy` menu item in the knowledge editor window takes the user to the window shown in Figure 9-8. The operation of this window is similar to the windows described in the previous two sections. The layouts shown in Figure 9-8 contain the layout definitions used in generating the parameter list and graphical representation image in windows such as the one shown in Figure 9-4.

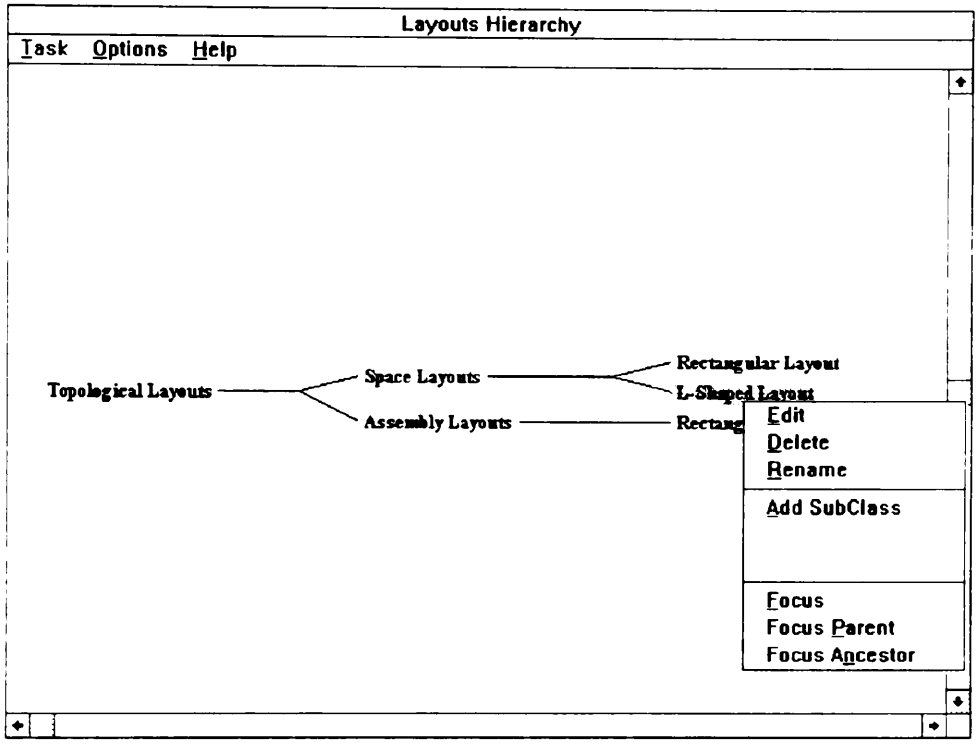


Figure 9-8. Modifying the Layouts Library

Selecting the `Edit` pop-up menu item takes the user to the window similar to the one shown in Figure 9-9. This window operates similarly to the performance library windows discussed in the previous section. Unlike the performances library, the parameters in the layouts library are all single-valued and numeric. Hence, there is no need for the parameter characteristics interface objects in Figure 9-9. Also, layout classes require the use of a graphical image to display the meaning of the names of the layout parameters.

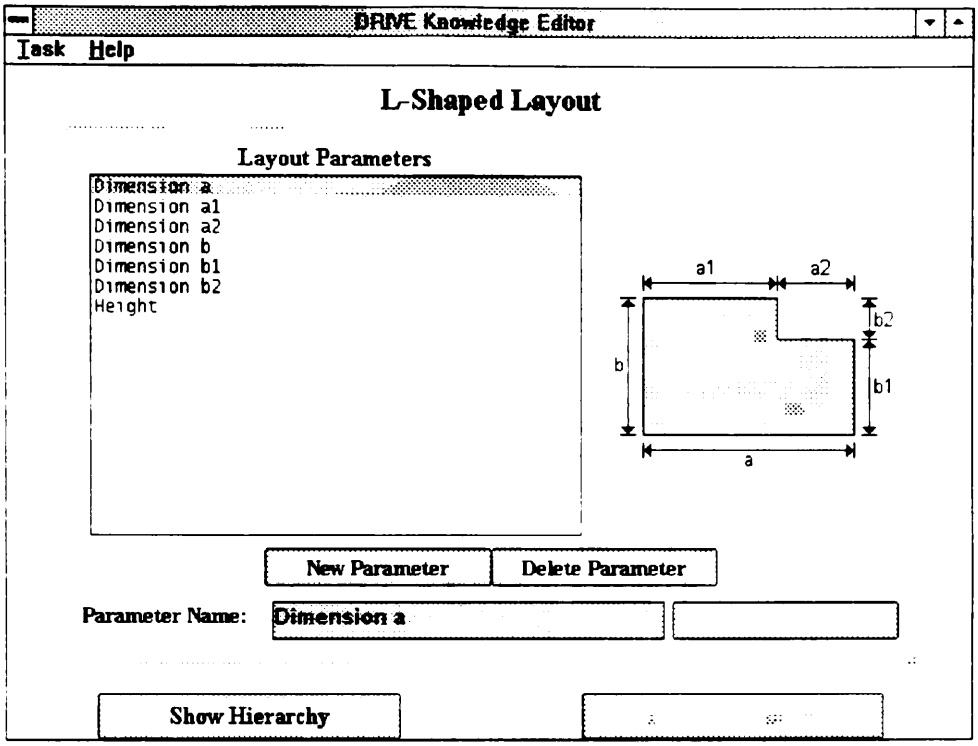


Figure 9-9. Modifying Existing Layout Parameters

APPENDIX C

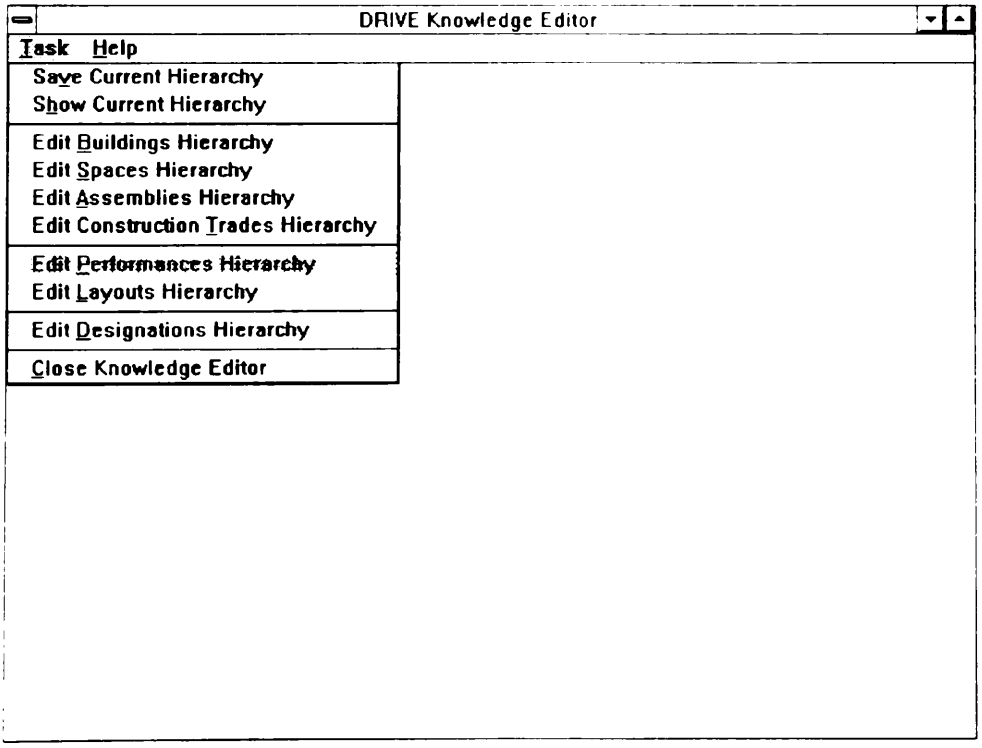
PRESENTATION SLIDES: DRIVE CAPABILITIES

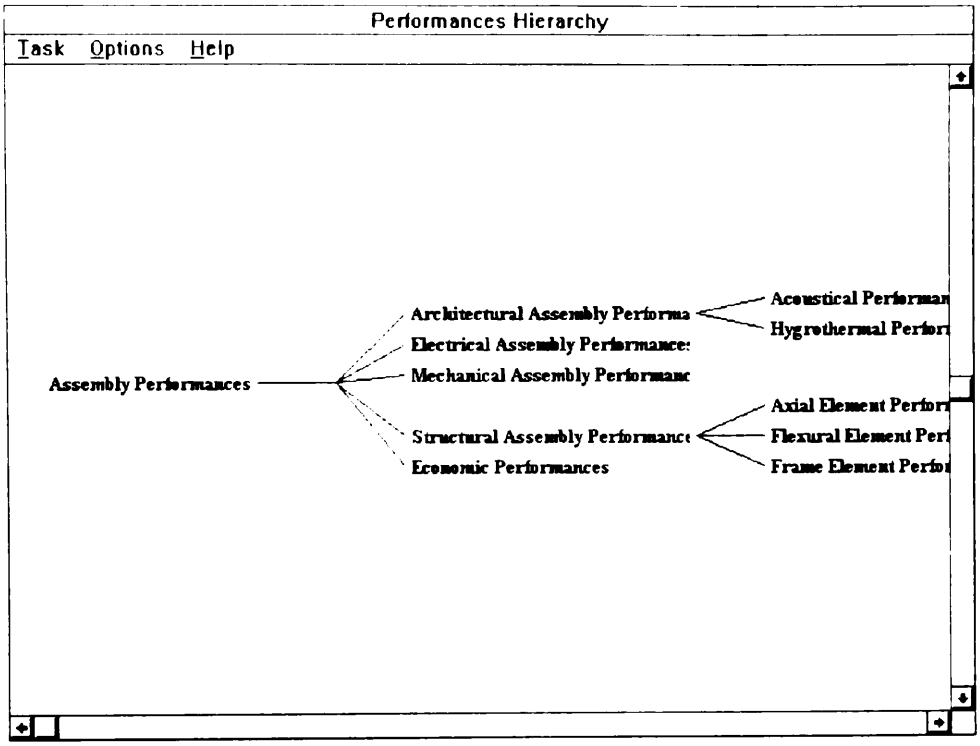
DRIVE Demonstration

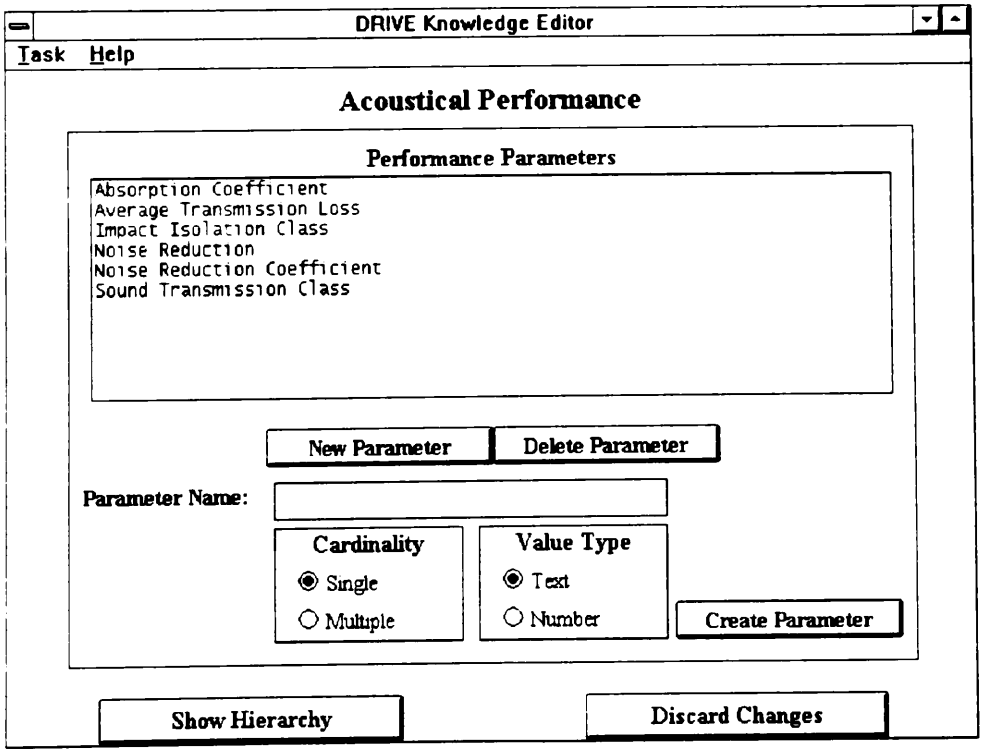
- Domain Knowledge Editor

Demo: Domain Knowledge Editor

- Add a new parameter Resonance Frequency to Acoustical Performance (Architectural Assemblies).

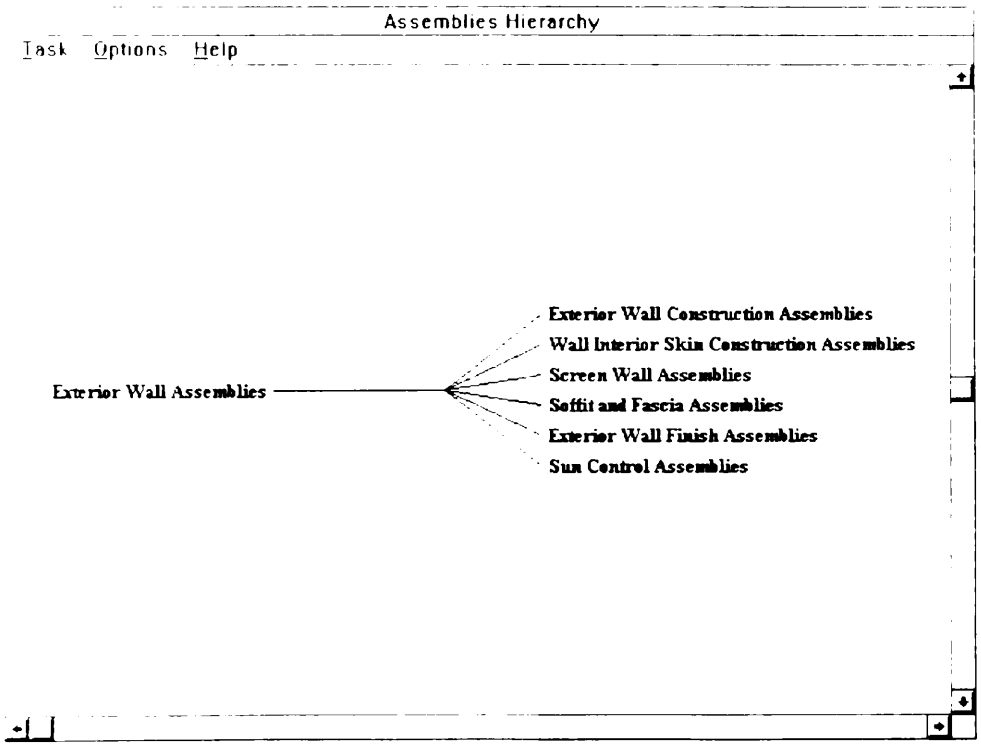


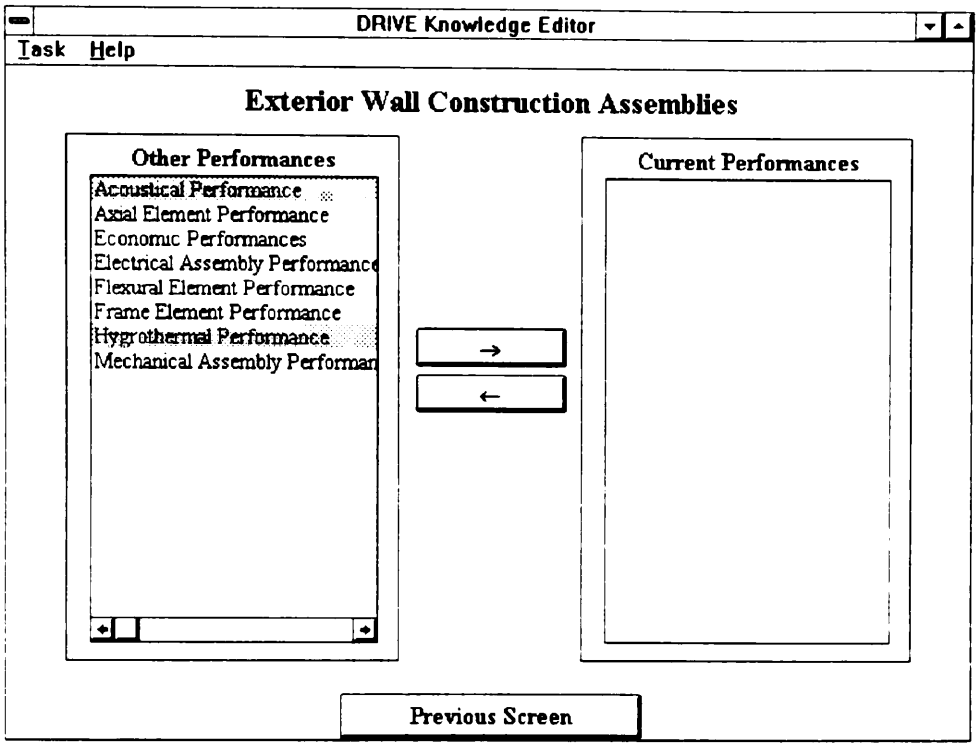

Performances Hierarchy
Task Options Help
Assembly Performances
Architectural Assembly Performa
Electrical Assembly Performance:
Mechanical Assembly Performanc
Structural Assembly Performance
Economic Performances
Acoustical Performan
Hygrothermal Perfori
Axial Element Perfori
Flexural Element Perf
Frame Element Perfo

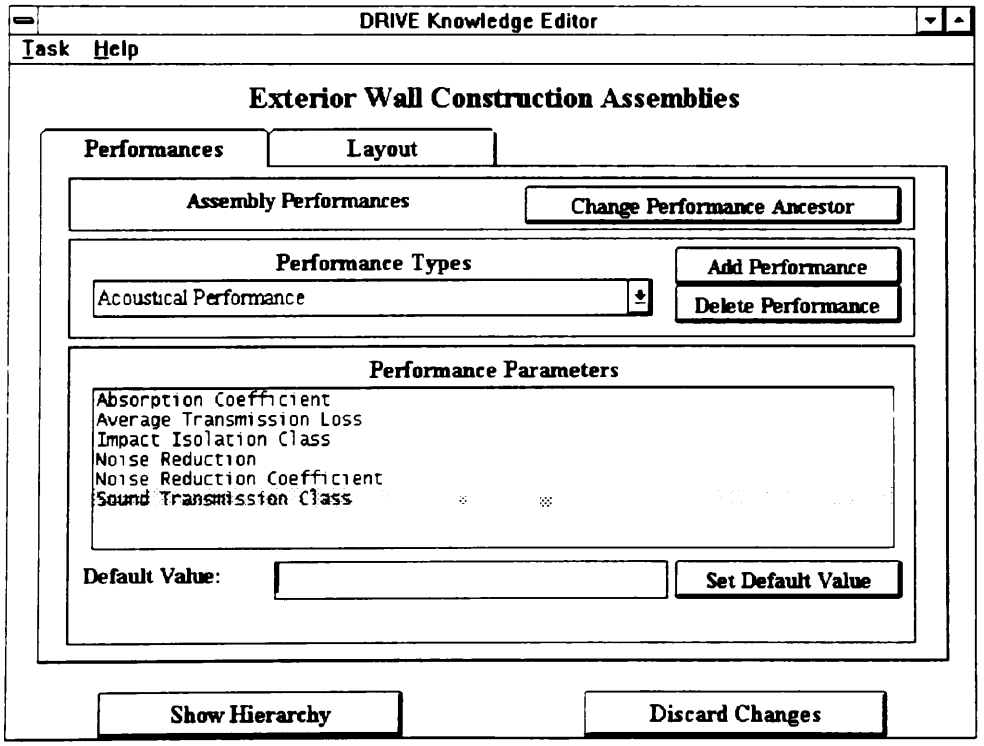

DRIVE Knowledge Editor
Task Help
Acoustical Performance
Performance Parameters
Absorption Coefficient
Average Transmission Loss
Impact Isolation Class
Noise Reduction
Noise Reduction Coefficient
Sound Transmission Class
New Parameter
Delete Parameter
Parameter Name:
Cardinality
Single
Multiple
Value Type
Text
Number
Create Parameter
Show Hierarchy
Discard Changes

Demo: Domain Knowledge Editor

- Add a new parameter Resonance Frequency to Acoustical Performance (Architectural Assemblies).
- Attach Performances to Exterior Wall Construction Assemblies and define applicable default Performance Parameter values.




DRIVE Knowledge Editor
Task Help
Exterior Wall Construction Assemblies
Other Performances
Acoustical Performance
Axial Element Performance
Economic Performances
Electrical Assembly Performance
Flexural Element Performance
Frame Element Performance
Hygrothermal Performance
Mechanical Assembly Performan
Current Performances
Previous Screen


DRIVE Knowledge Editor
Task Help
Exterior Wall Construction Assemblies
Performances
Layout
Assembly Performances
Change Performance Ancestor
Performance Types
Acoustical Performance
Add Performance
Delete Performance
Performance Parameters
Absorption Coefficient
Average Transmission Loss
Impact Isolation Class
Noise Reduction
Noise Reduction Coefficient
Sound Transmission Class
Default Value:
Set Default Value
Show Hierarchy
Discard Changes

Discussion: Domain Knowledge Editor

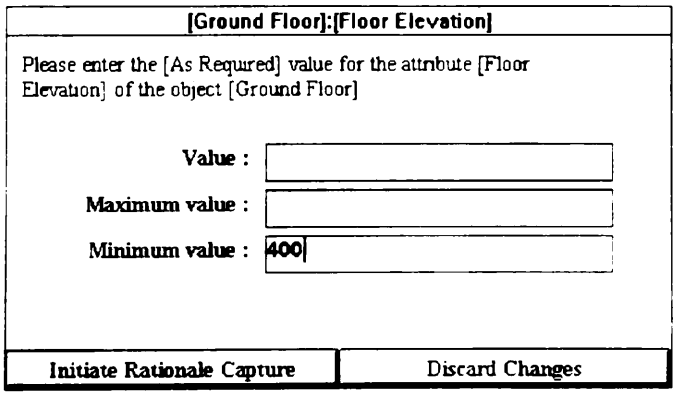
- Performances "groups" related Performance Parameters together
- Domain objects "get" parameters from Performances
- Domain Knowledge Editor allows users to easily modify and expand the database without having to go through the rigors of computer programming

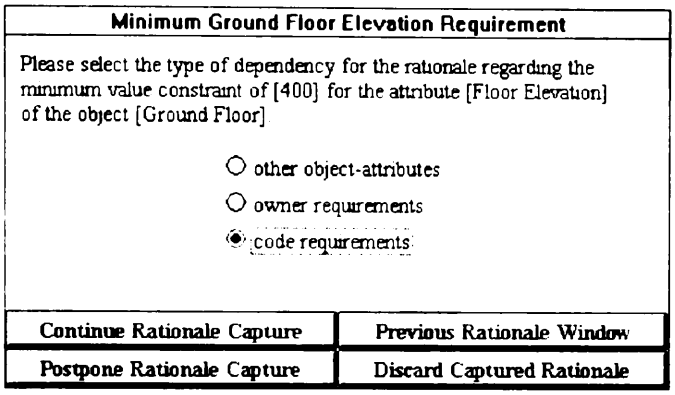
DRIVE Demonstration

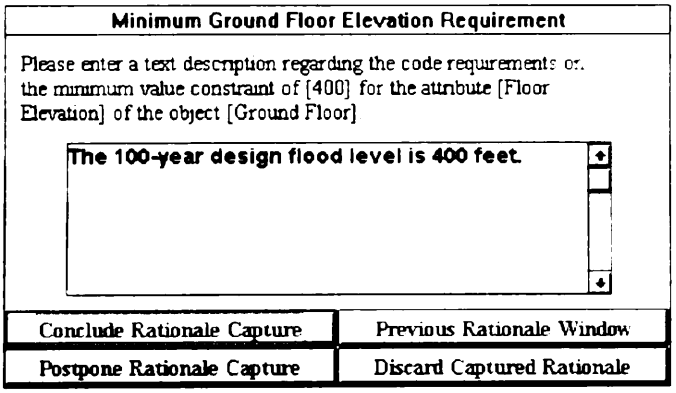
- Domain Knowledge Editor
- Design Rationale Capture

Demo: Design Rationale Capture

- The design rationale for the minimum Floor Elevation of 400 feet stems from the fact that the 100-year design flood level is 400 feet.

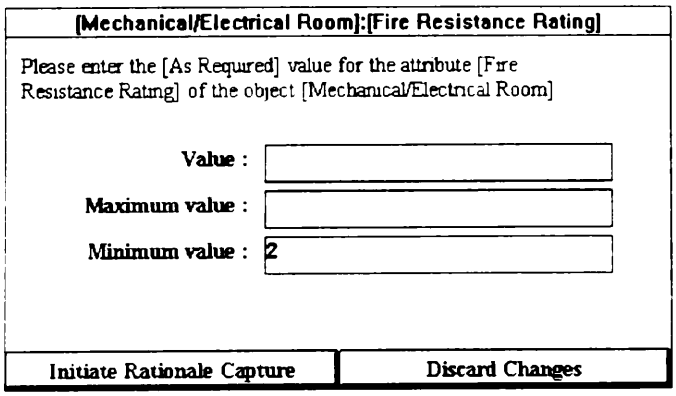


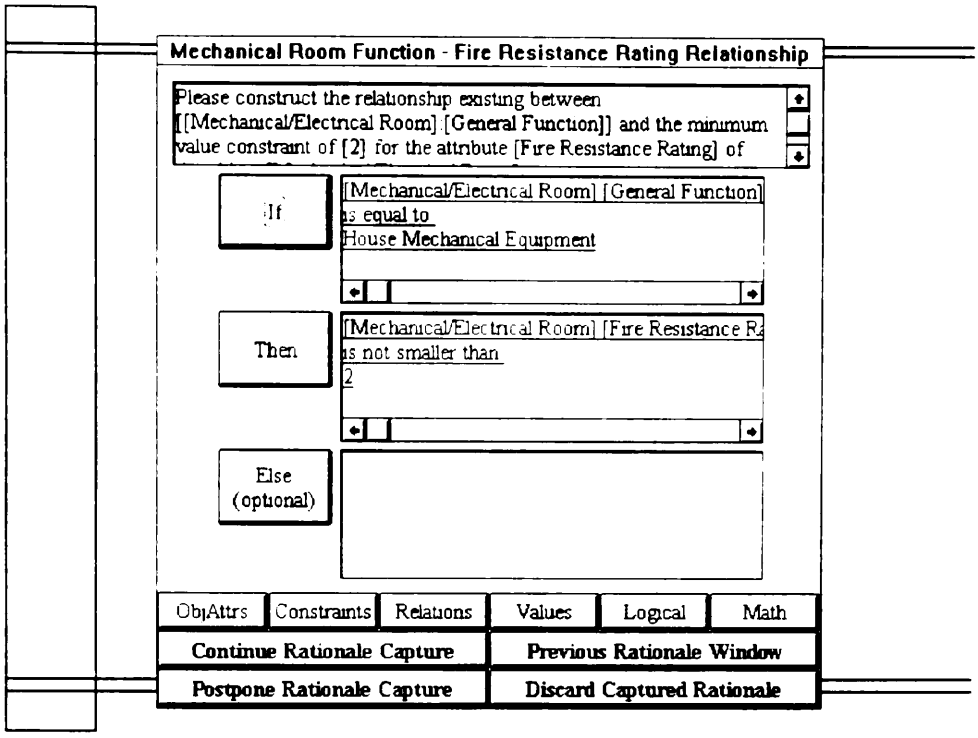

Minimum Ground Floor Elevation Requirement
Please select the type of dependency for the rationale regarding the minimum value constraint of [400] for the attribute [Floor Elevation] of the object [Ground Floor]
other object-attributes
owner requirements
code requirements
Continue Rationale Capture
Previous Rationale Window
Postpone Rationale Capture
Discard Captured Rationale

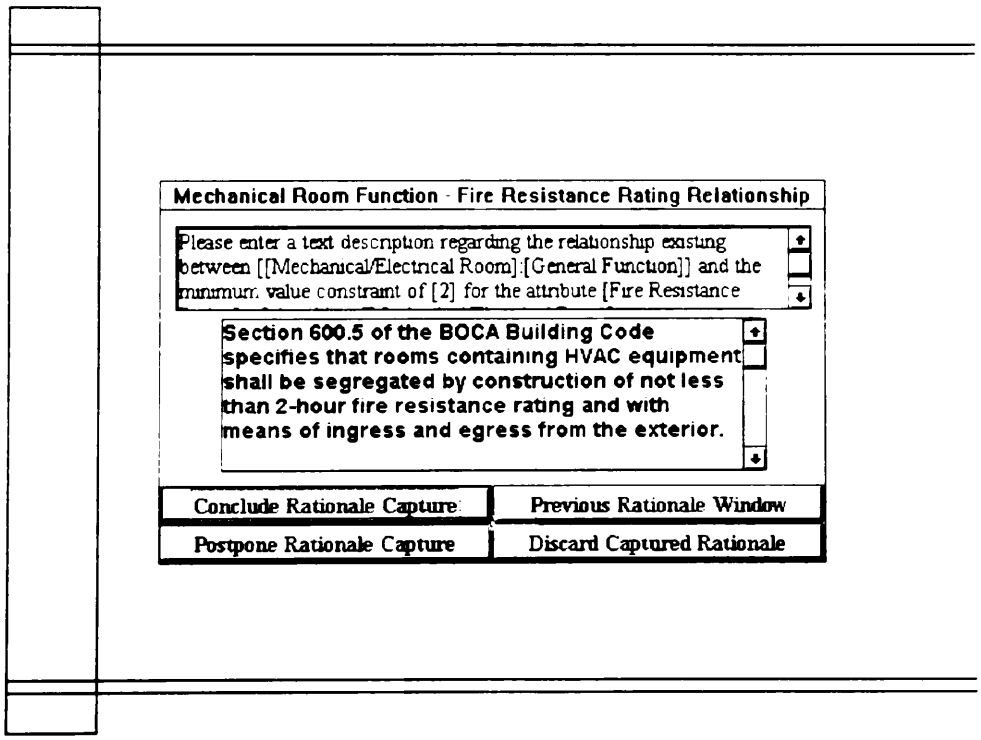

Minimum Ground Floor Elevation Requirement
Please enter a text description regarding the code requirements or the minimum value constraint of [400] for the attribute [Floor Elevation] of the object [Ground Floor]
The 100-year design flood level is 400 feet.
Conclude Rationale Capture
Previous Rationale Window
Postpone Rationale Capture
Discard Captured Rationale

Demo: Design Rationale Capture

- The design rationale for the minimum Floor Elevation of 400 feet stems from the fact that the 100-year design flood level is 400 feet.
- The design rationale for the minimum value of 2 hours for the Fire Resistance Rating of the Mechanical/Electrical Room stems from the fact that if the function of a room is to House Mechanical Equipment, then the Fire Resistance Rating of the room must be at least 2 hours.

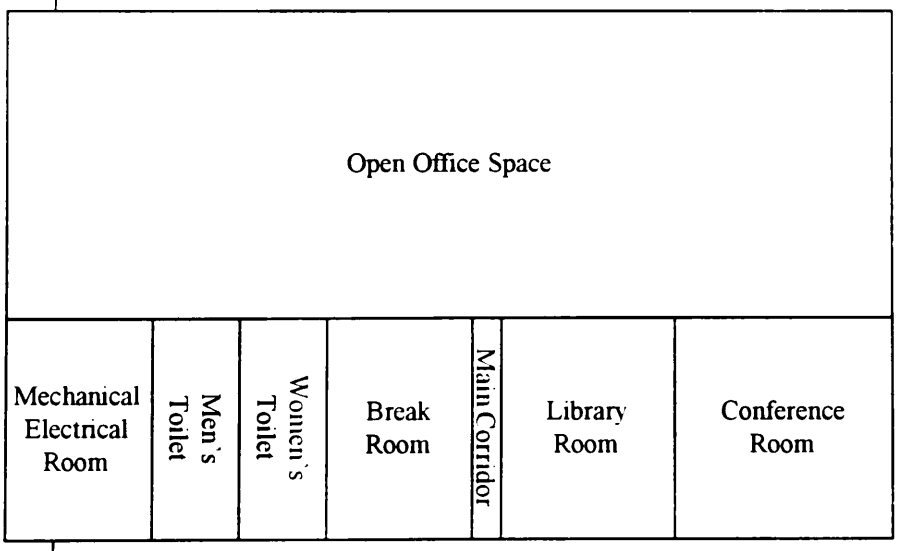


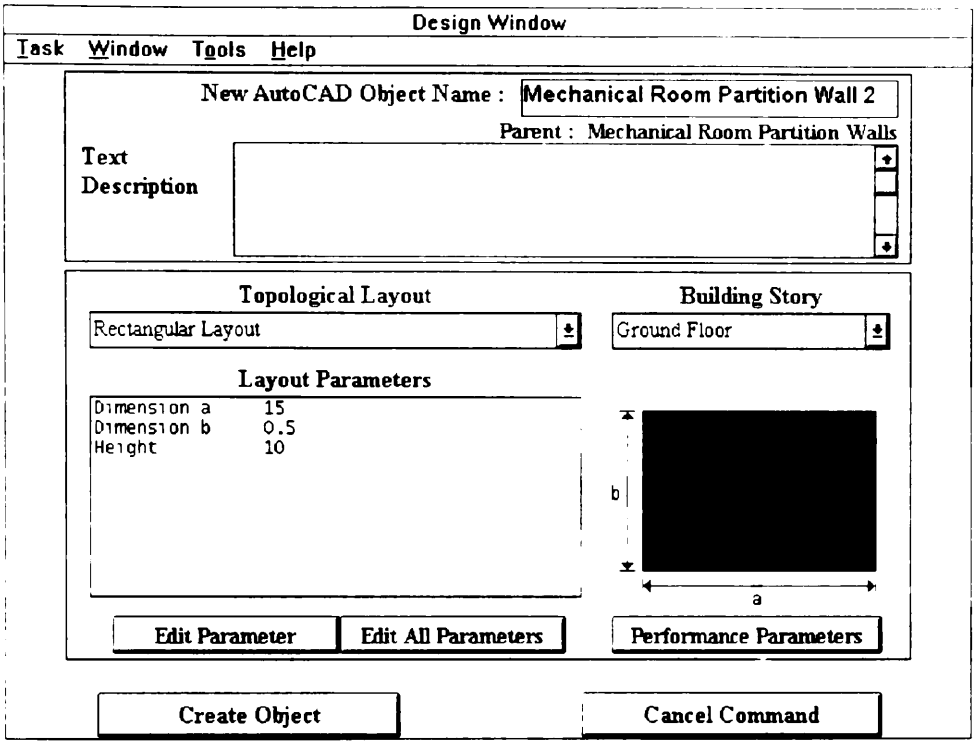

Mechanical Room Function - Fire Resistance Rating Relationship
Please construct the relationship existing between [[Mechanical/Electrical Room]:[General Function]] and the minimum value constraint of [2] for the attribute [Fire Resistance Rating] of
If
[Mechanical/Electrical Room] [General Function]
is equal to
House Mechanical Equipment
Then
[Mechanical/Electrical Room] [Fire Resistance R
is not smaller than
2
Else (optional)
ObjAttrs
Constraints
Relations
Values
Logical
Math
Continue Rationale Capture
Previous Rationale Window
Postpone Rationale Capture
Discard Captured Rationale

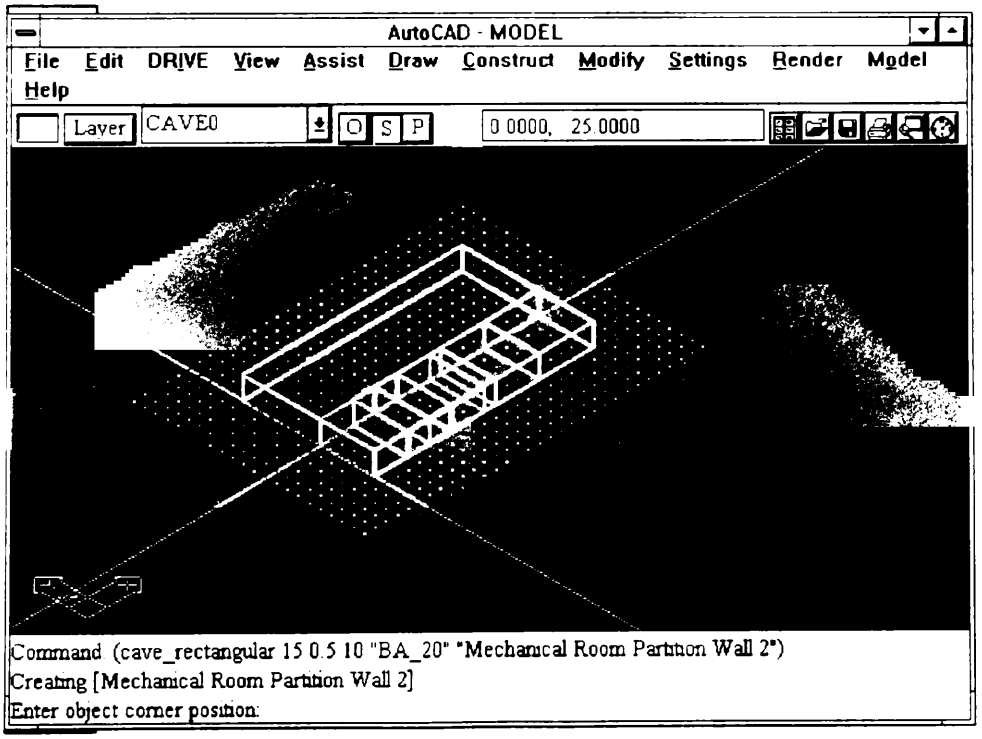

Mechanical Room Function - Fire Resistance Rating Relationship
Please enter a text description regarding the relationship existing between [[Mechanical/Electrical Room]:[General Function]] and the minimum value constraint of [2] for the attribute [Fire Resistance
Section 600.5 of the BOCA Building Code specifies that rooms containing HVAC equipment shall be segregated by construction of not less than 2-hour fire resistance rating and with means of ingress and egress from the exterior.
Conclude Rationale Capture
Previous Rationale Window
Postpone Rationale Capture
Discard Captured Rationale

Demo: Design Rationale Capture

- Create a Mechanical Room Partition Wall 2 separating the Mechanical/Electrical Room and the Open Office Space.

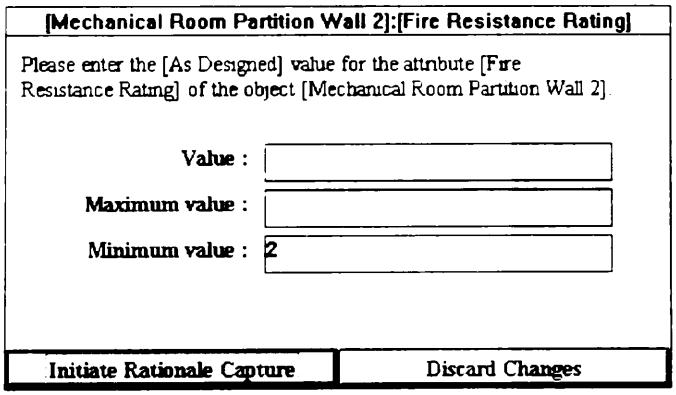


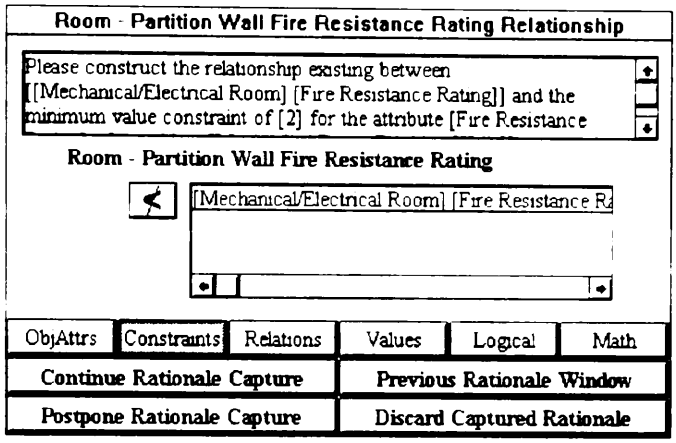

Design Window
Task Window Tools Help
New AutoCAD Object Name : Mechanical Room Partition Wall 2
Parent : Mechanical Room Partition Walls
Text Description
Topological Layout
Rectangular Layout
Building Story
Ground Floor
Layout Parameters
Dimension a 15
Dimension b 0.5
Height 10
b
a
Edit Parameter
Edit All Parameters
Performance Parameters
Create Object
Cancel Command

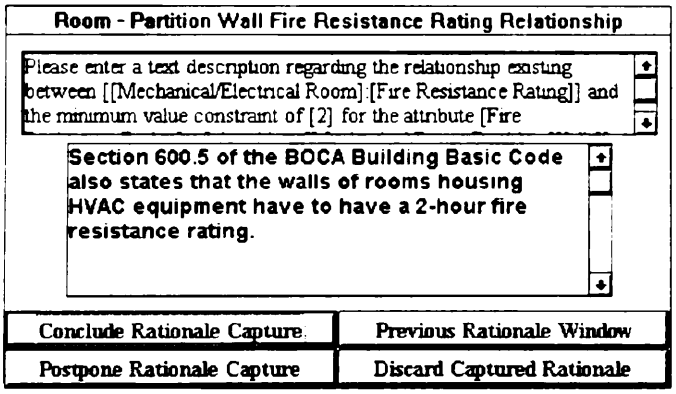

AutoCAD - MODEL
File Edit DRIVE View Assist Draw Construct Modify Settings Render Model
Help
Layer
CAVE0
O S P
0.0000, 25.0000
Command (cave_rectangular 15 0.5 10 "BA_20" "Mechanical Room Partition Wall 2")
Creating [Mechanical Room Partition Wall 2]
Enter object corner position:

Demo: Design Rationale Capture

- Create a Mechanical Room Partition Wall 2 separating the Mechanical/Electrical Room and the Open Office Space.
- The design rationale for the minimum value of 2 hours for the Fire Resistance Rating of the Mechanical Room Partition Wall 2 stems from the relationship that the minimum value of Fire Resistance Rating for the Mechanical Room Partition Wall 2 must be at least equal to that of the Mechanical/Electrical Room.




Room - Partition Wall Fire Resistance Rating Relationship
Please construct the relationship existing between [[Mechanical/Electrical Room] [Fire Resistance Rating]] and the minimum value constraint of [2] for the attribute [Fire Resistance
Room - Partition Wall Fire Resistance Rating
[Mechanical/Electrical Room] [Fire Resistance Ra
ObjAttrs
Constraints
Relations
Values
Logical
Math
Continue Rationale Capture
Previous Rationale Window
Postpone Rationale Capture
Discard Captured Rationale


Room - Partition Wall Fire Resistance Rating Relationship
Please enter a text description regarding the relationship existing between [[Mechanical/Electrical Room]:[Fire Resistance Rating]] and the minimum value constraint of [2] for the attribute [Fire
Section 600.5 of the BOCA Building Basic Code also states that the walls of rooms housing HVAC equipment have to have a 2-hour fire resistance rating.
Conclude Rationale Capture
Previous Rationale Window
Postpone Rationale Capture
Discard Captured Rationale

Discussion: Design Rationale Capture

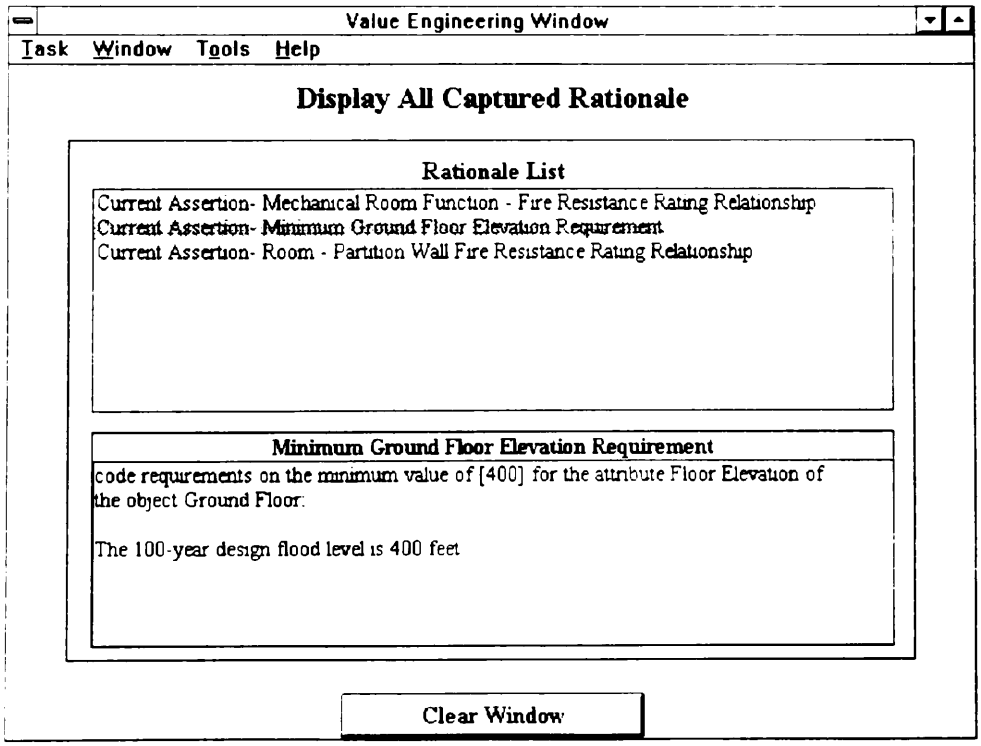
- Three types of Design Rationale
  - Simple Text
  - Logical (If-Then-Else) Relationship
  - Mathematical Relationship
- Simple Text Design Rationale is unstructured
- Logical and Mathematical Relationships are structured forms of Design Rationale

DRIVE Demonstration

- Domain Knowledge Editor
- Design Rationale Capture
- Design Rationale Retrieval

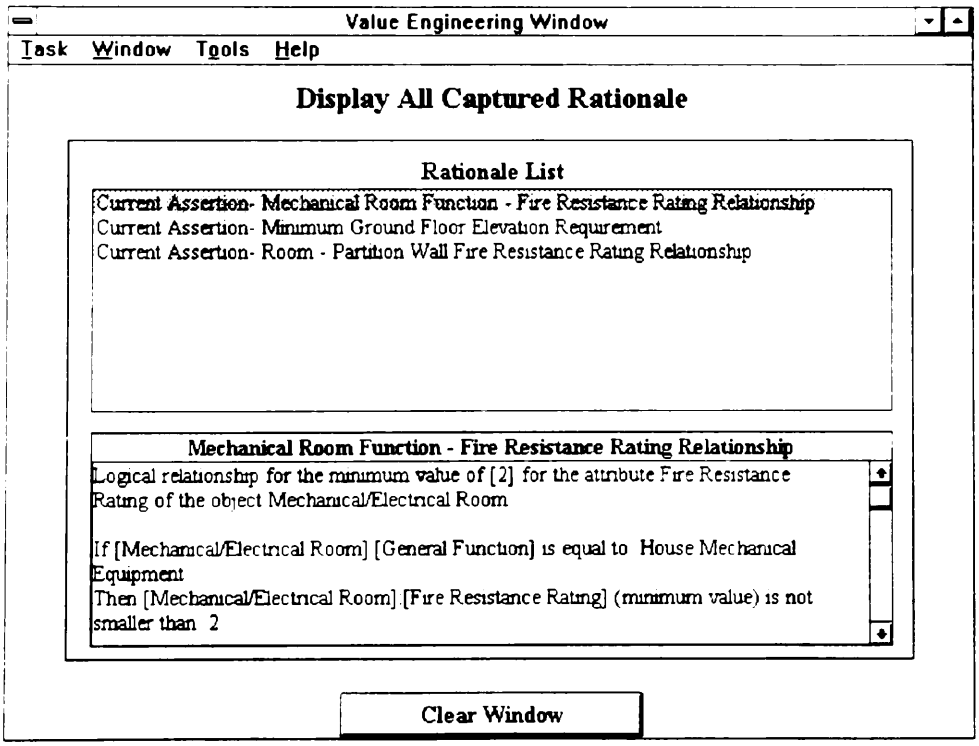
Demo: Design Rationale Retrieval

- Retrieve the design rationale for the minimum Floor Elevation of 400 feet.



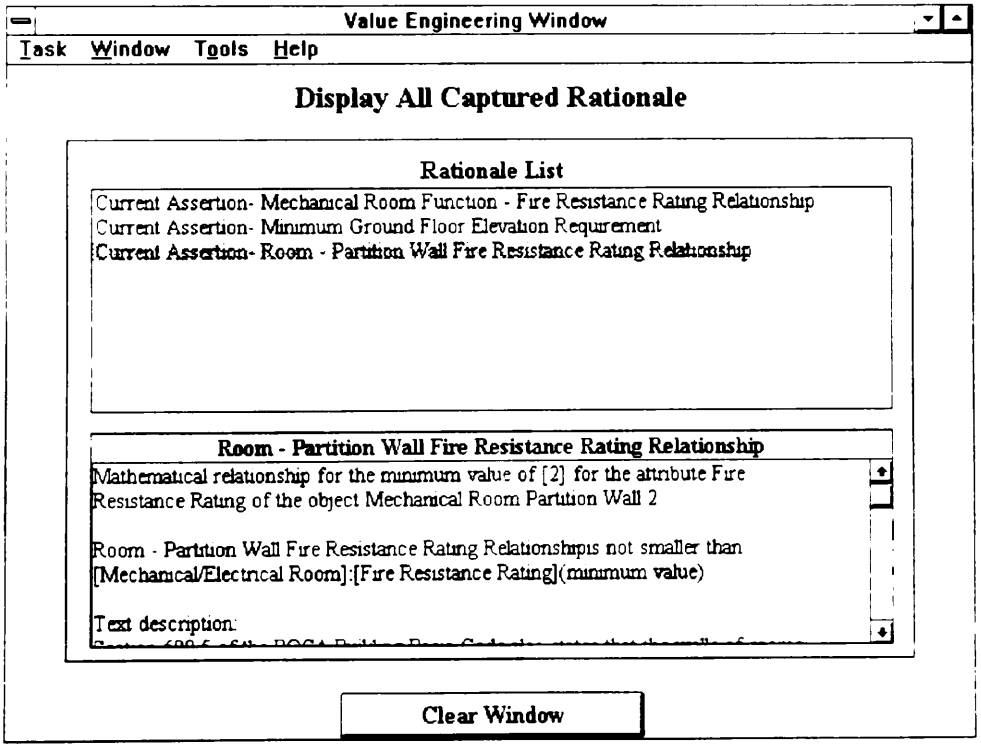
Demo: Design Rationale Retrieval

- Retrieve the design rationale for the minimum Floor Elevation of 400 feet.
- Retrieve the design rationale for the relationship existing between the General Function and the Fire Resistance Rating of the Mechanical Room.



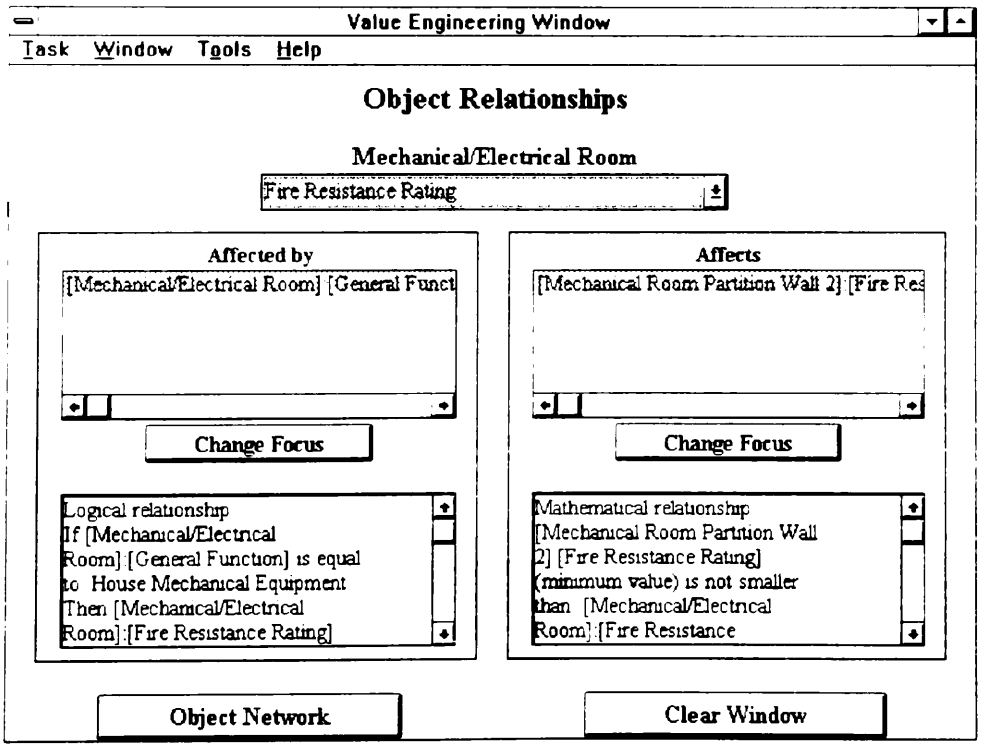
Demo: Design Rationale Retrieval

- Retrieve the design rationale for the minimum Floor Elevation of 400 feet.
- Retrieve the design rationale for the relationship existing between the General Function and the Fire Resistance Rating of the Mechanical Room.
- Retrieve the design rationale for the relationship existing between the Fire Resistance Rating attributes of the object Mechanical Room and the object Mechanical Room Partition Wall 2.



Demo: Design Rationale Retrieval

- Display the Parameter Dependency Network for the attribute Fire Resistance Rating of the object Mechanical Room.



Discussion: Design Rationale Retrieval

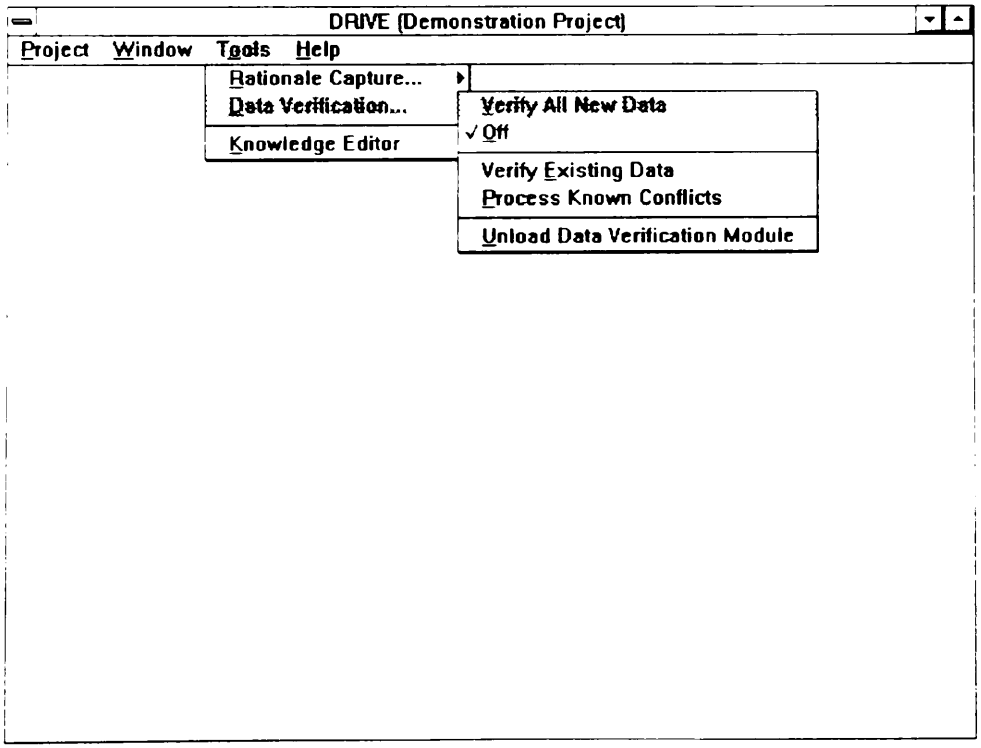
- Displays captured Design Rationale in "natural language" text
- Suitable for increasing the user's understanding of the design

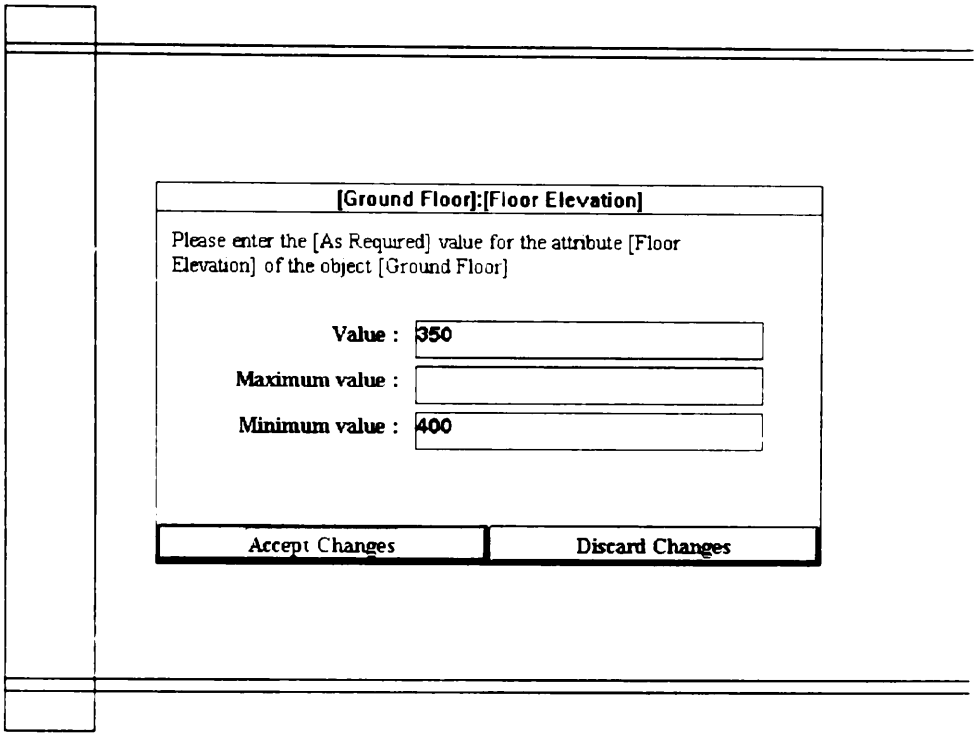
DRIVE Demonstration

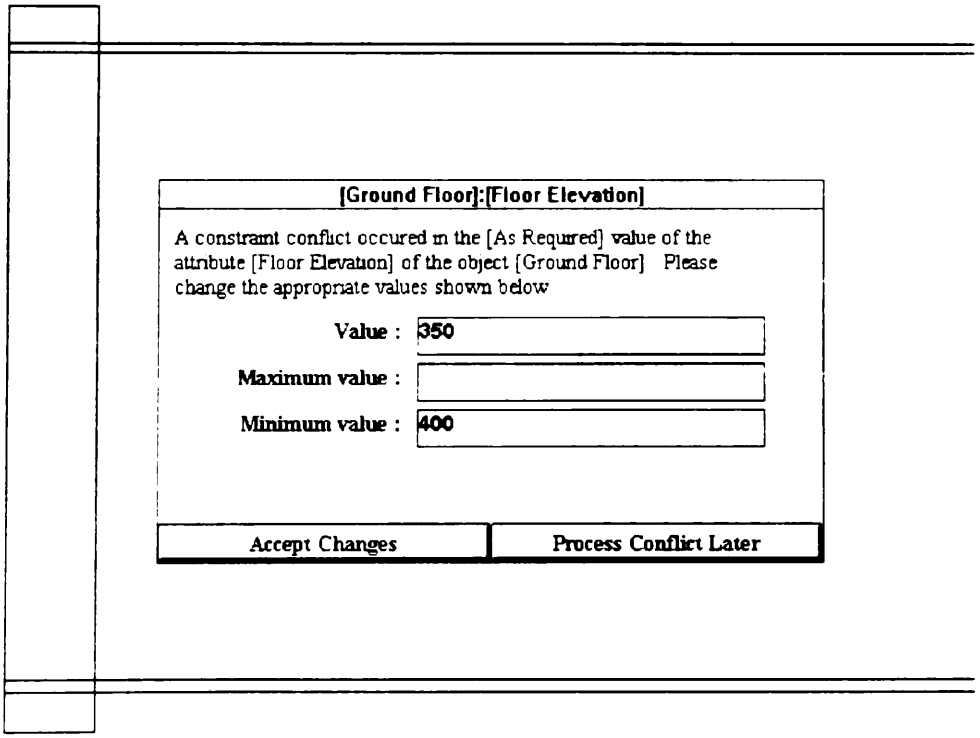
- Domain Knowledge Editor
- Design Rationale Capture
- Design Rationale Retrieval
- Data Verification and Conflict Resolution

Demo
Data Verification and Conflict Resolution

- Switch on the Data Verification module and trigger the Conflict Resolution module by violating the minimum value constraint for the Ground Floor Elevation.

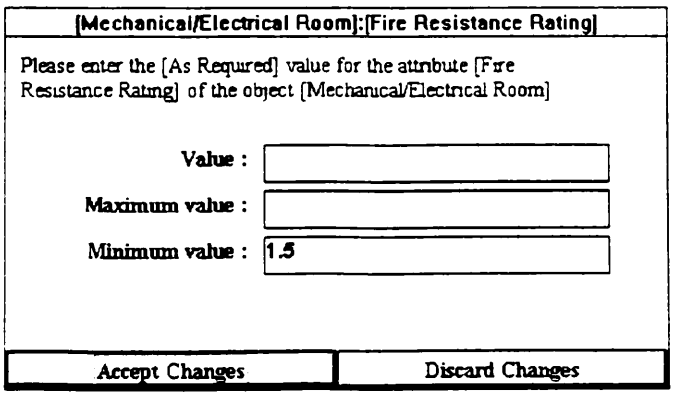


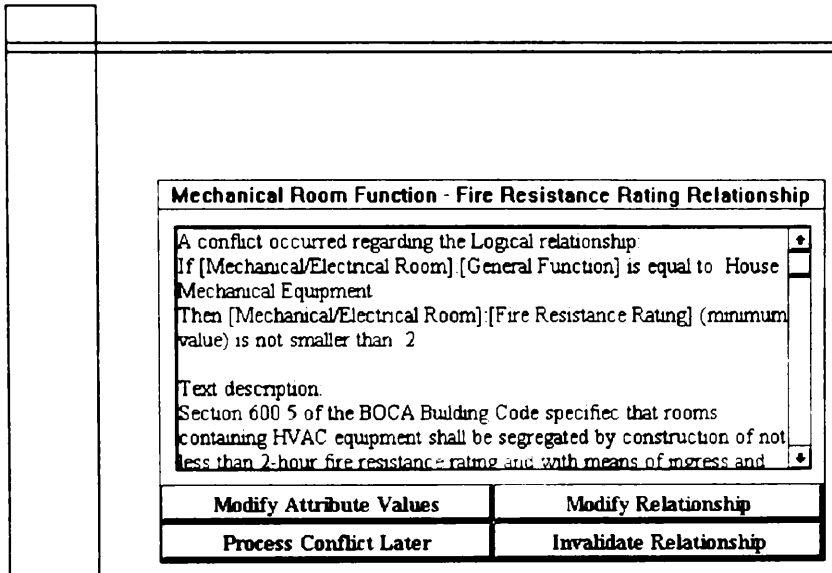

[Ground Floor]:[Floor Elevation]
Please enter the [As Required] value for the attribute [Floor Elevation] of the object [Ground Floor]
Value : 350
Maximum value :
Minimum value : 400
Accept Changes
Discard Changes

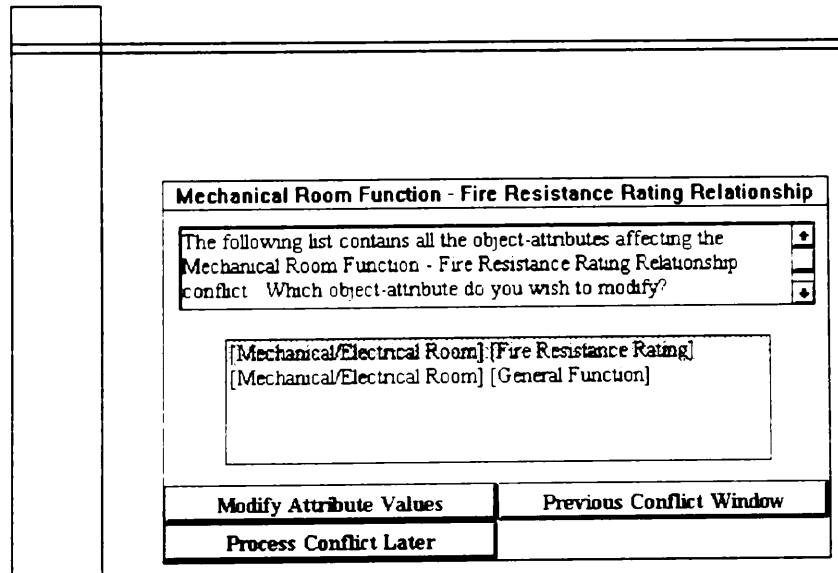

[Ground Floor]:[Floor Elevation]
A constraint conflict occured in the [As Required] value of the attribute [Floor Elevation] of the object [Ground Floor] Please change the appropriate values shown below
Value : 350
Maximum value :
Minimum value : 400
Accept Changes
Process Conflict Later

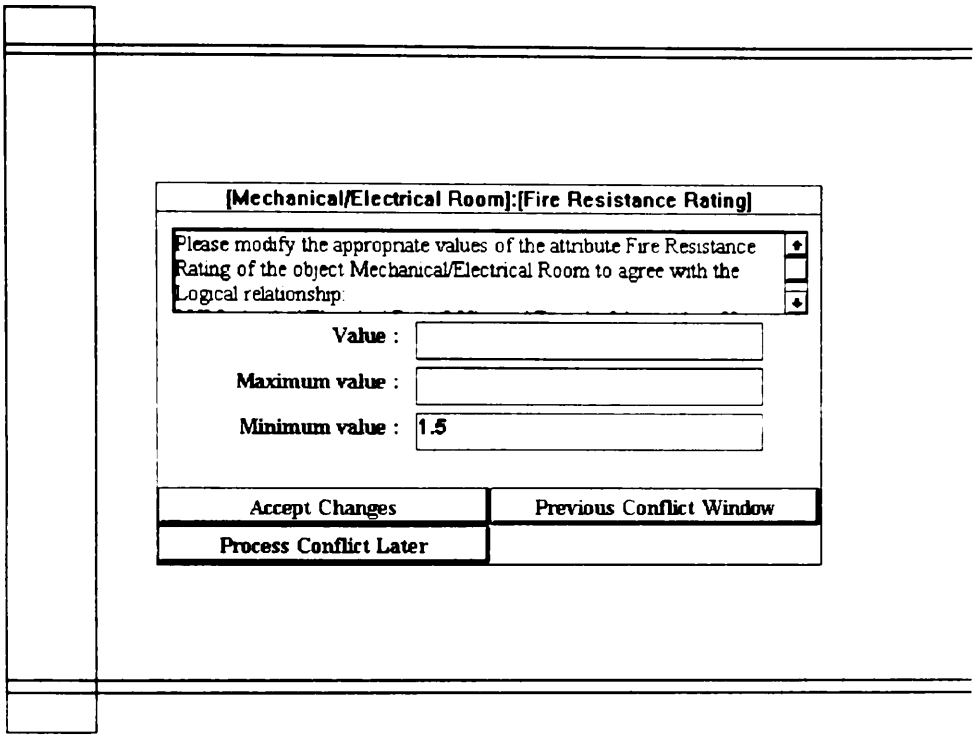
Demo
Data Verification and Conflict Resolution

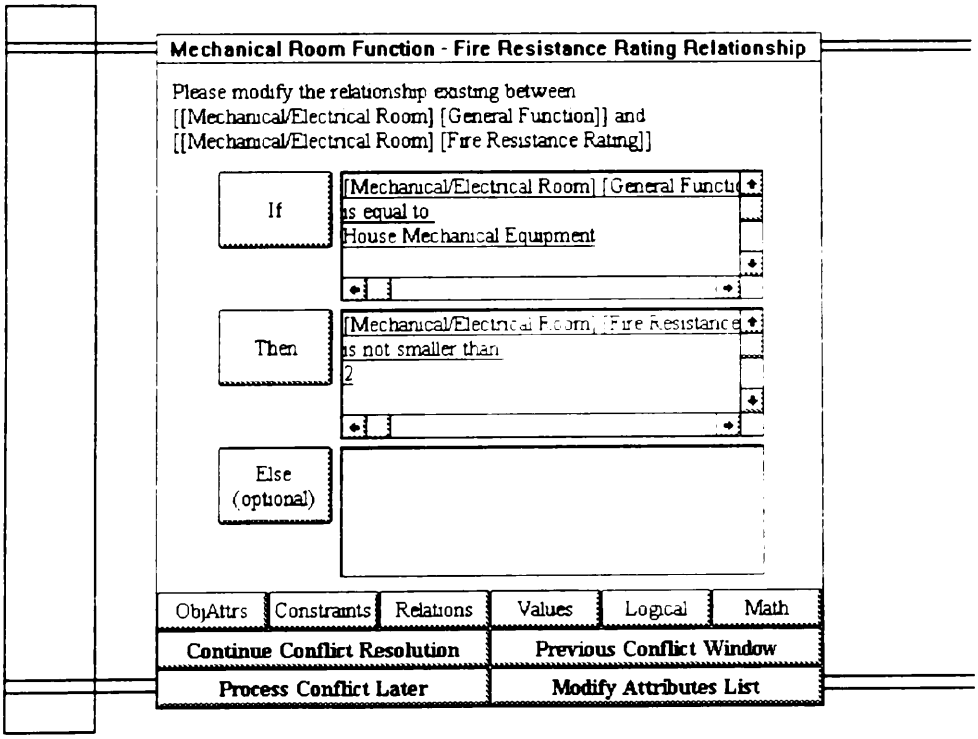
- Switch on the Data Verification module and trigger the Conflict Resolution module by violating the minimum value constraint for the Ground Floor Elevation.
- Trigger the Conflict Resolution module by violating the Mechanical Room Function - Fire Resistance Rating Relationship.




Mechanical Room Function - Fire Resistance Rating Relationship
A conflict occurred regarding the Logical relationship:
If [Mechanical/Electrical Room].[General Function] is equal to House Mechanical Equipment
Then [Mechanical/Electrical Room]:[Fire Resistance Rating] (minimum value) is not smaller than 2
Text description:
Section 600 5 of the BOCA Building Code specifies that rooms containing HVAC equipment shall be segregated by construction of not less than 2-hour fire resistance rating and with means of egress and
Modify Attribute Values
Modify Relationship
Process Conflict Later
Invalidate Relationship


Mechanical Room Function - Fire Resistance Rating Relationship
The following list contains all the object-attributes affecting the Mechanical Room Function - Fire Resistance Rating Relationship conflict. Which object-attribute do you wish to modify?
[Mechanical/Electrical Room]:[Fire Resistance Rating]
[Mechanical/Electrical Room] [General Function]
Modify Attribute Values
Previous Conflict Window
Process Conflict Later


[Mechanical/Electrical Room]:[Fire Resistance Rating]
Please modify the appropriate values of the attribute Fire Resistance Rating of the object Mechanical/Electrical Room to agree with the Logical relationship:
Value :
Maximum value :
Minimum value :
1.5
Accept Changes
Previous Conflict Window
Process Conflict Later


Mechanical Room Function - Fire Resistance Rating Relationship
Please modify the relationship existing between
[[Mechanical/Electrical Room] [General Function]] and
[[Mechanical/Electrical Room] [Fire Resistance Rating]]
If
[Mechanical/Electrical Room] [General Functi
is equal to
House Mechanical Equipment
Then
[Mechanical/Electrical Room] [Fire Resistance
is not smaller than
2
Else
(optional)
ObjAttrs
Constraints
Relations
Values
Logical
Math
Continue Conflict Resolution
Previous Conflict Window
Process Conflict Later
Modify Attributes List

Discussion
Data Verification and Conflict Resolution

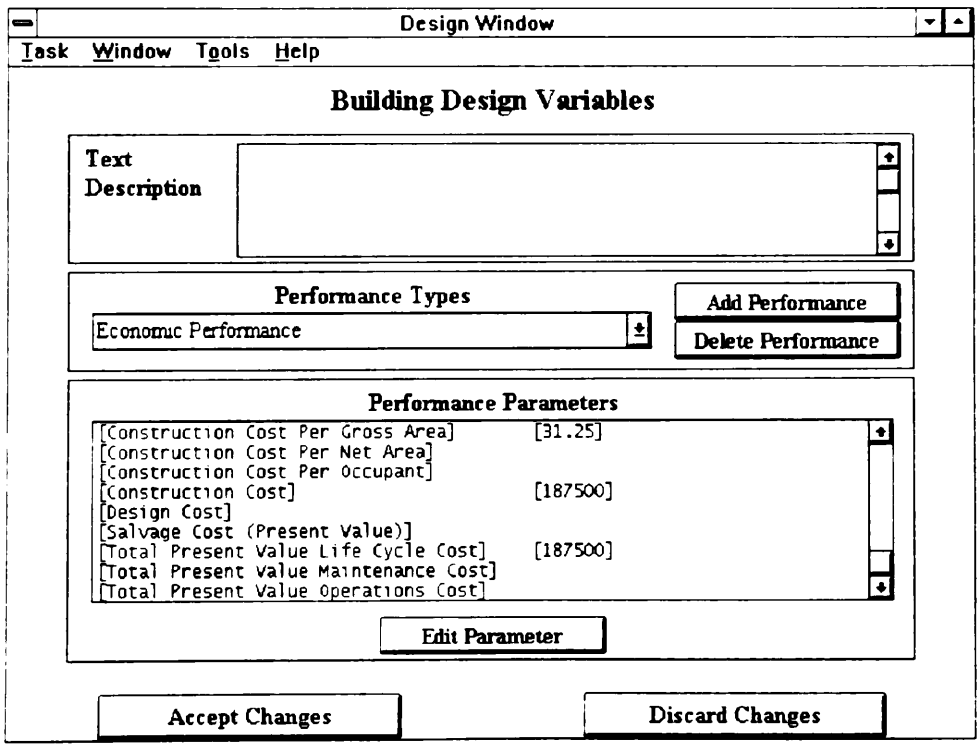
- Computer program acts as a design assistant maintaining consistency between the object database and the rationale database
- Computer program performs operations on the captured design rationale

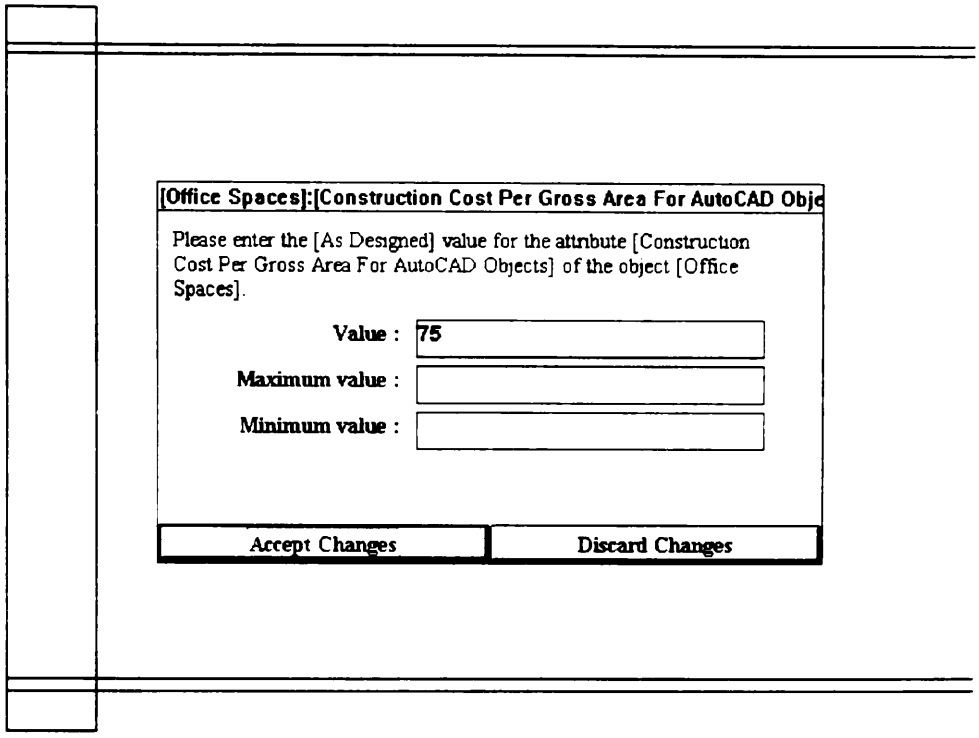
DRIVE Demonstration

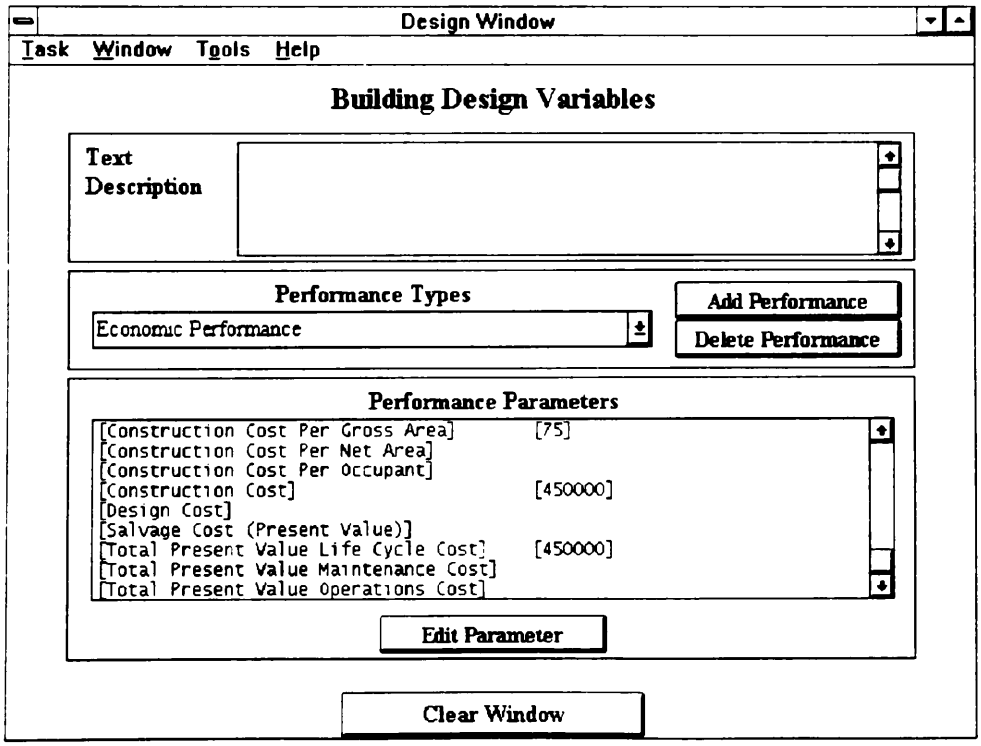
- Domain Knowledge Editor
- Design Rationale Capture
- Design Rationale Retrieval
- Data Verification and Conflict Resolution
- Value Engineering Diagrams

Demo: Value Engineering Diagrams

- Set the value of the attribute Construction Cost Per Gross Area For AutoCAD Objects for the object Office Spaces to 75.

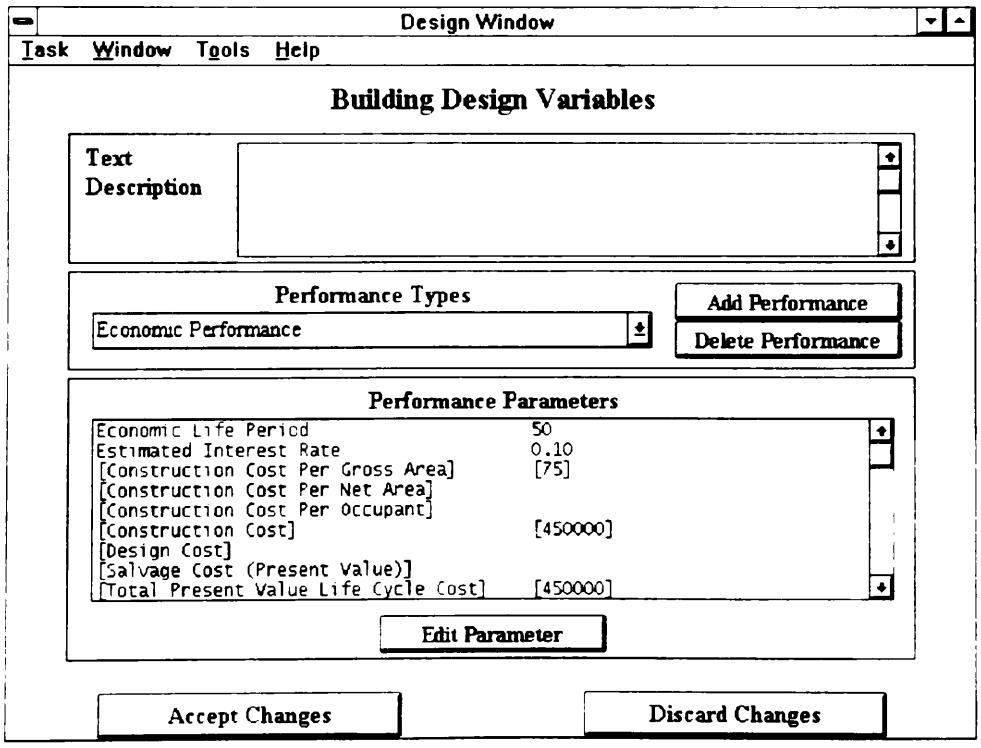



[Office Spaces]:[Construction Cost Per Gross Area For AutoCAD Obje
Please enter the [As Designed] value for the attribute [Construction Cost Per Gross Area For AutoCAD Objects] of the object [Office Spaces].
Value :
75
Maximum value :
Minimum value :
Accept Changes
Discard Changes


Design Window
Task Window Tools Help
Building Design Variables
Text Description
Performance Types
Economic Performance
Add Performance
Delete Performance
Performance Parameters
[Construction Cost Per Gross Area] [75]
[Construction Cost Per Net Area]
[Construction Cost Per Occupant]
[Construction Cost] [450000]
[Design Cost]
[Salvage Cost (Present Value)]
[Total Present Value Life Cycle Cost] [450000]
[Total Present Value Maintenance Cost]
[Total Present Value Operations Cost]
Edit Parameter
Clear Window

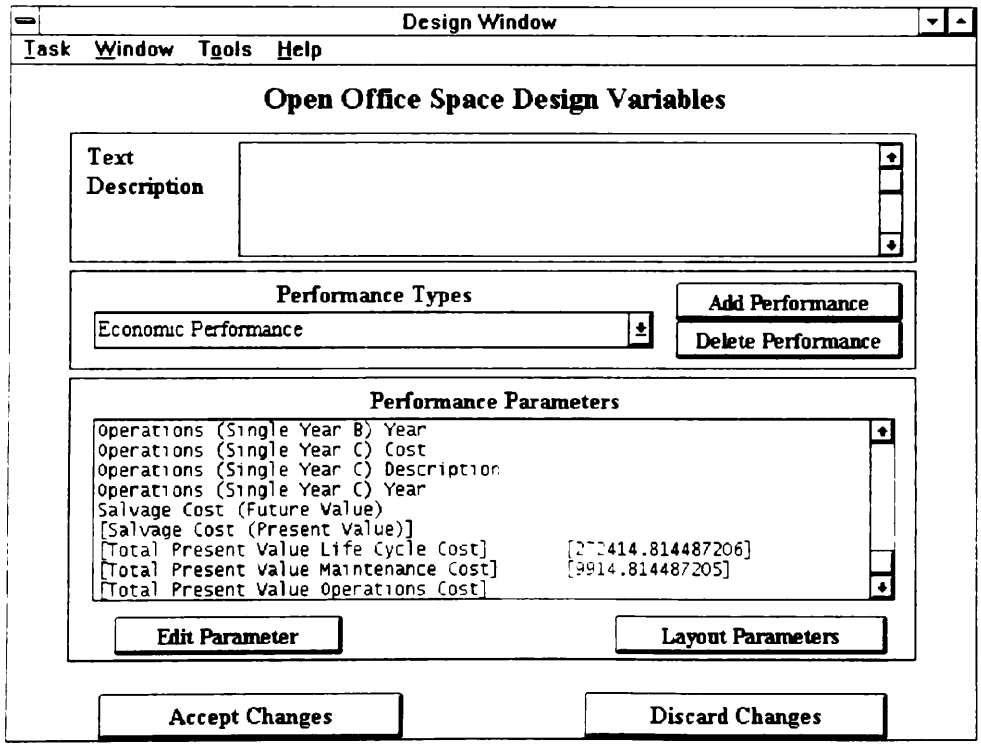
Demo: Value Engineering Diagrams

- Set the value of the attribute Construction Cost Per Gross Area For AutoCAD Objects for the object Office Spaces to 75.
- Set the Economic Life Period to 50 years and the Estimated Interest Rate to 0.10.



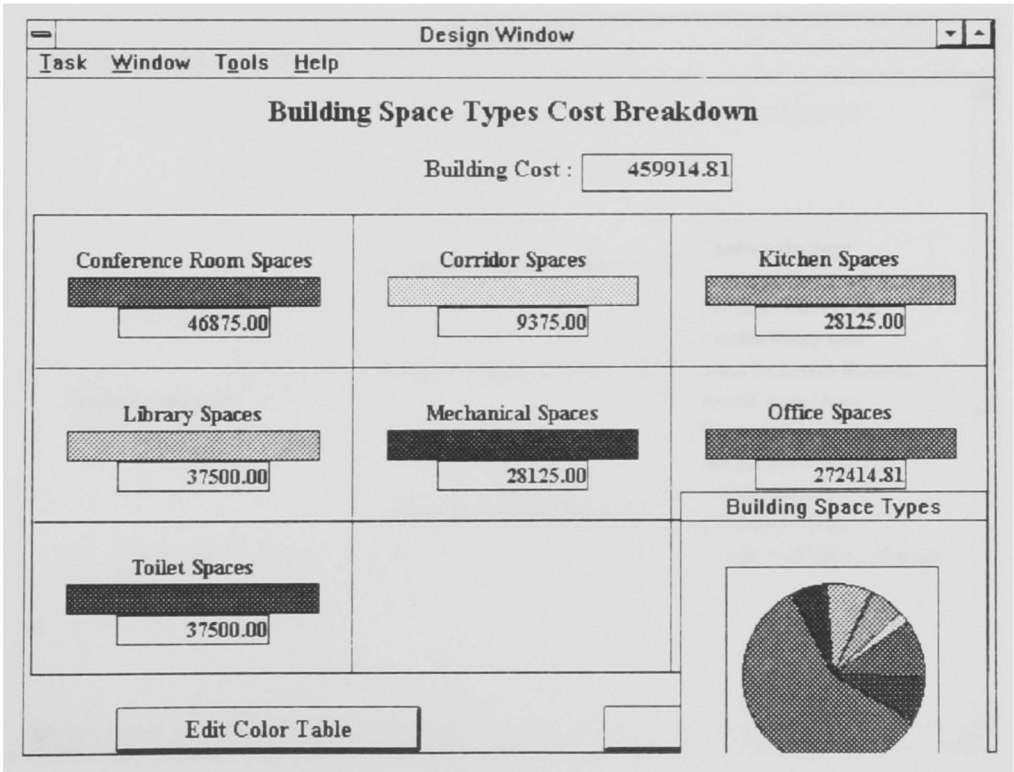
Demo: Value Engineering Diagrams

- Set the value of the attribute Construction Cost Per Gross Area For AutoCAD Objects for the object Office Spaces to 75.
- Set the Economic Life Period to 50 years and the Estimated Interest Rate to 0.10.
- Set the Maintenance (Annual A) Description to Annual General Cleaning and the Maintenance (Annual A) Cost to 1000 for the Open Office Space.



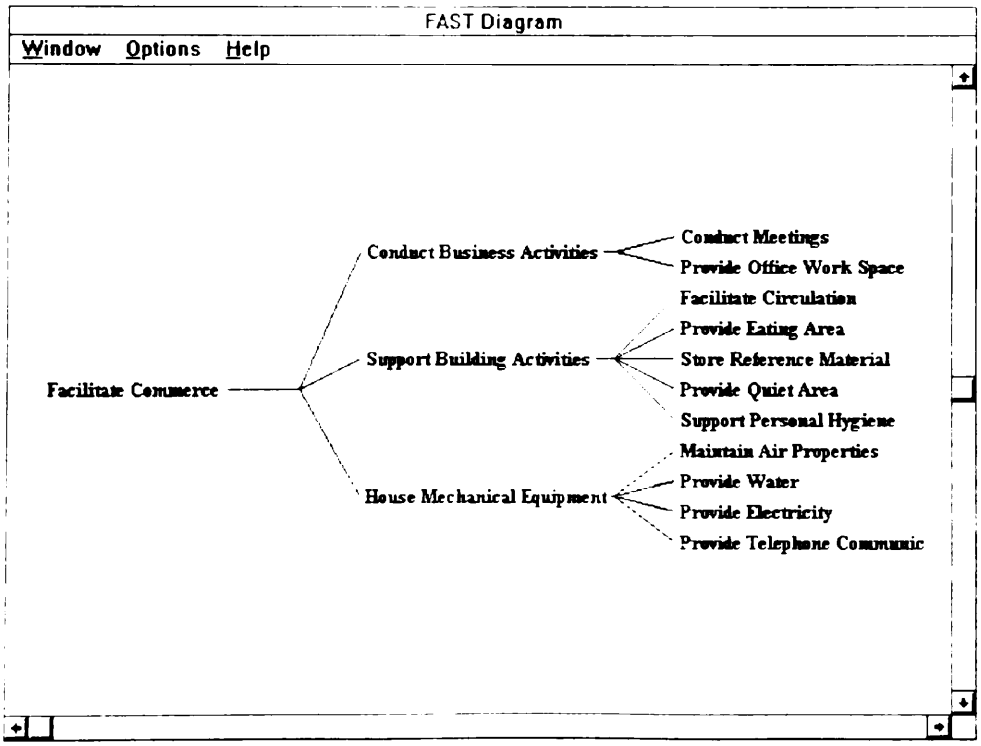
Demo: Value Engineering Diagrams

- Display and navigate the Life Cycle Cost Breakdown Diagram for the various space types.



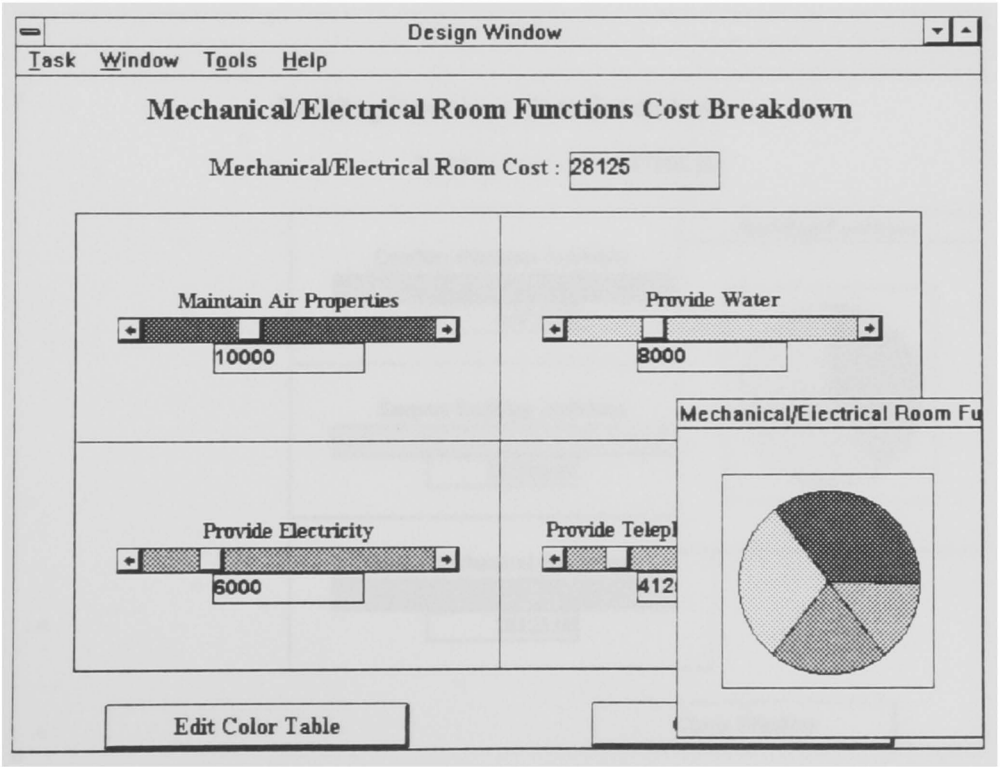
Demo: Value Engineering Diagrams

- Display and navigate the Life Cycle Cost Breakdown Diagram for the various space types.
- Display the Functional Analysis Systems Technique (FAST) Diagram.



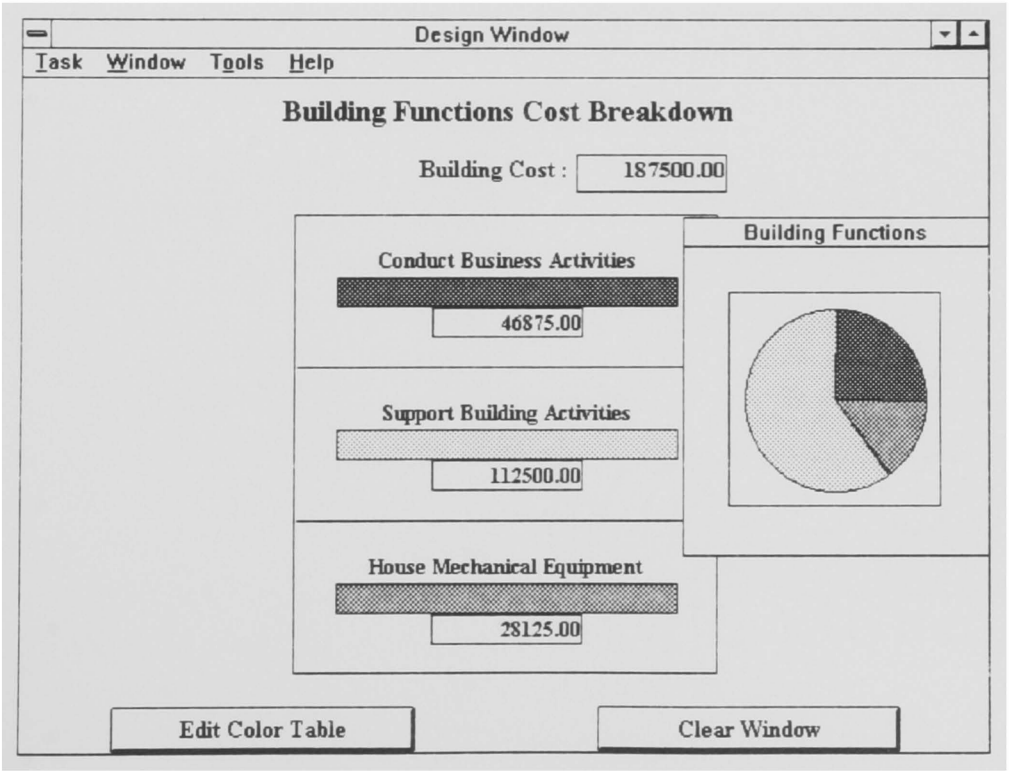
Demo: Value Engineering Diagrams

- Display and navigate the Life Cycle Cost Breakdown Diagram for the various space types.
- Display the Functional Analysis Systems Technique (FAST) Diagram.
- Display the Function Cost Breakdown Diagram for the Mechanical/Electrical Room.



Demo: Value Engineering Diagrams

- Display and navigate the Life Cycle Cost Breakdown Diagram for the various space types.
- Display the Functional Analysis Systems Technique (FAST) Diagram.
- Display the Function Cost Breakdown Diagram for the Mechanical/Electrical Room.
- Display and navigate the Life Cycle Cost Breakdown Diagram for the various functions.



Discussion: Value Engineering Diagrams

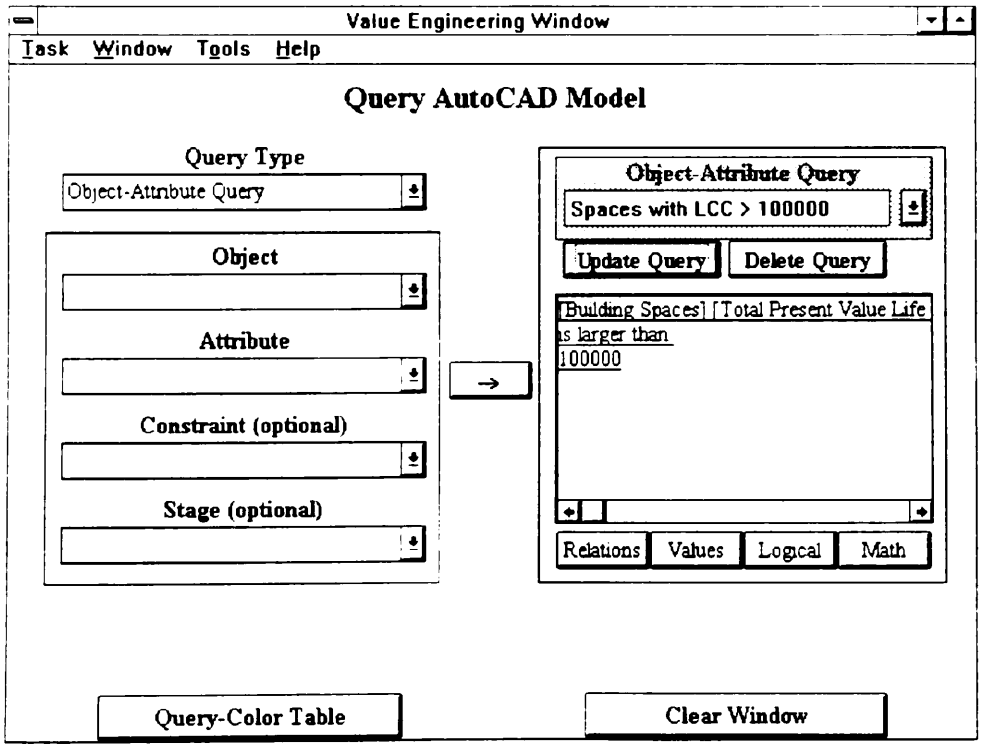
- Supports the following value engineering tasks:
  - Life Cycle Cost Estimating
  - FAST Diagramming
  - Functional Cost Breakdown Generation

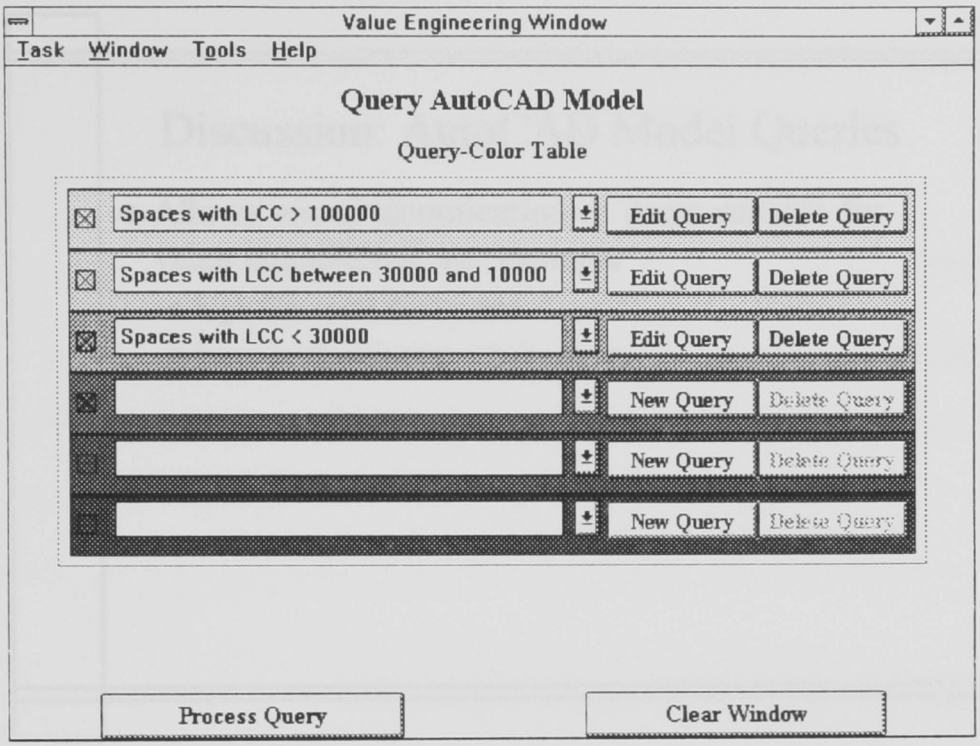
DRIVE Demonstration

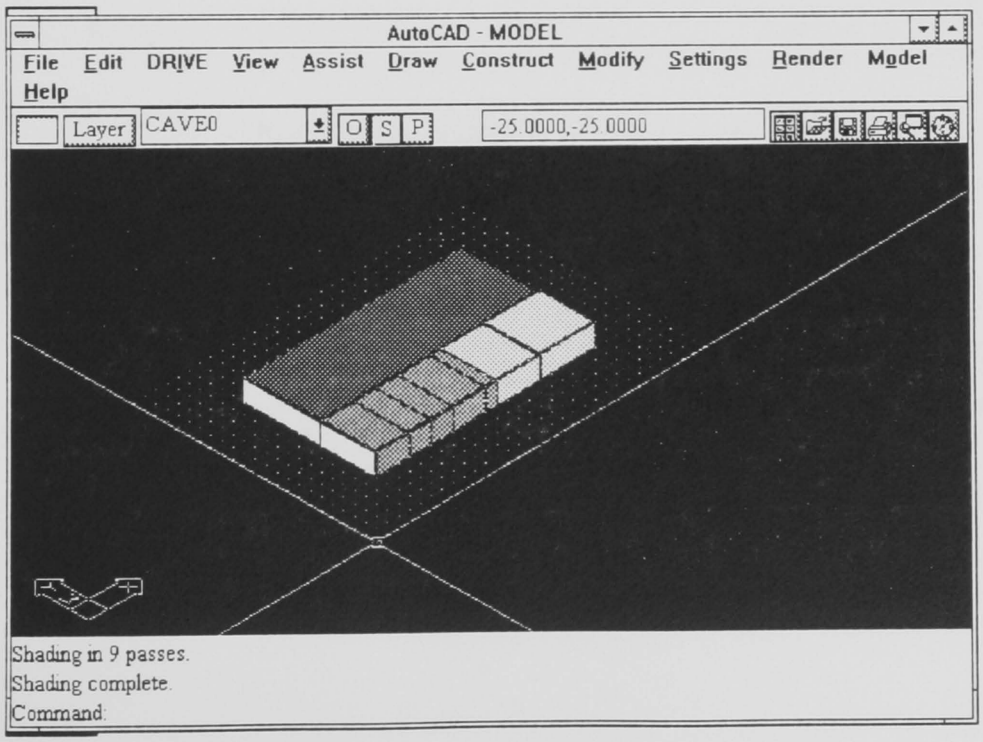
- Domain Knowledge Editor
- Design Rationale Capture
- Design Rationale Retrieval
- Data Verification and Conflict Resolution
- Value Engineering Diagrams
- AutoCAD Model Queries

Demo: AutoCAD Model Queries

- Create a query to display all Spaces having a Life Cycle Cost greater than 100000.




Value Engineering Window
Task Window Tools Help
Query AutoCAD Model
Query-Color Table
Spaces with LCC > 100000
Edit Query
Delete Query
Spaces with LCC between 30000 and 10000
Edit Query
Delete Query
Spaces with LCC < 30000
Edit Query
Delete Query
New Query
New Query
New Query
Process Query
Clear Window


AutoCAD - MODEL
File Edit DRIVE View Assist Draw Construct Modify Settings Render Model
Help
Layer
CAVE0
O S P
-25.0000,-25.0000
Shading in 9 passes.
Shading complete.
Command:

Discussion: AutoCAD Model Queries

- Allows visual identification of items suitable for value engineering, for example
  - high life cycle cost items
  - items with stringent requirements

APPENDIX D

DRIVE SOURCE CODE

VITA

Primo T. Alcantara, Jr. was born on July 28, 1967 in Manila, Philippines. He obtained his Bachelor of Science degree in Civil Engineering from the University of the Philippines, Diliman in 1988. He passed the Philippine Civil Engineering Professional Examination also in 1988. He practiced structural engineering in the Philippines from 1988 to 1991. He then attended the Construction Engineering and Management program of the Department of Civil Engineering of Virginia Tech from 1991 to 1996. This is where he obtained his Master of Science and Doctor of Philosophy degrees in May 1995 and July 1996, respectively.