

**The Primitive-Composite (P-C) Approach-  
A Methodology for Developing Sharable  
Object-Oriented Data Representations  
For Facility Engineering Integration**

by

Dr. D. H. Douglas Phan

Dr. H. Craig Howard

**TECHNICAL REPORT**

**Number 85A**

August, 1993

**Stanford University**

© Copyright by Dung Huu Douglas Phan 1993

All Rights Reserved



# Summary

---

---

## 1. Abstract

This research proposes a new approach for modeling facility engineering processes and data to achieve integration. The so-called *Primitive-Composite (or P-C) Approach* is a structured methodology that can be used to design an *integrated* information management system for a given facility engineering domain. With this methodology, designers can analyze the domain of interest and design an object-oriented schema that can be shared among multiple users across the life-cycle phases in that domain. The P-C Approach was applied to the domain of electrical utility transmission towers and tested for that domain. A database for one specific tower was implemented using a commercial object-oriented database management system (ONTOS).

## 2. Subject

This report describes the Primitive-Composite (or P-C) Approach, a methodology for analyzing a given facility engineering domain and designing a common object-oriented schema for that domain (called a *domain primitive schema* or *primitive schema*). This approach includes four phases: (1) Preliminary Domain Study, (2) Functional Analysis, (3) Domain Entities Analysis and (4) Domain Schema Design. It also incorporates the requirements of sharability and the criteria of cohesion and reusability for designing the schema. In addition, this approach offers the modeling tools used in the phases that lead to the development of the schema. In this approach, the domain primitive schema consists of *primitive object classes* (or primitive classes) that represent the data shared by users throughout the life-cycle phases. Each primitive class is a module that is designed to have maximum cohesion and reusability. On the other hand, *composite object classes* (or composite classes) represent the complex user views of the underlying facility data. Using primitive classes, individual users can assemble any number of composite classes customized to their own needs.

## 3. Objectives/Benefits

The objectives of this research were *to better understand what is required to model complex facility data for the purpose of integration* and *to develop modeling tools that aid modelers*. The P-C Approach is the resulting methodology that uniquely combines the power of object-oriented data modeling with the notions of primitives and composites to enable data sharing and facilitate data exchange. Indeed, primitive classes represent the fundamental data representations that are shared by multiple users throughout the life-cycle phases in the domain. Composite classes can be assembled by individual users from primitive classes to represent complex user views. One advantage here over other modeling approaches is that composite classes need not be defined a priori and are not limited in number. Moreover, the primitive schema can be extended by adding primitive classes, and as a result, more composite classes can be defined from primitive classes, both old and new. This extension of the schema can accommodate evolving life-cycle phases. In addition, the schema is rich in content, representing the physical descriptions (or form), engineering functions and behavioral states of the design objects in the domain,

and the design process itself. Finally, the primitive schema provides a natural common ground for exchanging data (via primitive instances) between the different applications.

#### **4. Methodology**

The project was divided into three parts: (1) the development of the P-C Approach, (2) the application of the approach to the transmission tower domain, and (3) the testing of the approach in the tower domain. The first two parts were pursued concurrently and the third part (testing) was conducted last. The results are presented in the next section.

#### **5. Results**

The primary result of the research was a comprehensive and coherent methodology, the P-C Approach, that encompasses the steps, requirements, criteria and tools for modeling facility data to support integration. Computer-Aided Software Engineering (CASE) tools automating this approach can assist database designers with process modeling and object-oriented data modeling.

The in-depth application of this approach to a real-life tower domain resulted in a detailed set of functional schemata describing the tower engineering process, as well as a comprehensive primitive schema of the domain. This schema includes more than two hundreds primitive object classes that are organized into thirty different class hierarchies. These hierarchies depict form, function, and behavior of design objects in the domain. This schema can be used as a convenient starting point for modeling other facility engineering domains. In addition, a database for one specific tower was implemented using a commercial object-oriented database management system (ONTOS). The sample data from this database can be used in other projects at CIFE.

The P-C Approach was also tested in the tower domain. As a result, the scope of applicability, strengths and limitations of the approach were defined. More importantly, the three measurements of the primitive schema's performance (i.e., schema completeness, efficiency, and sharing) introduced here is an important step toward operationalizing the testing and acceptance of data standards or schemata developed for the purpose of integration.

#### **6. Research Status**

The Primitive-Composite Approach has been tested and carefully documented (in this report and in CIFE report #77, which described how to use PANDA for functional analysis). The four phases use formal methods for domain analysis and/or schema design, including rules and guidelines for users.

Two main areas are open for future development:

1. Computer-Aided Software Engineering (CASE) tools developed after the P-C Approach can help automate the modeling effort by assisting modelers in analyzing processes and data, in designing database schemata, and in building information systems to support collaborative facility engineering work.
2. Automated data integration tools can use the notion of primitives to drive data exchange between different applications. Algorithms need to be developed to perform mappings on data to translate it from the context of a composite schema and to the context of the primitive schema. The P-C Approach can provide a much simpler and more powerful basis for data exchange than data exchanges based on a complex global schema.

# ***Extended Abstract***

---

---

A facility engineering process typically involves participants from different disciplines, several life-cycle phases, and many computer applications. Large amounts of data must be communicated among these participants and applications during these evolving phases. However, communicating this data is often difficult. Users (i.e., participants or computer applications) have different needs for the facility data and thus use different data representations. Data in different representations cannot be communicated directly between these users. This research addresses the incompatibility or, put differently, the lack of “integration” of various data representations. In particular, data integration focuses on improving the compatibility of data and data representations within the same process. Scholars have proposed different approaches to data integration. Direct data translation tools have been used for this purpose, but they do not provide an economical long-term solution. Current international efforts such as the STandard for the Exchange of Product model data (STEP) involve developing a more economical solution to this problem. Their goal is to develop machine-readable data representations that can be shared within a domain and across domains. Data modeling is a major undertaking in those standards development efforts. The research described here shares that goal.

In this project, we studied the modeling of data in facility engineering for the purpose of supporting data integration. More specifically, *we developed a methodology, the Primitive-Composite (or P-C) Approach, with which a designer of a database or information system (or modeler for short) can analyze a given facility engineering domain and can design an integrated, object-oriented schema representing that domain.* In this approach, the domain primitive schema consists of “primitive object classes” that represent the data shared by users throughout the life-cycle phases. Using these primitive classes, individual users can assemble any number of “composite object classes” customized to their own needs. One advantage of this schema is that composite classes representing complex user views need not be defined a priori and are not limited in number. In addition, primitive classes can be added incrementally to the schema, and as a result, more composite classes can be defined from primitive classes, both old and new. This extension of the schema can accommodate evolving life-cycle phases. Moreover, the schema is rich in content, representing the physical descriptions (or form), engineering functions and behavioral states of the design objects in the domain, and the design process itself.

The result of this research, the P-C Approach, includes the following:

- Four phases for analyzing the given domain and designing a schema, namely (1) Preliminary Domain Study, (2) Functional Analysis, (3) Domain Entities Analysis and (4) Domain Schema Design. The first three analysis phases lead to the schema design phase. The resulting schema must enable multiple users to share data representations across life-cycle phases in the domain. Specifically, it must support multiple user views and must be extensible.
- The modeling tools used in these phases, which lead to the development of a schema meeting the requirements mentioned above. Specifically, these modeling tools directly incorporate the criteria of cohesion and reusability to design primitive

classes of the schema. These modeling tools also provide the elements (i.e., concepts, graphical representations, operations, rules, etc.) necessary for building Computer-Aided Software Engineering (CASE) tools with which a modeler can represent facility data using the P-C Approach.

The modeling tools are:

- *The PARTitioned eNginEering DATA flow model (or PANDA)* is an extension of the Data Flow model. A modeler uses PANDA in Phase 2 to analyze the facility engineering process in the domain in order to understand how data is used in that process. The resulting functional schema of the process provides the information that is used directly in the subsequent phase. PANDA provides the concepts needed to represent complicated facility engineering processes, as well as those needed to represent the participants, their participation, and complex flows of data, material, and products. In addition, PANDA provides graphical representations, syntactic and semantic rules, schema transformation operations, a customized method, and modeling guidelines. Moreover, PANDA has a partitioned architecture that helps the modeler organize thoughts about complicated engineering processes.
- *The Domain Entities AnaLysis method (or DEAL)* analyzes data-intensive objects such as beams, columns, tower panels, etc. that experts in the domain design. A beam, for instance, may be described by geometry, material, fabrication features, load-resisting functions, bending stresses, etc. Entities representing these design objects are called “domain entities.” A modeler uses DEAL in Phase 3 to identify the primitive entities that are building blocks of the domain entities. These primitive entities are used in Phase 4 to design the domain primitive schema. DEAL provides concepts, terms, assumptions, graphical representations, procedures, operations, and rules needed for decomposing domain entities into (conceptual) primitive entities using the criteria of cohesion and reusability. Cohesion is defined here as a measurement that indicates how closely the data items of an entity relate to one another. Specifically, DEAL considers five principal dimensions when evaluating the cohesion of the domain entities: (1) how the data is organized and thus how it can be accessed by humans in the work environment, (2) to which concepts the data relates, (3) at what time the data is created, (4) from which computational sources the data is derived, and (5) how the data is used in activities of the engineering process. Reusability is another measurement that indicates the extent to which an entity can be reused (i.e., used without modifications) in describing other domain entities. DEAL uses five levels of reusability: reusable (1) for more than one domain entity of a common type, (2) for a single domain, (3) for a single industry (i.e., for more than one domain), (4) for more than one industry of a common type, and (5) for more than one industry type.
- *The Primitive-Composite (or P-C) Data Model and Method* are used in Phase 4 to design the domain primitive schema. This schema includes primitive classes that come from primitive entities identified in the domain entity analysis. *Therefore, the design of this schema is optimized using the criteria of cohesion and reusability: Each primitive class is a module of attributes designed to have maximum cohesion and reusability.* The P-C Data Model provides the concepts of primitive and composite classes and instances, generalization, aggregation, association, and derivation relationship types. The accompanying method includes the steps, rules, and guidelines necessary to refine and transform the primitive entities into primitive classes based on the concepts of the model.

We applied the P-C Approach to the domain of electrical utility transmission towers. We first studied the information collected about the tower engineering process. Using PANDA, we then created a detailed set of graphical functional schemata describing that process [Phan 92]. As the final result, we developed a domain primitive schema of the tower domain. This schema shows that form, function, and behavior of the design objects in such a domain, and the design itself, can all be represented in a coherent fashion. First, form, function, behavior, and design are represented in separate primitive class hierarchies in the domain primitive schema. Indeed, geometric curves, topological elements, material properties, fabrication features, functions, requirements, strength behavior (i.e., response forces and stresses), serviceability behavior (i.e., displacements and strains), design description primitives, etc., are represented in these primitive class hierarchies. In particular, design description primitive classes represent important elements of a design such as design artifacts, features, parameters, constraints, versions, and alternatives. Using this schema, a tower can be decomposed hierarchically in two ways: by the functions that it provides and by the design artifacts created by the facility engineers. The schema includes primitive classes needed to support both types of decomposition. We also tested the P-C Approach in the tower domain and built a database for a selected tower (using an object-oriented database management system).

The immediate contribution of this research is a methodology, the P-C Approach, for modeling facility data to support data integration. In the long run, this research will have the following impacts on data modeling and data exchange: First, the P-C Approach will aid the modeling of large complex facility engineering domains. The schema for the tower domain can be used as a starting point for the development of schemata for those domains. Second, CASE tools automating the approach will assist modelers in many aspects of modeling such as analyzing processes and data and designing object-oriented database schemata. These CASE tools will also make communication between modelers and domain experts more effective. Third, the P-C Approach suggests a new data exchange paradigm, under which application developers will no longer have to build special-purpose translators. Each application essentially carries the descriptive knowledge needed to support data exchange with any other application. By the same token, application developers will no longer need to build complex product models to support data exchange. Applications will share only the common primitive classes and will exchange only data that they really need.

Future research is needed to enhance the P-C Approach through application in other engineering domains, to study distributed data management issues in collaborative facility engineering work environment, to implement CASE tools automating the approach and supporting data exchange using the approach, and to further develop measures for the validation of data standards or schemata supporting data integration.





# ***Acknowledgments***

---

---

This technical report is based on the doctoral thesis of Dr. Dung Huu Douglas Phan. The research project was supported by the National Science Foundation through grant MSM-8958315 ("Integrating Databases and Knowledge Bases for Life-Cycle Facilities Engineering") and the Stanford Center for Integrated Facility Engineering (CIFE) through grants DB9201 ("Conceptual Development and Database Applications of the Primitive Composite Data Model in Structural Engineering") and FP8902 ("Linking Design Data with Knowledge-Based Construction Systems").

We gratefully acknowledge the contributions of many individuals from both the academic community and industries, without whom this research would not be possible:

- Dr. Martin Fischer, Dr. Kincho Law and Dr. Gio Wiederhold (dissertation committee members who guided the research);
- Dr. Gamaleldin Abubakr Abdalla, Tanya Do, Dr. Thomas Froese, Dr. Renate Fruchter, Dr. Thomas Gruber, Asish Gupta, Cynthia Howard, Glenn Katz, Dr. Arthur Keller, Dr. Raymond Levitt, Jared Nedzel, Dr. William Rasdorf, Dr. Paul Teicholz, Sanjay Tiwari and Julie Wald (research mentors and collaborators who provided intellectual inputs to the work); and
- Mike Cheung and William Mills (domain experts who offered their valuable time and expertise to the research).



# Contents

---

---

<b>SUMMARY</b> .....	iii
<b>EXTENDED ABSTRACT</b> .....	v
<b>ACKNOWLEDGMENTS</b> .....	viii
<b>PART I: PROBLEM DEFINITION AND BACKGROUND</b> .....	<b>1</b>
<b>Chapter 1—Introduction</b> .....	<b>1</b>
1.1 Research Problem .....	2
1.1.1 The Facility Data Integration Challenge .....	2
1.1.2 Facility Data Modeling Difficulties .....	6
1.2 Research Idea Underlying the Proposed Solution.....	8
1.3 Research Objectives .....	10
1.4 Reader’s Guide .....	10
<b>Chapter 2—Field of Study</b> .....	<b>13</b>
2.1 Background on Data Modeling .....	14
2.1.1 Terminology: From "datum" to "data modeling" .....	14
2.1.2 Historical Development of Data Models .....	14
2.1.3 Object-Oriented Data Modeling Paradigm: Models and Concepts .....	17
2.1.3.1 Object-Oriented Data Models.....	17
2.1.3.2 Key Object-Oriented Concepts.....	17
2.1.3.3 Why Object-Oriented Data Modeling in this Research? .....	18
2.2 Background on Data Modeling for Facility Engineering.....	19
2.2.1 Existing Work in Facility Data Modeling.....	19
2.2.1.1 Hierarchical Models with Distinct Levels of Aggregation .....	19
2.2.1.2 Models with Multiple Linked Aggregation Hierarchies .....	20
2.2.1.3 Models for A/E/C Product Data Exchange.....	20
2.2.2 Difficulties in Facility Data Modeling .....	22
2.3 Background on Data Integration .....	26
2.4 Chapter Summary .....	27

<b>PART II: THE PROPOSED SOLUTION .....</b>	<b>29</b>
<b>Chapter 3—The Primitive-Composite Approach .....</b>	<b>29</b>
3.1 A Perspective on the Development of the P-C Approach.....	30
3.2 Overview of the P-C Approach.....	32
3.2.1 Requirements and Criteria .....	33
3.2.2 Phases and Modeling Tools .....	34
3.3 Evaluation of the P-C Approach .....	37
3.3.1 Scope of Applicability of the Approach .....	37
3.3.2 Strengths of the Approach.....	39
3.3.3 Limitations of the Approach .....	40
3.4 Chapter Summary .....	41
<b>Chapter 4—Functional Analysis Using PANDA.....</b>	<b>43</b>
4.1 Introduction to Functional Analysis in the P-C Approach.....	44
4.2 Development Perspective.....	44
4.2.1 Required Capabilities and Properties.....	44
4.2.2 Background Study.....	46
4.2.3 Evaluation of Selected Models .....	47
4.3 The Extended Model: PANDA .....	47
4.3.1 Overview of PANDA.....	47
4.3.2 Key Concepts and Graphical Representations .....	49
4.3.3 Syntactic and Semantic Rules .....	55
4.3.4 Schema Transformation Operations.....	57
4.4 Using PANDA .....	59
4.4.1 Method for Using PANDA .....	59
4.4.2 Additional Guidelines for Using PANDA .....	61
4.4.3 Illustrative Example in Transmission Tower Engineering .....	63
4.5 Chapter Summary .....	69
<b>Chapter 5—Domain Entities Analysis Using DEAL .....</b>	<b>71</b>
5.1 “What Should An Entity Be?”: The Domain Entities Analysis in the P-C Approach.....	72
5.2 An Overview of the Domain Entities AnaLysis method .....	72
5.3 A Formal View of DEAL.....	75
5.3.1 Basic Concepts and Terms.....	75
5.3.1.1 Entity, Instance, and Data Item.....	75

5.3.1.2	Conceptual Categories .....	75
5.3.1.3	Domains and Their Classifications .....	76
5.3.1.4	Cohesion .....	76
5.3.1.5	Reusability .....	79
5.3.1.6	Domain Entity and Primitive Entity .....	79
5.3.2	Assumptions .....	80
5.3.3	Graphical Representation: Domain Entity Decomposition Tree .....	81
5.3.4	Procedures and Operations in DEAL-1 and DEAL-2 Versions .....	82
5.3.4.1	The Basic Version, DEAL-1 .....	82
5.3.4.2	The Improved Version, DEAL-2 .....	86
5.3.5	Rules .....	88
5.4	Example of Using DEAL-1 .....	88
5.5	DEAL as a Medium for Mediating Data Representations .....	96
5.6	Chapter Summary .....	98

**Chapter 6—Domain Schema Design Using the P-C Data Model and Method ... 99**

6.1	Introduction .....	100
6.2	The P-C Data Model .....	100
6.2.1	Building Blocks of the Model .....	100
6.2.2	Direct Extensions to Object-Oriented Concepts .....	101
6.2.3	Relationships and Relationship Types .....	105
6.2.3.1	Definition of Relationships .....	105
6.2.3.2	Types of Relationships .....	106
6.2.3.3	Semantics of Relationships .....	107
6.2.3.4	Permissible Relationship Types Among Primitive and Composite Classes and Instances .....	108
6.3	The P-C Data Modeling Method .....	108
6.3.1	Overview of the Method .....	108
6.3.1.1	Steps for the Design of a Domain Primitive Schema .....	108
6.3.1.2	Rules and Guidelines of the Method .....	110
6.3.2	Entity Dependencies and Dependency Types .....	112
6.3.3	Designing A Domain Primitive Schema .....	113
6.3.4	Designing A Composite Schema .....	118
6.4	Chapter Summary .....	119

**Chapter 7—Form, Function, and Behavior Representations in the P-C Approach 121**

7.1 Overview .....	122
7.2 Form, Function, and Behavior Representations .....	122
7.2.1 Form Representation .....	122
7.2.2 Function and Behavior in addition to Form .....	124
7.2.3 Function Representation .....	126
7.2.3.1 Description of Functions.....	126
7.2.3.2 Description of Requirements .....	128
7.2.3.3 Description of Design .....	131
7.2.4 Behavior Representation .....	134
7.3 Hierarchical Facility Decomposition by Functions or by Artifacts .....	137
7.3.1 Decomposition by Functions .....	137
7.3.2 Decomposition by Design Artifacts .....	140
7.3.3 Mappings Between Functions and Artifacts .....	141
7.4 Chapter Summary .....	143

**PART III: TESTING, CONTRIBUTIONS, AND CONCLUSIONS 145**

**Chapter 8—Testing of the P-C Approach ..... 145**

8.1 Introduction.....	146
8.2 Testing of the P-C Approach .....	146
8.2.1 Research Question.....	146
8.2.2 Research Hypothesis .....	146
8.2.3 Independent and Intermediate Variables and Their Measurements.....	147
8.2.4 Key Dependent Variable and Its Measurements .....	148
8.2.5 Test Cases .....	151
8.2.5.1 Description of the Main Test Problem and Cases .....	151
8.2.5.2 Selection of Data Uses.....	152
8.2.6 Test Procedure.....	153
8.2.7 Test Results .....	154
8.3 Chapter Summary .....	159

**Chapter 9—Contributions, Conclusions, and Future Research..... 161**

9.1 Contributions.....	162
9.1.1 Definition of the Modeling Requirements and Criteria .....	162
9.1.2 Development of the P-C Approach .....	162
9.1.3 Development of the Modeling Tools .....	163

9.1.4 Development of the Test Domain Schema .....	163
9.2 Conclusions .....	164
9.3 Impacts of the Research on Data Modeling and Data Exchange .....	166
9.4 Directions for Future Research .....	167
9.5 Final Remarks .....	169
<b>REFERENCES .....</b>	<b>171</b>





# ***List of Tables***

---

---

TABLE 4.1: Concepts and Graphical Representations in Partition I. ....	52
TABLE 4.2: Concepts and Graphical Representations in Partition II. ....	53
TABLE 4.3: Concepts and Graphical Representations in Partition III. ....	54
TABLE 4.4: Matrix of Permissible Node Linkages in PANDA Using the Appropriate Link Types. ....	56
TABLE 8.1: The Selected Data Uses Selected in the Six Test Cases. ....	152
TABLE 8.2: Measurements for P-1 and P-2 of the Tower Primitive Schema. ....	154
TABLE 8.3: Measurements for P-3 of the Tower Primitive Schema. ....	155
TABLE 8.4: Customization of Composite Classes for Data Uses Considered in the Test. .....	156



# List of Figures

---

---

FIGURE 1.1: Six Phases of the Tower Engineering Process. ....	3
FIGURE 1.2: A Sample Electrical Utility Transmission Tower. ....	4
FIGURE 1.3: Customizing Composite Classes from a Primitive Schema. ....	9
FIGURE 2.1: Historical Development of Data Models. ....	15
FIGURE 2.2: Aggregation Levels in the Structural Steel Framing Data Model [Lavakare 90]. ....	23
FIGURE 2.3: A Sample Non-Homogeneous Class Hierarchy. ....	24
FIGURE 2.4: A Sample Large Object Cluster. ....	25
FIGURE 3.1: Research Methodology and Results. ....	31
FIGURE 3.2: Overview of the Primitive-Composite Approach. ....	33
FIGURE 3.3: Phases, Modeling Tools, and Outputs of the P-C Approach. ....	35
FIGURE 4.1: A Sample Partitioned Data Flow Diagram Using PANDA. ....	48
FIGURE 4.2: Legend for the Partitioned Data Flow Diagrams that follow. ....	64
FIGURE 4.3: Diagram Notes for the Partitioned Data Flow Diagrams that follow. ....	65
FIGURE 4.4: Intermediate Skeleton of Phase IV, Tower Construction Planning. ....	66
FIGURE 4.5: Function IV.S1, Dimensioning Members and Laying out Connections, and Function IV.S2, Detailing Fabrication Parts, of Phase IV (Tower Construction Planning). ....	67
FIGURE 4.6: Function IV.S3, Generating Detailed Drawing, and Function IV.S4, Compiling Erection Bill of Material and Bundling List, of Phase IV (Tower Construction Planning). ....	68
FIGURE 5.1: Analysis of A Given Domain Entity Using DEAL. ....	74
FIGURE 5.2: Vertex Intension and Extension. ....	82
FIGURE 5.3: Step 1 of DEAL-1 Considering Access-Cohesion. ....	89
FIGURE 5.4: Repeating Step 1 of DEAL-1 Using Direct and Specific Logical Access Paths. ....	90
FIGURE 5.5: Conceptual Categories Assigned to the Leaves of the Tree during the ASSIGN Operation in Step 2. ....	91
FIGURE 5.6: Decomposition Tree at the End of Step 2 Considering Concept-Cohesion. ....	92
FIGURE 5.7: Logical Times Assigned to the Leaves of the Tree during the ASSIGN Operation in Step 3. ....	93
FIGURE 5.8: Time-Ordered Decomposition Tree at the End of Step 3 Considering Time-Cohesion. ....	94
FIGURE 5.9: Vertices Decomposed at the End of Step 4 Considering Source-Cohesion. ....	95
FIGURE 5.10: Decomposition Tree at the End of Step 5 Considering Use-Cohesion. ....	97

FIGURE 6.1: Building Blocks of the Primitive-Composite Data Model. ....	101
FIGURE 6.2: A Sample Primitive Characterization Hierarchy. ....	102
FIGURE 6.3: Customization of a Composite Class. ....	104
FIGURE 6.4: Overview of the Design of a Domain Primitive Schema. ....	109
FIGURE 6.5: Summary of Rules and Guidelines of the P-C Data Modeling Method. .	111
FIGURE 7.1: Legend for the Graphical Representations of Sample Primitive Characterization Hierarchies. ....	123
FIGURE 7.2: Sample Primitive Classes Representing Material Properties. ....	125
FIGURE 7.3: What Should Be Communicated About A Facility Design? ....	126
FIGURE 7.4: Sample Primitive Classes Representing Structural Engineering Function Descriptions. ....	127
FIGURE 7.5: Sample Primitive Classes Representing Specifications of Load Conditions and Load Sources. ....	128
FIGURE 7.6: Sample Primitive Classes Representing Specifications of Load Cases and Load Paths. ....	129
FIGURE 7.7: Sample Primitive Classes Representing External Load Specifications. ..	130
FIGURE 7.8: Sample Primitive Classes Representing External Load Application Specifications. ....	130
FIGURE 7.9: Sample Primitive Classes Representing Requirement Specifications.....	131
FIGURE 7.10: Three Principal Dimensions of a Facility Design Supported by the P-C Approach. ....	132
FIGURE 7.11: Sample Design Description Primitive Classes. ....	134
FIGURE 7.12: Sample Primitive Classes Representing Descriptions of Structural Analysis Elements and Nodes. ....	135
FIGURE 7.13: Sample Strength Behavior Primitive Classes. ....	136
FIGURE 7.14: Sample Serviceability Behavior Primitive Classes. ....	136
FIGURE 7.15: A Sample Transmission Tower' s Function Hierarchy. ....	138
FIGURE 7.16: A Sample Transmission Tower' s Functions and Requirements. ....	139
FIGURE 7.17: A Sample Transmission Tower' s Artifact Hierarchy. ....	140
FIGURE 7.18: Sample Function-Artifact Mapping Primitive Classes. ....	141
FIGURE 7.19: Dual Artifact and Function Hierarchies of A Transmission Tower. ....	142
FIGURE 8.1: All Test Variables. ....	147
FIGURE 8.2: An Illustration of Generation of User-Specific Schemata from a Common Domain Schema. ....	149
FIGURE 8.3: Three Components of the Schema Performance Variable. ....	150
FIGURE 8.4: The Main Test Problem. ....	151

# ***PART I: PROBLEM DEFINITION AND BACKGROUND***

## ***Chapter 1***

---

---

### ***Introduction***

#### ***Chapter Abstract:***

The research described in this dissertation involves the development of a methodology, called the Primitive-Composite (P-C) Approach, for modeling data in facility engineering to support data integration. In introducing this research, this chapter first presents the research problem, which involves both facility data integration and modeling. It then explains the idea underlying the P-C Approach. It also states the objectives in the development of this idea into a working solution. It ends with a reader's guide to the remainder of this dissertation.

#### ***Organization:***

##### *1.1 Research Problem*

###### *1.1.1 The Facility Data Integration Challenge*

###### *1.1.2 Facility Data Modeling Difficulties*

##### *1.2 Research Idea Underlying the Proposed Solution*

##### *1.3 Research Objectives*

##### *1.4 Reader's Guide*

The design and construction of a facility—whether it is a building, bridge, transmission tower or space station—involves a complex engineering process. This process requires close coordination among owners, architects, engineers, and contractors in all phases of the project. The goals are to maintain efficiency and minimize costs while ensuring high engineering quality. Effective communication of facility data among the principal players throughout the phases is vital to the achievement of these goals. In recent years, integration has been recognized as one viable project management strategy for establishing such communication [Howard 89a]. Integration means coordinating all phases and aspects of a project through the cooperative use of information from database and knowledge-base systems [Tatum 90]. However, even with increasingly affordable hardware and powerful software, designing and implementing information systems capable of supporting integration still poses an important and challenging research area. Data modeling plays a central role in developing these information systems.

This dissertation reports on research in modeling facility data for the purpose of supporting integration. *The goal of the research described here is to better understand how to model facility data to support integration and to develop modeling tools that aid modelers in handling this task. This research verifies the hypothesis that a methodology for modeling data in a given facility engineering domain can produce a schema that can be shared among multiple users across the life-cycle phases in that domain.* In developing such a methodology, this research synthesizes ideas from fields such as data modeling, knowledge representation in artificial intelligence, software engineering, and information systems design. The research also closely relates to emerging object-oriented database technology. Specifically, this research involves the development of a methodology called the “Primitive-Composite” (or P-C) Approach. This approach incorporates the steps needed to analyze a given domain and develop an object-oriented database schema of the domain, the requirements and criteria for designing the schema, and the modeling tools used in the steps that lead to the development of the schema meeting the requirements. The P-C Approach was also applied to the domain of electrical utility transmission towers. In the course of testing the approach in the tower domain, the scope of applicability, strengths and limitations of the approach were defined. In addition, a proof-of-concept software demonstration of the research concept was built.

This chapter first explains the research problem that involves both facility data integration and modeling. It then describes the goal and objectives of the research. Finally, it provides a reader’s guide to the remainder of this dissertation.

## **1.1 Research Problem**

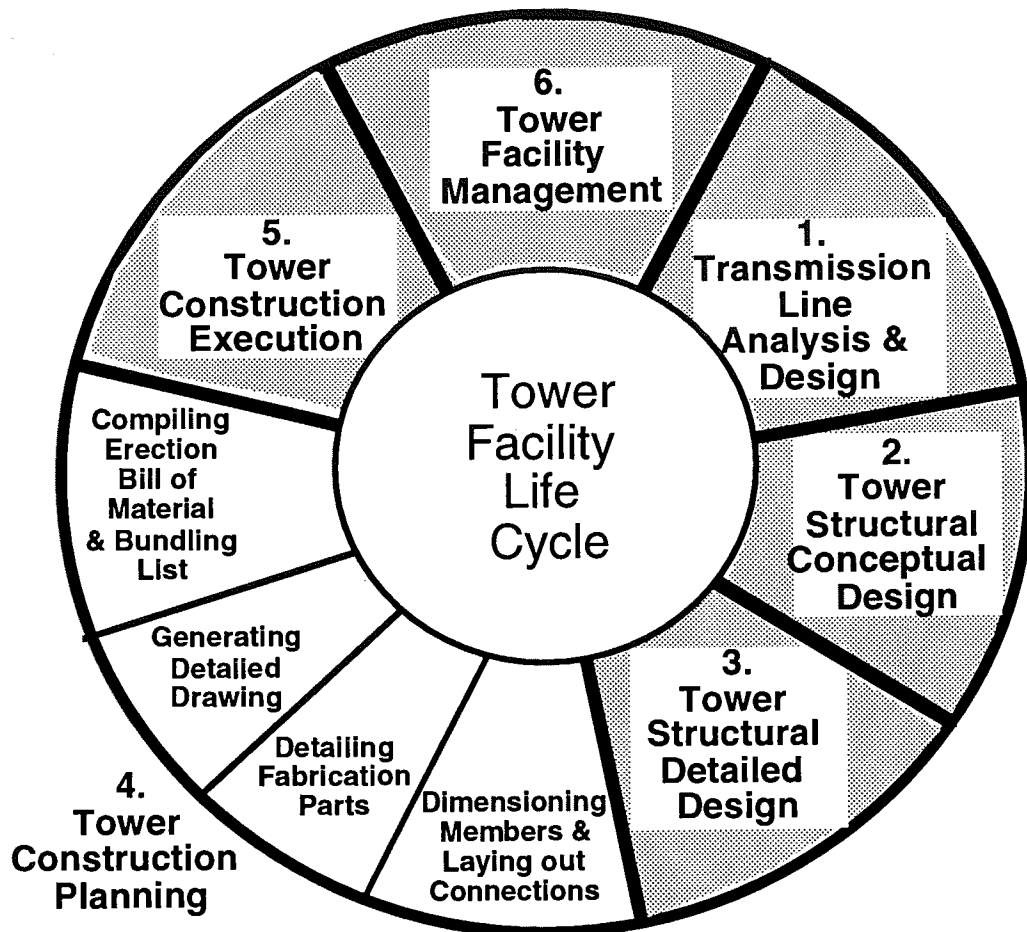
---

In this research, I studied data integration and data modeling in facility engineering. The following sections describe the challenges and difficulties concerning these areas.

### **1.1.1 The Facility Data Integration Challenge**

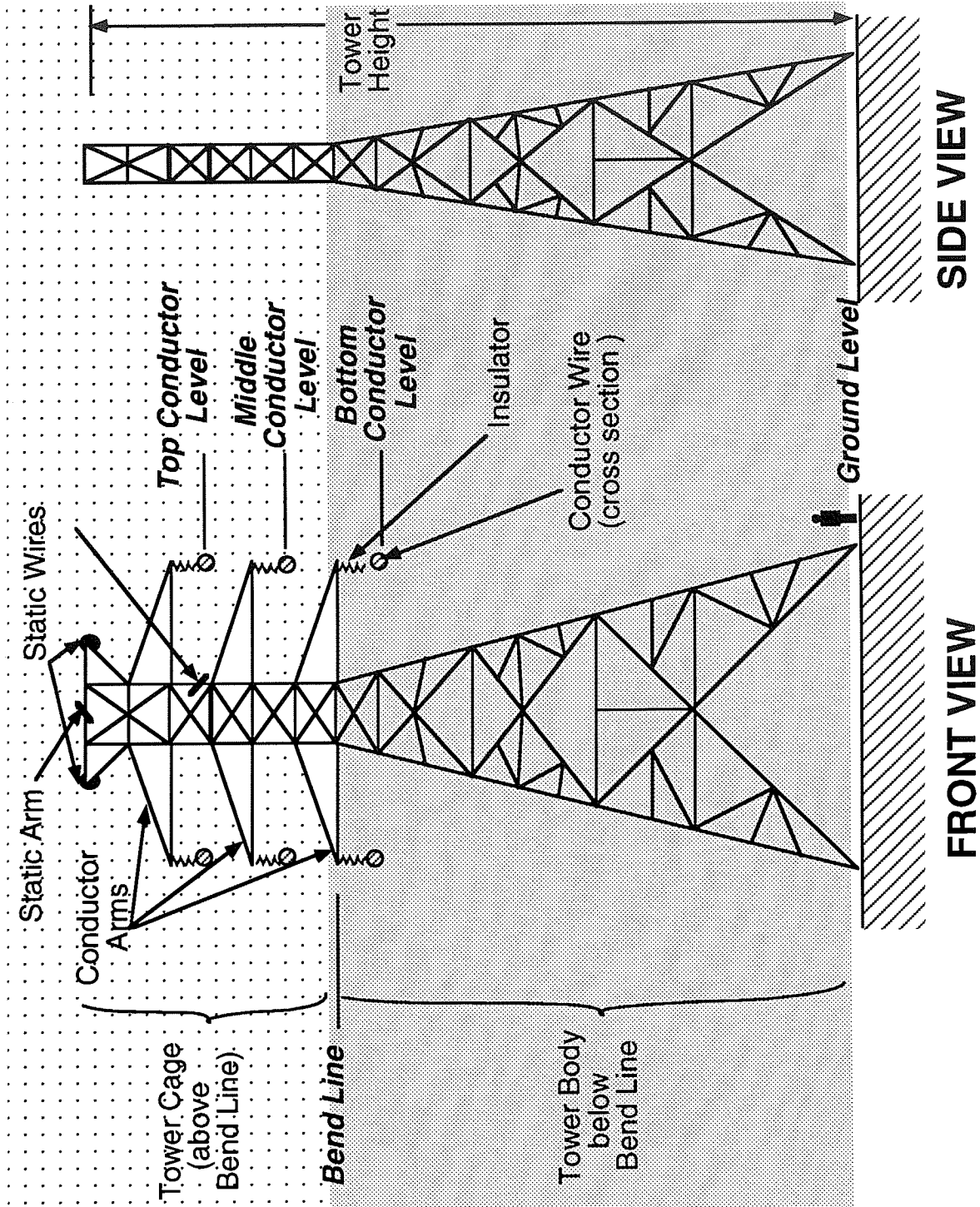
In a typical facility engineering process, large amounts of data must be communicated among participants and computer applications during evolving life-cycle phases. However, communicating this data is often difficult: users (i.e., participants or applications) have different needs for the facility data and thus use different data representations. Data integration aims at improving the compatibility and reusability of data and data representations within the same process. The following paragraphs describe the need for data integration in facility engineering and then briefly review existing data integration approaches.

**An Engineering Perspective: Need for Communicating Data** Facility engineering involves the design and construction of a facility such as a building, bridge, transmission tower, space station, etc. In this research, I studied the domain of electrical utility transmission towers. A facility engineering process typically involves a myriad of skilled individuals from different disciplines, who may be in one organization, in multiple industries, or even at various geographic locations. Designing and constructing transmission towers for instance, involves electrical engineers, structural engineers, foundation engineers, structure detailers, fabricators, construction contractors and crews. As a result, the process is highly fragmented. “Horizontal fragmentation” refers to division of the process according to the specialization of participants involved; “vertical fragmentation” refers to division of the process into phases and into smaller functional units [Howard 89a]. Indeed, the engineering process extends throughout a very long life-cycle and includes several phases of development. Figure 1.1 shows the six phases of the tower engineering process: (1) Transmission Line Analysis and Design, (2) Tower Structural Conceptual Design, (3) Tower Structural Detailed Design, (4) Tower Construction Planning, (5) Tower Construction Execution and (6) Tower Facility Management. These phases are further divided into several functions (or subprocesses), each of which consist of many activities. Figure 1.2 illustrates a sample transmission tower.



**FIGURE 1.1: Six Phases of the Tower Engineering Process.** The heavy lines mark the beginning and end of each phase. The four functions (or subprocesses) of the Tower Construction Planning phase are also shown to exemplify the next level of decomposition of a phase.





**FIGURE 1.2: A Sample Electrical Utility Transmission Tower.**

Large amounts of data are generated during a typical facility engineering process. Different participants and applications use this data to carry out a variety of activities. Data generated by one activity is used in many downstream activities. Therefore, this data must be communicated among several different “users” (i.e., participants and applications) during the various phases. Failure to communicate this data can cause delays, errors, cost overruns, accidents, structural catastrophes, etc. In short, effective communication of facility data in any given project is vital to maintaining work productivity, minimizing costs, and ensuring high engineering quality.

***The Challenge: Difficulty in Communicating Data and Need for Integrating Data*** The inherent fragmentation, both vertical and horizontal, of the facility engineering process makes the communication of data between participants and applications difficult. First, participants have different views of the underlying facility data. (A view corresponds to a subset of data that is defined to serve a particular user need.) Second, participants use computer applications that assist them in various engineering tasks. Examples of these computer applications are architectural layout, design loads computation, geometry configuration, structural analysis, member design, computer-aided drafting, construction project scheduling, etc. These applications are usually stand-alone programs that are not designed to exchange data. These applications often run on different computers and use varying representation paradigms, programming languages, development tools and environments, etc. Moreover, data used in different applications may be syntactically and semantically incompatible. To exchange data among applications, the engineer normally either enters data for each application manually or builds computer programs that translate data between pairs of applications. These alternatives are neither expedient nor efficient. Third, as the life-cycle phases evolve, new property values are defined for the facility design objects, or these design objects are decomposed into physical or functional components [Eastman 78]. The engineer usually represents these design objects in an ad hoc manner in different phases. This results in duplicate efforts and incompatible data representations. Consequently, data in different representations cannot be communicated directly between users. This project addresses these incompatibilities, which can also be seen as lack of “integration” of various data representations. Data integration seeks solutions to improve the compatibility of data and data representations used in the same process.

***Data Integration Approaches*** The need for data integration in engineering domains has motivated research and development in different areas. Scholars have proposed different approaches to data integration, two of which are of interest to this research. These two approaches are explained in [Abdalla 89]. The first of these, the “direct translator approach,” involves writing a translator that directly translates the output of application A to the input format required by application B. This approach is not economical because of the large number of translators required to support data exchange among applications (i.e.,  $N * (N - 1)$  translators for  $N$  applications). The second approach, the “standard exchange format approach,” aims at providing a more economical solution to this problem. It involves developing machine-readable data representations that are used as standards for data exchange among all applications. These data representations can then be shared within a domain and across domains. This approach is efficient because it eliminates the need to write a separate translator for each pair of distinct applications. However, this approach requires time, effort, and suitable methods for developing the standard for a given discipline, as well as the communal agreement and commitment to use such a standard. The research described here has a similar motivation of sharing data representations, but focuses on the modeling of facility data to produce sharable data representations. Chapter 2 discusses other data integration approaches.

### 1.1.2 Facility Data Modeling Difficulties

Data modeling plays a significant role in data integration approaches such as the standard exchange format approach described above. In fact, data modeling is a major undertaking in many national and international efforts to develop product data exchange standards. This research focuses on modeling of facility data to support data integration. Data modeling involves observing the real world, abstracting the objects of interest and their properties, and building data representations that can be processed by computers. Data modeling is often difficult: Real-world objects are complicated, and modeling them requires suitable methods and tools. The following paragraphs emphasize three important difficulties that must be overcome in modeling facility data.

***Inherent Complexity of Facility Data*** Modeling facility data is difficult because of the inherent complexity of the data. Many past research studies focused on modeling data in engineering domains in general [Batory 76], [Lorie 83], [Johnson 83], [Wiederhold 85], [Ketabchi 86], [Kersten 86], [Wiederhold 88], [Barsalou 90], etc., and in facility engineering domains [Law 86], [Eastman 87], [Gielingh 88], [Gerardi 88], [Abdalla 89], [Bjork 89a], [Lavakare 89], [Law 89], [Rasdorf 90], [Abudayyeh 91], [Eastman 91], [Luiten 91b], [Eastman 92], [Froese 92], [Garrett 92], [Law 92], [Sause 92], etc. Like data in other engineering domains, facility data is more complex than data in traditional business applications (e.g., payroll, accounting, inventory control). Indeed, facility data describes complicated design objects with many levels of detail. This data can have complex types rather than simple types such as integers, real numbers, and strings. It can involve nested data structures and intricate object relationships. Representing this data usually requires many types of data structures and results in very large schemata. Finally, facility data describes the physical properties (or “form”), engineering functions, and behavioral states of the design objects. Form, function and behavior must be represented as distinct, yet tightly coupled, elements of a schema to describe facility design objects completely and to support reasoning about these objects. Representation of form, function, and behavior is still an important topic of research.

***Issues in Representing Facility Data*** Inappropriate techniques of modeling facility data can lead to poor representations. Such representations can be ill-suited to the project or difficult to maintain, reuse and share among different applications. The major issues that must be addressed in representing facility data include:

- ***Limited Ability to Support Multiple User Views:*** Existing facility data representations such as the Structural Steel Framing Data Model [Lavakare 89], the Ratas Building Product Model [Bjork 88], and the Component-Connection Abstraction Model [Powell 88] tend to predefine a number of descriptions of the facility and lack the ability to allow users to customize their own views.
- ***Lack of Support for Schema Evolution:*** Once defined, current facility data representations such as those listed in the first item cannot be extended once defined, to accommodate evolving facility life-cycle phases.
- ***Lack of Form, Function, and Behavior Representation:*** Existing data representations (e.g., those listed in the first item) include mainly form descriptions and do not distinguish form, function, and behavior.
- ***Common Facility Data Representation Traps:*** A study of existing data models and application schemata proposed for facility engineering [Phan 91b] revealed a

number of common representation traps. Chapter 2 explains these representation traps in detail. The three most critical are:

- *Misuse of Aggregation*: counterproductive misuse of the ubiquitous “part-of” relationship by imposing hierarchical levels of aggregation such as building, systems, subsystems, components, analysis elements, parts, and connections in describing a facility. These levels are rigid and may not apply to a given facility.
- *Non-homogeneous Class Hierarchies*: using many different criteria in defining subclasses at different levels of the class hierarchy. This is an undesirable feature for two reasons. First, different views and semantics are mixed in an indiscriminate manner in the class hierarchy. This hierarchy lacks the clean separation between the different description aspects (i.e., form, function, and behavior) of the facility design objects. Second, the modeler must anticipate all possible combinations of subclasses. This can lead to what is called the “cross product phenomenon.”
- *“Large Object Clusters”*: creating highly complex object class definitions that include multiple aspects of description in order to satisfy all user needs. The resulting object classes are inefficient and difficult to extend or reuse. Creating and managing instances from these classes is also a problem.

**The Task of Modeling Facility Engineering Domains** From a broader perspective, modeling real-life facility engineering domains is a technically challenging and time-consuming task that deserves special attention in research and development. In the beginning, a modeler<sup>1</sup> is overwhelmed by the size and complexity of the domain. In fact, the first challenge is to organize and analyze the plethora of information collected about the facility engineering. The modeler may not understand how the engineering process works and how data is used to support that process. However, she must have that understanding to model the data. The next critical challenge is the conceptual modeling of the domain. The Entity-Relationship (or E-R) model [Chen 76] is commonly used for conceptual modeling. This is explained in detail in [Batini 92]. In the E-R model, entities represent distinctive things, abstract or concrete, in the domain of interest. However, as we learned in this research project, identifying and defining the proper entities for a large complex engineering domain is not a trivial task, and the E-R model does not provide any criteria for doing this. This task becomes even more difficult when the goal is to build data representations to support data integration. Moreover, experts in the domain typically design data-intensive objects such as beams, columns, floors, and tower panels. A beam, for instance, may be described by geometry, material, fabrication features, load-resisting functions, bending stresses, etc. The representations of these design objects vary according to the engineering tasks or applications being considered. Schemata of existing applications can provide a starting point for the conceptual modeling, but they present only views that are specific to those applications. Moreover, these schemata often have the representational limitations described in the previous paragraph. In short, methods for modeling large complex engineering domains need to be developed, as Wimmer and Wimmer [92] point out. Currently, there is no suitable method for doing this. Consequently, this deficiency has limited the formal modeling of facility data to trivial academic exercises or, at most, to small and simple real-life domains.

---

<sup>1</sup> A person or team who is designing a database or information system.

## ***1.2 Research Idea Underlying the Proposed Solution***

---

This research proposes the P-C Approach as a solution to the problem described previously. This approach evolved from the intuitive idea of using the notions of primitives and composites with object-oriented concepts<sup>2</sup> (e.g., class, instance, attribute, method). The approach was named after this idea. A “primitive class” (or primitive object class) represents one simple concept such as curves, material properties, fabrication features, functions, or behavior stresses. It serves as an atomic description about form, function, or behavior in a schema of the domain. On the other hand, a “composite class” (or composite object class) describes a complex concept in terms of many atomic concepts in the domain for a particular user. For example, a beam may be described using combinations of specific forms, functions, and behaviors that provide different users with the views to which they have become accustomed [Howard 92].

To further explain this research idea, Howard et al. [92] has drawn the following analogy to human language. Since many different users contribute to and draw from the same overlapping set of data objects, the natural tendency in modeling facility data is to add more and more description to these objects to satisfy all users. As a result, extremely complex object definitions are created, and no one specialist completely understands them all. The schema that consists of these object definitions is like a “phrase book” that contains a limited number of predigested ideas. As long as the phrase that the user needs is in the book, she can use it and communicate with others. However, the user is out of luck if she needs to communicate about something not in the book or even a subtle variation on a defined phrase. By contrast, with the vocabulary of a language (along with the syntax), any phrase or sentence can be constructed, not just the limited set in the phrase book. The P-C Approach can be compared to a methodology for creating a data “vocabulary” for a domain—formally, a “domain primitive schema” (or primitive schema for short). From words—primitive classes—and syntax—rules belonging to the P-C Approach, users can customize the complex phrases—composite classes—that they need and still communicate data by sharing the same vocabulary. Thus, such a word-based vocabulary is far more powerful than a book containing a limited set of phrases. The power of the vocabulary is that complex phrases need not be defined a priori and thus need not be limited in number.

Similarly, in the P-C Approach, the domain primitive schema consists of primitive classes that represent the data shared by users throughout the life-cycle phases. Primitive classes are organized into several separate “primitive characterization hierarchies,” each of which involves one simple concept about form, function, or behavior. Examples of these hierarchies are geometric curves, basic topological elements, shape cross-section properties, material properties, fabrication features, functions, requirements, strength behavior, etc. By selecting the appropriate primitive classes from these hierarchies, individual users have the flexibility to assemble any number of “composite object classes” customized to their own needs. Figure 1.3 illustrates the customization of four different user views from the underlying primitive schema. A composite class is a subclass, aggregation, or association of two or more primitive classes. In addition, primitive classes can be added incrementally to the schema, and as a result, more composite classes can be defined from primitive classes, both old and new. This extension of the schema can accommodate evolving life-cycle phases. By the same token, a computer application can be described by a “composite schema” that includes

---

<sup>2</sup> Chapter 2 explains the fundamental object-oriented concepts and the reasons for adopting the object-oriented paradigm in this research.

composite classes customized for it. Applications do not share composite classes. They need only share primitive classes from which their composite classes are formed. A comparison of the composite schemata of two applications immediately reveals the data to be exchanged, since it is that belonging to the primitive classes referenced by both composite schemata.

Nevertheless, when applying this idea to a facility engineering domain, we faced three fundamental questions: "How is data actually used in a real-life facility engineering domain?," "How can primitive entities be identified in a given domain?," and then "How can these primitive entities be designed as object classes, from which composite classes can be defined?" These issues must be resolved to produce a working solution. As a result, we developed the P-C Approach as a structured methodology to answer these issues. This approach incorporates the steps, requirements, design criteria, and modeling tools that can be used to analyze a given facility engineering domain and to design a primitive schema representing that domain.

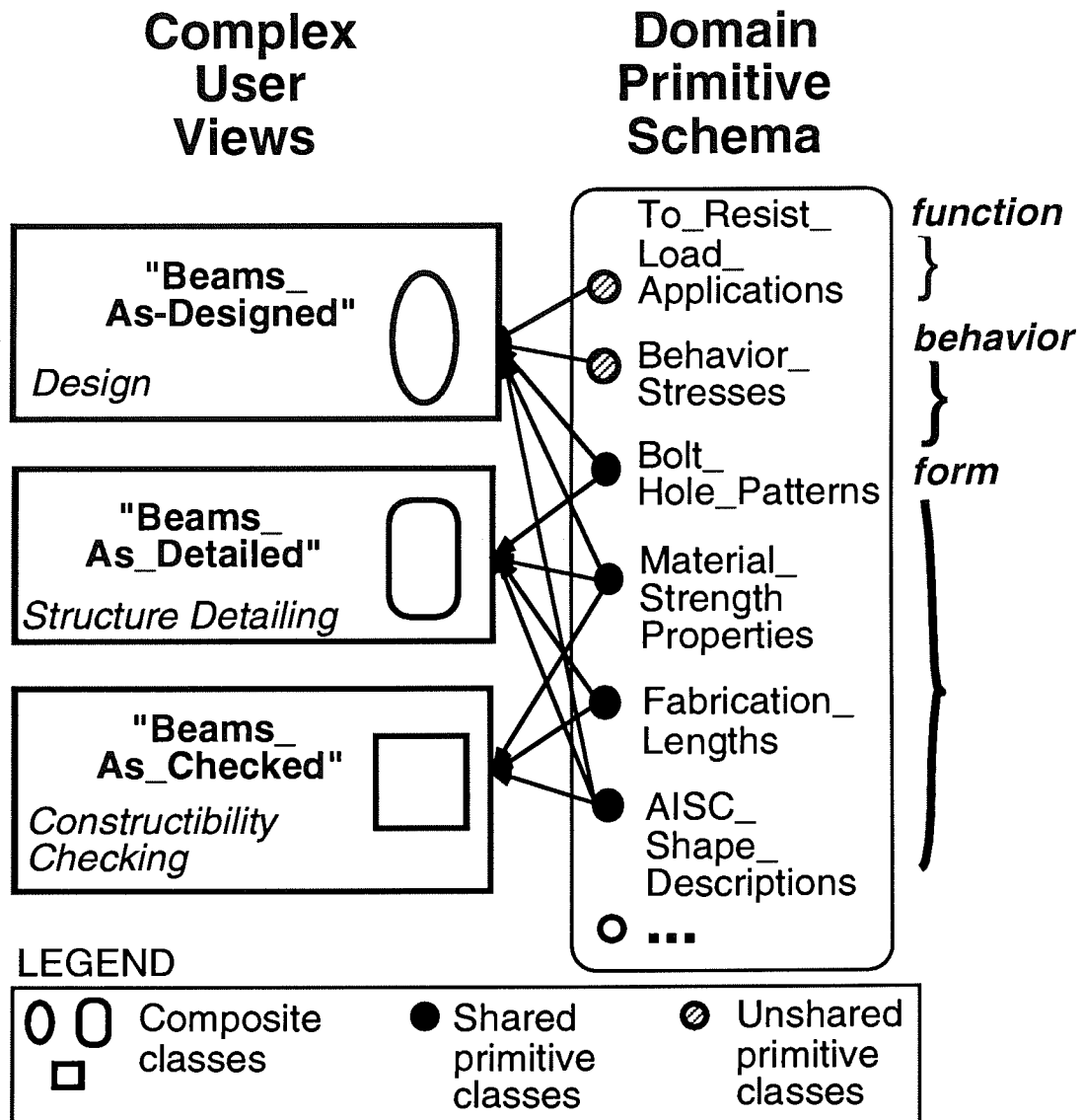


FIGURE 1.3: Customizing Composite Classes from a Primitive Schema.

### **1.3 Research Objectives**

---

To produce a working solution, this research has three specific objectives:

1. *To develop the P-C Approach fully:* This objective involves developing the P-C Approach by defining the requirements for designing a schema representing a given domain, identifying the analysis and design steps of the approach, and developing the modeling tools used in the steps that lead to the development of the schema meeting those requirements.
2. *To apply the P-C Approach to a real-life facility engineering domain:* This objective involves applying the P-C Approach to a real-life facility engineering domain and thereby demonstrating its usefulness. This application includes the development of a functional schema of the engineering process in the selected domain, as well as a database schema of the domain. It also entails investigating the form, function, and behavior of engineering design objects in the domain, and the incorporation of these representations into the database schema. In addition, this application includes building a database of a selected tower in the domain using the database schema.
3. *To test the P-C Approach:* This objective involves defining a method for testing the approach, conducting the test in the transmission tower domain, obtaining test results, and drawing conclusions about the approach's scope of applicability, strengths, and limitations.

### **1.4 Reader's Guide**

---

To provide readers with the options of reading in sequence or by selected topics, we organized the remainder of this dissertation as follows:

- Readers can consult Chapter 2 to gain more insight into the current state of knowledge in the fields of data modeling and data integration. This chapter describes the historical development of data models and explains the object-oriented data modeling paradigm that closely relates to this research. It also reviews existing work in facility data modeling and discusses the difficulties in this area. Finally, it presents current data integration approaches and system architectures proposed to achieve better data integration.
- Those interested in the P-C Approach will find explanation of the approach in Chapter 3. This chapter begins with a perspective on the development of the P-C Approach, including a description of the research methodology used. The chapter then explains the way the approach can be executed as a structured and coherent methodology for modeling facility data.
- Practitioners who share the goal of modeling facility data to support data integration can read Chapters 4, 5, and 6 to understand the modeling tools used in this approach. Chapter 4 describes the PArTitioned eNginEering DAta flow model (or PANDA), an extension of the Data Flow model developed for modeling facility engineering processes. This chapter explains PANDA's concepts, graphical representations, syntactic and semantic rules, and schema transformation operations, as well as a customized method and guidelines for using the model. Appendix A describes the domain of electrical utility transmission towers to which we applied

the P-C Approach and then shows the graphical functional schemata of the tower engineering process using PANDA.

Chapter 5 presents the Domain Entities AnaLysis method (or DEAL). This methodology was developed for decomposing entities used by experts in a given domain into primitive entities based on the criteria of cohesion and reusability. This chapter explains the concepts and terms, assumptions, graphical representations, rules, procedures and operations included in DEAL. Appendix B shows a detailed analysis of the “Transmission Tower Members” entity.

Chapter 6 describes the Primitive-Composite (or P-C) Data Model and Method, both of which are used to design the common object-oriented schema of the domain being modeled, called the domain primitive schema. This chapter explains the concepts and relationship types included in the model. It also presents the steps, rules, and guidelines of the accompanying method, which are necessary to design the domain primitive schema based on the concepts of the model. Appendix C provides the full documentation for the rules and guidelines of the method.

- Those who are interested in data representation should refer to Chapter 7. This chapter discusses how form, function, and behavior are represented in the domain primitive schema of transmission towers. Appendix D contains the documentation of the primitive classes of that schema.
- The test method in Chapter 8 should prove useful to those who wish to develop operational measures for testing and accepting data standards or schemata supporting data integration. This chapter explains how the P-C Approach was tested in the transmission tower domain. It also presents the test results. Appendix E provides the documentation of the composite classes defined as a result of the testing.
- Chapter 9 presents the contributions, conclusions, and implications of this research for data modeling and data exchange, as well as future research directions.





# **Chapter 2**

---

## **Field of Study**

### **Chapter Abstract:**

This chapter describes the two subjects studied in this research: data modeling and data integration. The aim of this chapter is to explain this research's point of departure. This chapter first introduces the data modeling terminology that will be used hereafter, and describes the historical development of data models. It also explains object-oriented data modeling, which closely relates to this research, and defines some fundamental object-oriented concepts. It then reviews existing work in data modeling for facility engineering and discusses the difficulties in this area. Finally, it presents different approaches to data integration.

### **Organization:**

#### *2.1 Background on Data Modeling*

2.1.1 Terminology: From "datum" to "data modeling"

2.1.2 Historical Development of Data Models

2.1.3 Object-Oriented Data Modeling Paradigm: Models and Concepts

#### *2.2 Background on Data Modeling for Facility Engineering*

2.2.1 Existing Work in Facility Data Modeling

2.2.2 Difficulties in Facility Data Modeling

#### *2.3 Background on Data Integration*

#### *2.4 Chapter Summary*

## **2.1 Background on Data Modeling**

---

According to Parsaye et al. [89], programmers in the early days of computer usage were dissatisfied with the limited data storage capability of computers and wanted their programs to have better means of handling data. In some applications, the need to handle data was so crucial that special programs were developed for this purpose. Databases were then created. Data structures in early programs involved simple data types such as integers, real numbers, and character strings. Then came more complex structures such as stacks, queues, arrays, lists, and files. In addition, the advancement in computer technology offered more economic means of electronic data storage. Consequently, data management had to evolve in order to keep up with these improvements. Better tools to represent and manage data have been introduced, leading to the establishment of the field of data modeling.

### **2.1.1 Terminology: From "datum" to "data modeling"**

A "datum" is a unit of information that corresponds to a discrete fact. Since data relate to specific instances of recorded factual information, they include much detail and change rapidly over time. In addition, they are numerous in quantity and may be stored in database systems [Wiederhold 86a].

A "data model" is a collection of conceptual tools for describing data semantics, relationships among data, constraints on the data definition, and operations on the data [Tsichritzis 82]. A "database schema" is a logical set of data structures and their properties, constraints, and relationships that is well-defined for a certain application using a particular data model.

A "database" is a formal collection of related data organized according to a pre-defined schema [Tsichritzis 82]. Databases often serve the needs of a large community of users. This function imposes a number of requirements on the development of the database: an external view level to support distinct types of users or processes that share the data; a conceptual schema level that integrates these different view models; a logical schema level that includes computer-processable specifications of conceptual schemata; and an internal physical level for persistent data storage [Brodie 84a].

A "database management system" (abbreviated as DBMS) provides facilities and tools for defining, manipulating, and controlling information in the database, including a protection mechanism for data and users. Aspects of the database control include semantic integrity maintenance, security in terms of user authorization, concurrency for multiple users, and recovery in the event of a failure.

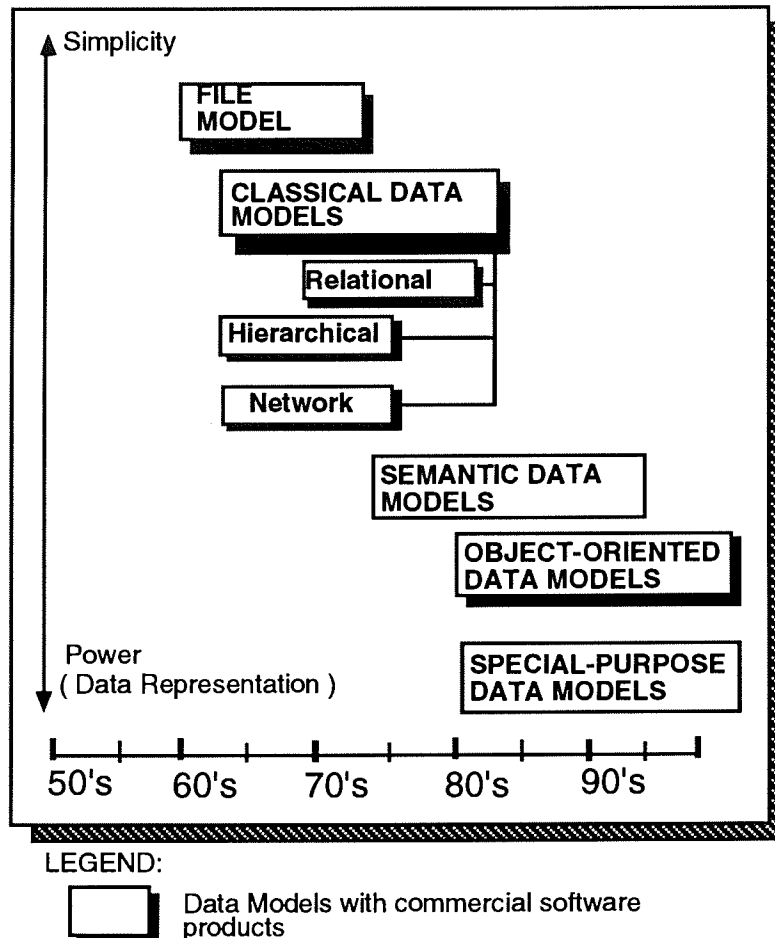
### **2.1.2 Historical Development of Data Models**

The following material is based on the literature on the data models' development, which can be found in [Tsichritzis 82], [Brodie 84a], [Wiederhold 83], [Wiederhold 84], [Parsaye 89], [Cardenas 90], [Date 90], [Kim 90], [Navathe 92], [Kim 93], [Hurson 93], etc.

**File Models** As illustrated in Figure 2.1, the "file models" date back to the earliest work in data modeling. These models correspond to files that contain organized data records. Data operations are limited to read and write operations over records [Brodie 84a]. In addition, there is little separation among the external view level, the conceptual schema level, the logical schema level, and the internal physical level of the file model. These limitations motivated the development of the latter data models. The "inverted list

model” was the most recognized model in this category. It had a commercial product implementation from Computer Associates for IBM mainframes (CA-DACOM/DB).

**Classical Data Models** The so-called “classical” or “traditional data models,” which started in the sixties, have prevailed for almost three decades. These models include the “hierarchical” data model [Tsichritzis 76], the “network” data model [CODASYL 71], and the “relational” data model [Codd 70]. In the sixties, many business applications were written in hierarchical database systems. Commercial products of hierarchical DBMS include the INTEL/MRL SYSTEM 2000 [Tsichritzis 76] and the IBM Information Management System (IMS) [McGee 77]. In the seventies, the network data model became popular. General Electric’s Integrated Data Store (IDS) was the first implementation of a network DBMS [Parsaye 89]. According to Date [90], the Integrated Database Management System (IDMS) is the best known commercial product from Computer Associates for IBM mainframes. In 1970, Codd introduced the relational data model in a series of seminal papers [Codd 70], [Codd 71a], [Codd 71b], [Codd 72a], [Codd 72b]. This model became the data model of choice for business applications in the eighties. After starting from a number of early research systems such as System R [Astrahan 76] and INGRES [Stonebraker 76], commercial relational database management systems continue to proliferate and greatly dominate the marketplace.



**FIGURE 2.1: Historical Development of Data Models.** This taxonomy of the file, classical, semantic, object-oriented, and special-purpose data models is based on the one presented in [Brodie 84a].

**Semantic Data Models** Paralleling the introduction of the relational model, a different research direction in data modeling emerged with the development of “semantic data models” [Brodie 84a]. This development was motivated by the need to facilitate the design of database schemata and to develop more semantically accurate data models [Zdonik 90]. Examples of these semantics include general factual information about the world (e.g., all columns are in a vertical position, the *subpart* relationship is transitive, etc.), events and their sequencing and times of occurrence, and generalization hierarchies which enable inheritance of properties and behavior from superclasses to subclasses [Reiter 84]. These semantics are not easily expressed in terms of records or relations. In fact, the semantic data models reacted against the simplicity of the earlier models whose representational limitation lay in the use of simple data structures. This limitation reduces the expressive power of the data model and creates the risk of losing information. In addition, the semantic data modeling movement complemented the work on knowledge representation in AI, particularly the semantic network and frame-based representation schemes [Zdonik 90]. According to Hull and King [90], the first published model was the Semantic Binary data model [Abrial 74]. Thereafter, scholars proposed a large number of semantic data models: the Infological data model [Sundgren 74], the Entity-Relationship (E-R) data model [Chen 76], the Extended RM/T data model [Codd 79], the Structural data model [Wiederhold 80], TAXIS [Mylopoulos 80], DAPLEX [Shipman 81], SDM [Hammer 81], the Event data model [King 84], SHM+ [Brodie 84b], etc. Although none of these data models have been commercially implemented, they established an important framework for the later development of object-oriented data models.

**Object-Oriented Data Models** Starting in the eighties, the increasing number of computer-aided design, engineering, software engineering, and manufacturing applications revealed the shortcomings of the relational and earlier database technology [Kim 90]. “Object-oriented data models” emerged as a new breed of models that could overcome the shortcomings of their predecessors. The driving force behind their development was the motivation to respond to the data needs of new types of applications and to incorporate more semantics into the data model (as in the case of semantic data models). Today, commercially object-oriented database management systems (e.g., *Gemstone* [Maier 86], *Versant* [Loomis 87], *ONTOS* [Andrews 90], *ITASCA* [Itasca 90], *Objectivity* [Objectivity 91], *ObjectStore* [Object Design 91]) have steadily increased in number and in database market share. These particular systems have been developed from object-oriented programming languages. Other systems with object-oriented features (e.g., *POSTGRES* [Stonebraker 86a], [Stonebraker 86b], [Rowe 87], *Sembase* [King 86], *Rose* [Spooner 86], [Hardwick 89], *PROBE* [Dayal 87], *IRIS* [Fishman 87], *EXTRA/EXCESS* [Carey 88], *PENGUIN* [Law 89], *Semantic Information Manager system (SIM)* [Fritchman 90] ) have been developed using a relational DBMS as the data storage subsystem. Since this research makes use of object-oriented data modeling, the next section will discuss this category in greater detail. Reviews of object-oriented DBMS can be found in [Ahmed 90] and [Hurson 93].

**Special-Purpose Data Models** To date, the field of data modeling has experienced an increasing need for more “special-purpose” data models. As opposed to the general-purpose data models in the above categories, these models are dedicated to specific domain areas (particularly in engineering), or to particular application types such as CAE, CAD, CAM, VLSI design, office automation, geographic information systems, etc. Section 2.2.1 will introduce existing data models for facility engineering.

### 2.1.3 Object-Oriented Data Modeling Paradigm: Models and Concepts

#### 2.1.3.1 Object-Oriented Data Models

The development of object-oriented data models combines important concepts from many areas. First, the influence of programming languages is evident. Key concepts from pioneering object-oriented programming languages such as *Alphard* [Wulf 76], *Simula-67* [Dahl 70], and *Smalltalk* [Goldberg 85] greatly influence this model. Second, powerful knowledge representation techniques from AI are utilized in object-oriented data modeling. These techniques include frame representation, classification hierarchies, delegation of behavior, and dynamic binding of messages to methods. Third, the preceding semantic data models strongly influenced the development of these data models.

Although there is no universal consensus on what the object-oriented data model is, object-oriented programming languages, knowledge representation languages, and semantic data models share a number of key object-oriented concepts. References on these concepts can be found in [Booch 86], [Cox 86], [Meyer 88], [Stefik 88], [Stroustrup 88], [Abdalla 89], [Atkinson 89], [Rettig 89], [Thomas 89], [Kim 90], [Meng 90], [Booch 91], and [Rumbaugh 91]. The next section introduces these concepts.

#### 2.1.3.2 Key Object-Oriented Concepts

The key object-oriented concepts include:

**Class (or Object Class)**—A “class” is a set of similar objects that exhibit some common properties and behavior. The formal definition of a class specifies the descriptive features that are shared by the members (or “instances”) of the class. Properties of a class are defined in terms of “attributes,” while behavior is defined in terms of “methods.” “Encapsulation” refers to the method of enclosing properties and behavior of the object class within its boundary. Further, a class can have several “subclasses” that inherit its properties and behavior. It is then called a “superclass.” This method of “inheritance” is implemented by using the ubiquitous *is a* relationship. There are two kinds of inheritance: “single inheritance” and “multiple inheritance.” In the former, a class has at most one superclass; in the latter, a class can have more than one superclass. Classes that are related to one another by the *is a* relationship form a “class hierarchy.” Examples of common classes in facility engineering are “Beams,” “Walls,” and “Loads.”

**Instance (or Object)**—An “instance” is a unique occurrence of an object class. For example, “beam23” is an instance of the class “Beams.” An instance has an “identity” that uniquely distinguishes it from all other instances. It exhibits all attributes and access methods of its class. At any time, the instance has a “state” that is described by its attributes and their current values.

**Attribute**—An “attribute” is a formal property of an object class. The attribute definition in the object class specifies a static property true for all instances of that class, while the “attribute value” of a particular instance describes a dynamic property in the current state of that instance. For example, the class “Beams” might have attributes such as depth, maximum-moment, and stiffeners-required. The attribute values of the instance “beam23” are: depth of 14.0 inch, maximum-moment of 50 kip-inches, and stiffeners-required of TRUE.

Each attribute value is associated with a data type. Data types can be basic types (e.g., integers, real numbers, characters, character strings) or more complex types such as

arrays, bags, sets, lists, or user-defined abstract data types. Useful abstract data types include three-dimensional coordinates (a triple of coordinate values), direction, orientation, date, or time. An attribute value can also be a reference pointer to another object. In this case, the attribute is called a “relationship attribute.” A “default attribute” has a predefined attribute value used for every instance where the user does not specify a value.

Two other types of attributes are of interest. “Independent attributes” are stored and defined independent of any other attributes. Most attributes of the object class are of this type. On the other hand, “derived attributes” are dependent on other attributes; a method is usually defined to compute the dependent attribute, which is derived on demand.

**Method**— “Methods” are operations defined on the object class in order to exhibit the behavior of a class. They are also the only means of modifying an object state [Abdalla 89]. Objects communicate with one another through “messages,” which are sent in order to provide data, request data, or trigger certain actions. The message receiver responds to a message by triggering the corresponding method.

### **2.1.3.3 Why Object-Oriented Data Modeling in this Research?**

The application of the P-C Approach to a domain results in an object-oriented primitive schema of the domain. This schema contains hierarchies of primitive object classes, from which primitive instances can be created. The following are reasons for adopting the object-oriented paradigm in this research:

- *Advantages in Representing Engineering Data:* An object class definition can include not only simple data types such as integers, real numbers, etc., but also complex, nested data structures such as lists, arrays, sets, bags, abstract data types, and logical pointers to other objects. In addition, each object class can encapsulate all the properties and behavior of the real-life object that it represents.
- *Explicit Incorporation of Powerful Semantic Modeling Concepts:* These concepts include generalization, classification, inheritance, encapsulation, typing, polymorphism, etc. They originated from the work on object-oriented programming languages and on knowledge representation in AI, and from the development of the semantic data models.
- *Support of Object Identity:* Each instance can be distinctly identified in the database by its own identity. This identity is globally unique to the instance, independent of the physical location of the instance, and created and maintained by the system [Khoshafian 86]. Consequently, users are not forced to associate every instance with some fictitious identifier.
- *Support of Distinguishable Semantic-Rich Relationships:* Relationships such as aggregation and association among instances carry their own unique semantics and are recognized as integral, but distinguishable parts of an object-oriented data model.
- *Object Uniformity:* All data, including meta-level data, are represented as objects in the database [Cardenas 90]. Therefore, information about the schema-level definitions and management of object classes can be stored in the database.
- *Data Access Convenience:* Data access from instances to instances can directly follow the predefined link paths among the instances. Consequently, access to data along these paths is easy to specify and efficient to perform. (This is an advantage

when these existing access paths are so desired, but also a drawback when other access paths are preferred.)

As documented in literature on engineering data modeling [Batory 76], [Lorie 83], , [Kersten 86], [Wiederhold 86b], [Ketabchi 88], [Abdalla 89], [Barsalou 90], [Abdalla 92], [Froese 92], [Kim 93], the above features are beneficial to the representation of engineering design objects and to the later implementation of engineering databases.

## **2.2 Background on Data Modeling for Facility Engineering**

### **2.2.1 Existing Work in Facility Data Modeling**

Over the years, a number of models and schemata have been proposed for facility engineering. These models are based on or influenced by general-purpose data models such as the Entity-Relationship model, semantic data models, and object-oriented data models, each of which have their own flavor. There are three observable categories of models: models with distinct hierarchical levels of aggregation, models with multiple linked hierarchies, and models for A/E/C product data exchange.<sup>11</sup>

The following sections present the three categories and their general characteristics. These sections also review the data models in each category by describing their salient features and providing a summary statement about the model at the end.

#### **2.2.1.1 Hierarchical Models with Distinct Levels of Aggregation**

These models articulate distinct levels of aggregation within a single hierarchy in order to organize facility engineering data. The *part of* relationship and its inverse are the primary relationships defined among the entities in these levels. This is probably the most obvious approach to modeling. Two data models that are typical of this category are the Structural Steel Framing Data Model [Lavakare 89] and the Ratas Building Product Model [Bjork 88], [Bjork 89a], [Bjork 89a].

**Structural Steel Framing Data Model (SSFDM)** This model [Lavakare 89] is an object-oriented data model for steel-framed structures. The model employs object-oriented concepts as well as extensions to the Entity-Relationship data model [Chen 76]. First, it utilizes a taxonomy of “generic entities” (equivalent to object classes), “instantiated entities” (equivalent to object instances), and “typical entities” (which function as an intermediate entity level between generic and instantiated entities). Second, relationships such as “is-a,” “instance,” “part-of,” “connected-to,” and “associated-with” are defined in order to represent the various ways in which entities of the model can be related to one another. The entire model is structured according to eight hierarchical levels of aggregation from the highest level of “Buildings” to the most detailed levels of “Parts” and “Connections.” Each level includes a number of entity definitions. In short, this is a detailed model that can be used for steel-framed structures.

**Ratas Building Product Model** This model [Bjork 88], [Bjork 89a], [Bjork 89a] is also an object-oriented data model for building product design. It has evolved from a national cooperative study known as the Ratas project sponsored by professional organizations in the building industry in Finland. The model reiterates the idea of a

---

<sup>11</sup> Not all of these models fit into the formal definition of a “data model” as presented in Section 2.1.1.



“product model” as the database description of a product. It also makes use of the concept of “abstraction hierarchies” and follows a similar approach by defining five hierarchical levels of aggregation from the “Building” level to the “Parts” and “Details” levels. In short, this model includes a number of high level entities common to general building product design.

**Others** Another model in this category is the *Component-Connection Abstraction Model* [Powell 88]. Its main building block is the component hierarchy, which consists of basic components (e.g., column, footing, and girder) and connections. Overall, it is a general conceptual model for representing facility engineering objects. However, it does not articulate a clear framework within which the component hierarchy can be implemented.

### **2.2.1.2 Models with Multiple Linked Aggregation Hierarchies**

The unique characteristics of these models lie in their use of multiple distinct aggregation hierarchies to support different engineering views about the data and the unification of these aggregation hierarchies into a single hierarchy. Two prominent data models in this category are the Data Model for Building Design [Law 86] and Eastman's 1978 model [Eastman 78].

**Data Model For Building Design** This model [Law 86] is based on the so-called “abstract” data model [Smith 77], which advocates using “abstract objects” as primitives in the database design. The model employs the aggregation and generalization abstraction methods. The relationships used in the model include the *part of* relationship and the *is a* relationship. The model consists of three hierarchies, each of which is constructed using the aggregation method. First, the topological hierarchy describes the spatial characteristics of the building in terms of the connectivities among its elements. Second, the structural hierarchy includes common structural components of a building. Third, the architectural hierarchy corresponds to the architectural designer's view of the building in terms of spatial divisions and architectural functions. These three view-related hierarchies are unified into a single hierarchy with the “Building” entity as the root node. Overall, this is a general data model used for building design that supports separation between the spatial, structural engineering, and architectural views.

**Eastman's 1978 Model** This is the model proposed by Eastman in 1978 [Eastman 78] for building design. This model contains two abstraction hierarchies. The spatial hierarchy starts with floor levels that are divided into spaces, interior walls, and exterior walls. The latter are further decomposed at the lower levels into more detailed components. An example of a detailed component is a concrete block, metal stud, or even a construction method. In the functional hierarchy, frames can be aggregated into bays, aisles, joints, etc. The two hierarchies are unified with the “Building” entity at the root node. In short, the model is very abstract and biased toward the architectural view of the building. It does not articulate the structural engineering view.

### **2.2.1.3 Models for A/E/C Product Data Exchange**

In the area of data exchange standards, the STandard for the Exchange of Product model data (STEP) was the first international standard under the sponsorship of the the International Standards Organization (ISO). Today, the STEP development involves organizations, agencies, and companies throughout Europe and in the United States. The

Product Data Exchange using STEP (PDES) is the major contributor to STEP from the United States (see [Warthen 88] and [Warthen 90] for background). Formerly the Product Data Exchange Specification, PDES was an outgrowth of the Initial Graphics Exchange Specification (IGES) standard [IGES 90] in the United States. In 1988, the ISO STEP Committee adopted the first version of the PDES standard as an initial draft of STEP. The PDES/STEP joint effort for an international standard was then initiated. The work of the PDES/STEP members has contributed toward the Information Product Integration Model (IPIM) [Wilson 88]. The current PDES/STEP goal is to develop standards for exchanging data to support integration of database, CAD, and knowledge base systems. These evolving standards follow the standard exchange format approach described in Chapter 1 and rely heavily on defining complex product models [Howard 89b].

PDES/STEP is the prime contributor of models for product data exchange in the Architecture/Engineering/Construction (A/E/C) industry. These models include the following submodels: building systems, ship structures, ship outfitting, plant design, distribution systems, and general A/E/C reference. The subsequent subsections review the A/E/C product data exchange submodels of building systems and ship structures.

**General A/E/C Reference Model (GARM)** This model [Gielingh 88] provides a general reference for modeling engineering products to support their design, production, and life cycle maintenance. The model introduces a number of important concepts for A/E/C product data modeling. The most fundamental building block of the model is the entity "Product-Definition-Unit" (PDU). A PDU can represent a design product, a system, a subsystem, a part, a part feature, etc. It includes properties that describe different aspects of the product. A PDU has different life cycle stages: as-required, as-designed, as-planned, as-built, as-used, as-altered, and as-demolished. Second, the product definition has three levels: generic, specific, and occurrence. Also, the model identifies three fundamental abstraction methods: generalization, aggregation, and characterization. Overall, this model establishes an important conceptual framework for developing more specific A/E/C data exchange models.

**NIDDESC Ship Structural Information Model** This model [Gerardi 88] is a more detailed data model used exclusively for ship structures. The Navy/Industry Digital Data Exchange Standards Committee (NIDDESC) is a cooperative effort between the Navy Sea Systems Command and the Marine Industry in the National Shipbuilding Research Program. This model includes many detailed entity definitions for ship structures. Those definitions include the following: the high-level descriptions of ship structures; the ship's construction project and site; the ship's breakdown into systems, assemblies and subassemblies; and the detailed description of parts, structural joints, openings, and other part features. In particular, the model includes entity definitions for ship geometry, topology, and material. In short, this model contributes detailed entity definitions that apply to ship structures, but that can also be used for other types of structures.

**Others** The *A/E/C Building System Model* [Turner 88] is a high-level model for general building systems. However, it does not present a coherent conceptual framework and is predominantly influenced by the architectural view of a building. The *Logical Product Model for Structural Steelwork* [LPM 90] is another product data exchange model for computer-integrated manufacturing applications of construction steel work. The model is still in an early stage of development. The *Shared Object Libraries (SOL)* developed by Froese [92] aims at establishing standard data representations for construction and project management. The SOL data model includes concepts such as object, attribute, relationship, attribute set, relationship set, etc. Froese contributed to the area of data integration by showing the object-oriented paradigm's usefulness in the development of

integrated computer-aided project management applications. Indeed, he built a database system, the *SOL OODBMS*, and defined a domain-specific schema for project management and construction. He demonstrated a prototype construction project planning system, the *Object-model-based Project Information System (OPIS)*, that consists of applications supporting a variety of planning tasks.

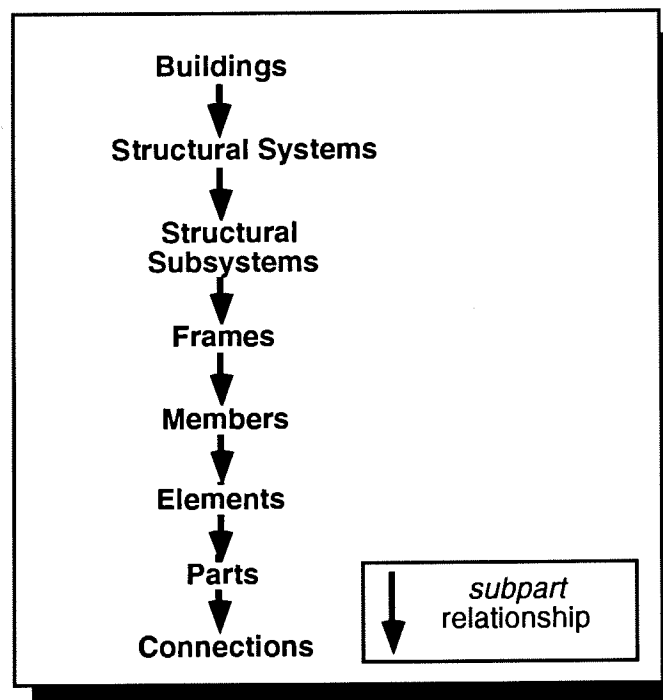
### **2.2.2 Difficulties in Facility Data Modeling**

The previous review of existing data models for facility engineering reveals the following difficulties of modeling facility engineering data:

- *Limited Ability to Support Multiple User Views:* Existing models tend to predefine a number of expected descriptions of the facility and do not allow users to customize their own views. Rigid, pre-defined hierarchical levels of aggregation (as in the first category of data models presented earlier) further limit that ability.
- *Lack of Support for Schema Evolution:* The majority of existing models are schemata that, once defined, cannot be extended gracefully to accommodate the evolving phases of the facility life cycle. By “gracefully,” we mean extension of the schema without abandonment of previously defined data representations.
- *Lack of Form, Function, and Behavior Representations:* Existing models include mainly form descriptions and do not distinguish form, function, and behavior. The inclusion and distinction of form, function, and behavior are essential to describing the facility design objects and to modeling the way in which facility engineers come up with their designs [Gruber 90a], [Jain 90], [Kuffner 91], [Luth 91], [Ullman 91a].
- *Lack of Support for Data Integration:* Existing models do not fully respond to the need of data integration. Only PDES/STEP A/E/C product data exchange models are currently contributing to the data integration effort; however, they were developed primarily as standardized schemata for the purpose of exchanging data, as formal data models. Moreover, the future usefulness of the PDES/STEP data exchange standard remains to be proven. Reed [88] also raised the following challenges in the PDES/STEP data exchange standard development: (1) the accommodation of multiple representations of building product data as well as canonical transformations among these representations; (2) principles to ensure minimal redundancy in exchanged data sets; (3) a mechanism to define explicitly and to exchange data constraints and methods; (4) guidelines to vendors for implementing translators for these models; and (5) guidelines and time frames for completing a global building product model.
- *Lack of Object-Oriented Modeling Methods:* The object-oriented paradigm provides useful concepts and techniques for modeling data in the real world. However, it leaves up to the modeler the task of deciding how to represent the problem domain in terms of object classes; how to organize object classes into class hierarchies; how to determine the attributes, number of attributes, and methods of each class; and so on. While some rules for making these decisions are imposed by the nature of the modeling problem at hand, others are dictated by the available constructs in the programming language or system in use. Unlike the relational data model that provides normalization techniques, object-oriented data models lack rigorous and accepted methods to guide the modeler in developing a good database schema design. Similarly, the existing facility engineering data models that are based on object-oriented data models do not provide such a method.

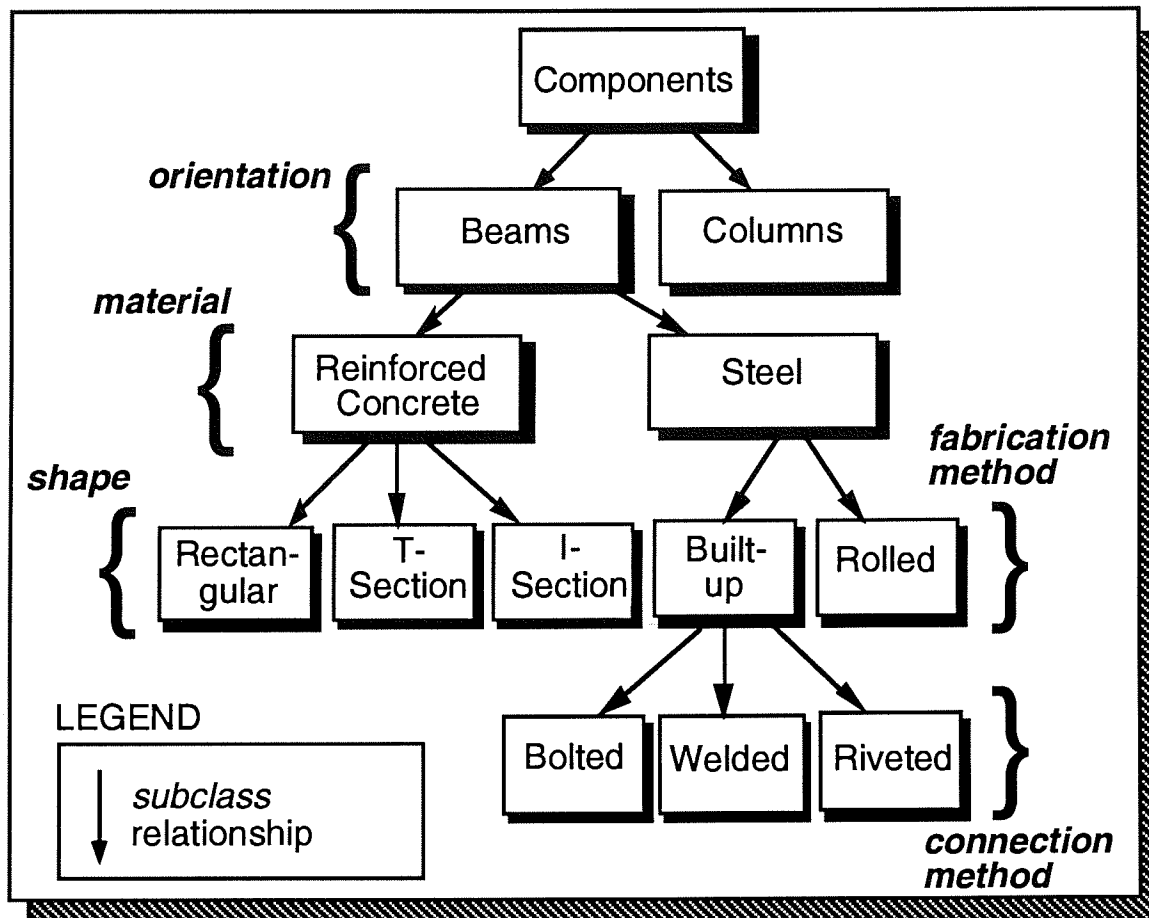
- *Lack of Requirements, Criteria, and Measures to Test and Evaluate Data Models and Database Schemata:* Little work has been done in the area of defining requirements, criteria, and measures to test and evaluate data models, database schemata, and data standards. Acceptance of a particular model has been primarily based on experience in using it. Database schemata and systems have either been accepted or rejected by users after they were developed. To our knowledge, no rigorous measures have been defined for testing how effectively a data standard can be shared within a domain and across domains and thus, for accepting that standard.
- *Common Facility Data Representation Traps:* A study of existing data models and application schemata proposed for facility engineering [Phan 91b] revealed a number of common representation traps. The three most critical traps are: misuse of aggregation, non-homogeneous class hierarchies, and large object clusters. The following section explains these representation traps.

**Misuse of Aggregation** “Aggregation” is a common method for constructing complex engineering objects from their components. Inversely, “decomposition” breaks down complex engineering objects into their components. The *part of* relationship and its inverse *subpart* relationship are used for these methods respectively. Aggregation is commonly used in engineering data modeling since engineering data tends to be hierarchical in nature [Ketabchi 86]. In fact, the first category of facility engineering data models discussed earlier advocates the use of clearly defined levels of aggregation. For instance, Figure 2.2 illustrates the aggregation levels of the Structural Steel Framing Data Model [Lavakare 89]. Although the importance of aggregation cannot be overlooked, it can also be easily misused. For example, *part of* relates two objects "structural-member-3" and "structural-element-3" whose relationship type is not aggregation. In actuality, these objects represent different aspects (i.e., functional aspect and structural analysis aspect) of the same object "beam3." Their relationship should be an association.



**FIGURE 2.2: Aggregation Levels in the Structural Steel Framing Data Model [Lavakare 90].** (Adapted from Howard et al. [92])

**Non-homogeneous Class Hierarchies** This phenomenon occurs when many different criteria are used to define subclasses at different levels of the class hierarchy, as shown in Figure 2.3. This is an undesirable feature for two reasons. First, since different views and semantics are mixed in an indiscriminate manner in the construction of the class hierarchy, this leads to a poor representation with minimal separation between the different aspects (i.e., form, function, and behavior) of the object class description. Second, the use of such non-homogeneous hierarchies requires the modeler to anticipate all possible combinations of subclasses. This can lead to what is called the "cross product phenomenon": when the number of criteria used in a single hierarchy increases, the resulting model grows exponentially larger, and the modeler must supply the expertise to eliminate subclasses that represent invalid combinations (and possibly some valid combinations as well) [Howard 92].



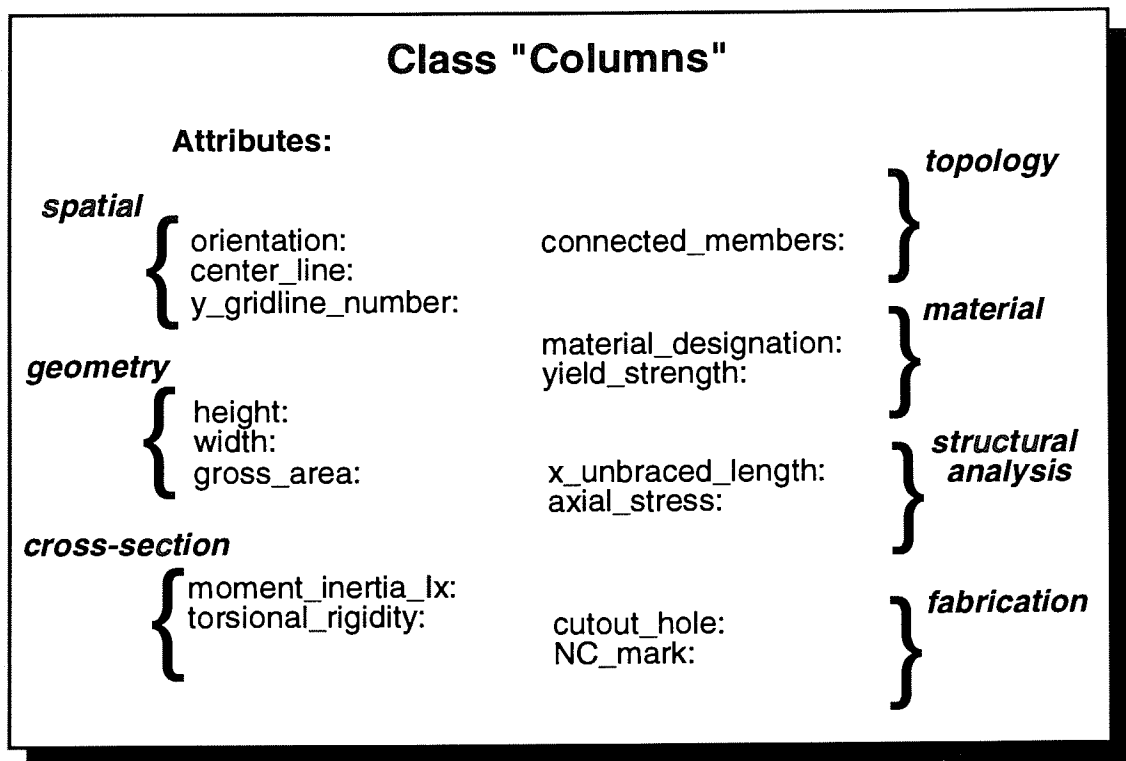
**FIGURE 2.3: A Sample Non-Homogeneous Class Hierarchy.**

**Large Object Clusters** The need to accommodate many users and the convenience of putting many attributes into one object class presents another common trap. "Large object clusters" are defined here as highly complex object class definitions that include multiple aspects of description in order to satisfy all user needs. As an example, consider an object class "Columns" as shown in Figure 2.4. These large object clusters create a number of problems:

- First, these object clusters are inefficient from a designer's point of view. As Minsky put it:

*"... in such a complex problem, one can never cope with many details at once. At each moment one must work within a reasonably simple framework... any problem that a person can solve at all is worked out at each moment in a small context and that the key operations in problem-solving are concerned with finding or constructing these working environments."* [Minsky 75]

- Second, it is difficult to create or modify instances of these large object classes, and to manage their versions. Instantiating such an object class demands attribute values that are not all defined at the same time. Update and version management of such instantiated objects are difficult since each instance involves a large cluster of attributes that are not all modified.
- Third, a schema with only a few large objects is too rigid and cannot accommodate future evolution of the schema as the design progresses. In addition, such a schema is difficult to reuse such a schema in other applications.



**FIGURE 2.4: A Sample Large Object Cluster.**

- Fourth, exchanging large objects among applications requires parsing out the needed data, which can be highly inefficient and computationally costly.
- Finally, the semantic composition of such large objects becomes incoherent and, thus, poorly defined. In some cases, it remains unclear whether certain attributes truly belong to the object, or merely describe some properties that should belong to other object definitions.

### **2.3 Background on Data Integration**

In recent years, integration has received much attention in manufacturing and engineering domains and has attracted research and development from many areas. Integration refers to the coordination of all phases and aspects of a given process through the cooperative use of information from computer-based systems. More recently, research and development on “Enterprise Integration” in the field of Artificial Intelligence (AI) has focused on acquiring an understanding of how an enterprise operates and how information plays a role in supporting the enterprise’s operation. Work in this area is under way at several universities [Fox 92], [Jagannathan 92], [Srinivasan 92] and industry-funded research centers [Billmers 92], [Bradshaw 92], [Grosf 92], [Gruber 92] across the country. The general direction here is to explore advanced AI techniques to support enterprise modeling, automation and integration.

In particular, data integration focuses on improving the compatibility and reusability of data and data representations used in computer-based systems. Among different data integration approaches that have been suggested [Abdalla 89], the following has been pursued:

- *Direct Translator:* In this approach, a translator is written to directly translate the output of an application A to the input format required by another application B. The translation requires multiple output files from different modules within application A to generate a single input file to application B. The main disadvantage of this approach is the large number of translators (i.e.,  $N * (N - 1)$  translators for N applications) required to support data exchange among various applications.
- *Standard Exchange Format:* This approach uses neutral files specified according to a common standard for data exchange among all applications in the environment. These applications read input from and write output to files in the same standard format. Examples of these standards include the Initial Graphics Exchange Specification (IGES) [IGES 90], the STandard for the Exchange of Product model data (STEP) [Wilson 88], and the Electronic Data Interchange File (EDIF) [Andrews 88]. This approach has two major advantages over the first one. First, it eliminates the need to write a separate translator for each pair of distinct applications. Second, addition or removal of an application program has minimal effect on other programs in the environment. However, this approach also has some disadvantages. The major disadvantage is both the time and effort required to develop the standard for a given discipline as well as the communal agreement and commitment to such a standard. In addition, criteria for testing and accepting a new standard are hard to define and measure. Finally, in the case of STEP, the data exchange standard is defined in terms of a comprehensive “product data model.” All applications in the environment must understand the product data model in order to generate their own views. To exchange data among applications, the entire product data model must be transferred through a neutral file format. Each application then holds a copy of the same neutral file regardless of whether some or all of its data are appropriate to the application.

- *Central database:* In this approach, a central database serves as the data repository for all applications in the environment. A database management system manages data access to the central database. A strong advantage of this approach is that all applications in the environment share the same data access utility from the central database. In addition, the data base design can be administered in order to provide data views that closely match the application need. The difficulty in this approach lies in the design and implementation of such a central database that requires integrating all user views from different organizations, disciplines, and tasks. This approach also requires the applications to work with the central database. In this case, performance issues in data retrieval among several client applications and a central data server can be critical.
- *Multiple databases:* Different approaches have been proposed to manage multiple autonomous and possibly heterogeneous databases in distributed computing environments. The more notable approaches are Knowledge Aided Database Management (KADBASE) [Howard 86], [Howard 89c], database systems with mediators [Wiederhold 89], [Wiederhold 91], federated database systems [Sheth 90], distributed database systems [Thomas 90], and interoperable autonomous database systems [Litwin 90].

## ***2.4 Chapter Summary***

---

This chapter described the two subjects studied in this research: data modeling and data integration. The aim of this chapter was to explain this research's point of departure. First, different approaches to data integration have been proposed. In particular, the "standard exchange format approach" aims at providing an economical solution to this problem. This approach involves the development of standards for data exchange among computer applications. The advantage here is that experts and modelers need only collaborate once in defining these standards, but users can use them many times and in many domains. However, a number of issues face this approach. Among them is the need for suitable methods for the development of the standard for a given discipline. The first point of departure is that this research has a similar motivation for sharing data representations, but concentrates on how to model facility data and particularly on how to develop sharable data representations.

On the subject of data modeling, the relational model, which was introduced in 1970, became the data model of choice for business applications in the eighties. However, literature pointed out that this model has yet proven its effectiveness in supporting other types of applications such as those in engineering [Kersten 86]. Meanwhile, the development of semantic data models aimed at overcoming the limitations of the relational data model. An outgrowth of this development was object-oriented data models, which provide more powerful concepts to represent complex engineering data. The second point of departure is that this research uses object-oriented concepts to model facility data.

Finally, this research's third point of departure comes the work done in facility data modeling, mainly: the Structural Steel Framing Data Model [Lavakare 90], the General A/E/C Reference Model [Gielingh 88], the Data Model for Building Design [Law 86], and the SOL Data Model [Froese 92]. The review and evaluation of this work helped identify issues facing facility data modeling. The P-C Approach's development has been an attempt to address these issues. The second part of this dissertation presents this approach and its components. In particular, Chapter 3 provides an insight into the development of the approach. It also explains how the approach can be applied to a facility engineering domain to model data in support of data integration.





# **PART II: THE PROPOSED SOLUTION**

## **Chapter 3**

---

---

### **The Primitive-Composite Approach**

#### **Chapter Abstract:**

The Primitive-Composite (P-C) Approach is a structured and coherent methodology for analyzing a given facility engineering domain and for designing an object-oriented database schema of that domain to support data integration. This chapter provides a development perspective on the research project and an overview of this approach, and thereby sets the stage for the subsequent chapters. It begins by describing the research project that resulted in the development of the approach. At the same time, it describes the research methodology used in the project. Next, the chapter presents the approach. Finally, it explains the requirements, criteria, phases, and modeling tools included in the approach.

#### **Organization:**

##### *3.1 A Perspective on the Development of the P-C Approach*

##### *3.2 Overview of the P-C Approach*

###### *3.2.1 Requirements and Criteria*

###### *3.2.2 Phases and Modeling Tools*

##### *3.3 Evaluation of the P-C Approach*

###### *3.3.1 Scope of Applicability of the Approach*

###### *3.3.2 Strengths of the Approach*

###### *3.3.3 Limitations of the Approach*

##### *3.4 Chapter Summary*

### ***3.1 A Perspective on the Development of the P-C Approach***

---

The following paragraphs describe the chronological progression of the research project that resulted in the development of the P-C Approach. At the same time, they describe the research methodology used in this project. These descriptions help the reader understand the factors leading to the approach's development and provide useful information to those who are pursuing similar goals. The titles given to the paragraphs reflect our own recollections of the drawn-out, but eye-opening research journey.

***“Getting Drafted: The Early Experimental Years”*** The original motivation behind the research project was to develop facility database schemata that could be used in a related project on facility data exchange (i.e., KADBASE [Howard 89c]). The earliest attempt in this project was Lavakare and Howard's development of an object-oriented data model for steel framed structures, the Structural Steel Framing Data Model (SSFDM) [Lavakare 89]. We then applied the model to electrical utility transmission poles and wrote a detailed evaluation of the model [Phan 90]. Moreover, Jamal Abdalla, Jared Nedzel and we experimented with the modeling of facility data. In many brainstorming sessions, we discussed the issues involved in modeling facility data and exchanged ideas for potential solutions to these issues. Meanwhile, we reviewed extensively existing work in this research area. Later, we made an inventory of data representations used in eight different research projects at the Center For Integrated Facility Engineering (CIFE) at Stanford University. Using this inventory, we reported our findings concerning the problems of facility data modeling and integration in [Phan 91b]. All this experience provided the basis for defining the research problem presented in Chapter 1.

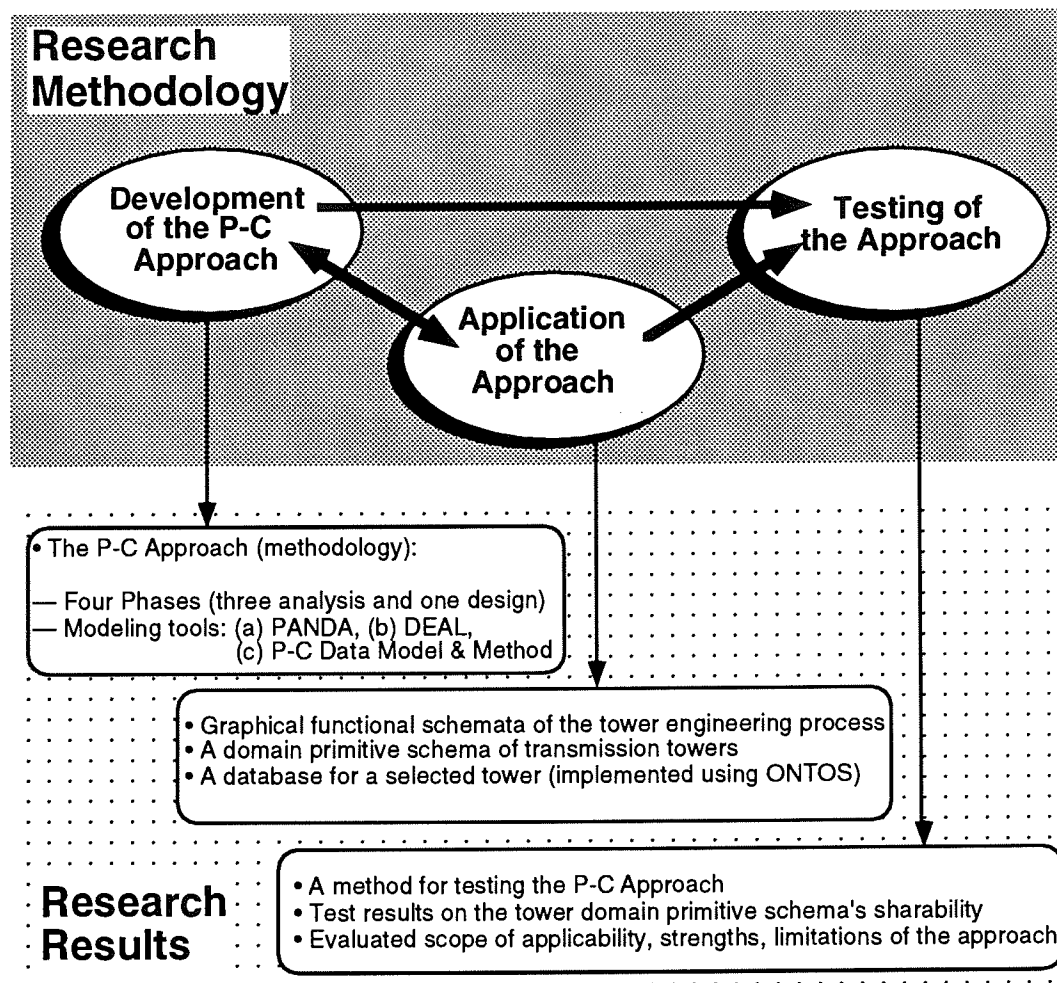
***“Understanding The Mission: The Research Problem”*** One important idea that evolved from the early brain-storming sessions was the use of primitives and composites to model facility data. The objective was to support data integration. This idea provided the basis for the early definition of the P-C Approach described in our first publication [Howard 92]. This idea was also presented in Chapter 1. However, in developing this approach, we faced three fundamental questions: “How is data actually used in a real-life facility engineering domain?,” “How can primitive entities be identified in a given domain?,” and then “How can these primitive entities be designed as object classes, from which composite classes can be defined?” The remainder of the research project comprised a tenacious attempt to provide answers to these questions.

***“Aiming High: The Research Methodology”*** With clear objectives in mind (see Chapter 1), we pursued three areas of development: (1) the development of the P-C Approach, (2) the application of the approach to a real-life tower engineering domain, and (3) the testing of the P-C Approach in the tower domain. In fact, these three areas constituted the research methodology used in this project. Figure 3.1 summarizes these areas and their interaction and results. We pursued the first two areas concurrently and worked on the third area when results from the first two became available. The next paragraph explains in more detail what has been done in each of these areas.

***“Down in the Trenches: The Making of the P-C Approach”*** To understand data use in facility engineering (i.e., the first question), we studied the domain of electrical utility transmission towers. we selected this domain for a number of reasons. First, the domain has all the characteristics of an engineering process described in Chapter 1:

multiple participants, a long facility life cycle, use of computer applications, complicated engineering processes, complex data, and above all, a critical need for data communication. In addition, transmission towers come in a wide range of sizes and levels of physical and engineering complexities. The large range of towers allowed us to choose a prototype structure that has the appropriate level of complexity for this research. My previous work experience in the field of utility structures design was another motivation. We interviewed engineers at a utility company who served as our domain experts. We collected and studied a large amount of documentation on transmission tower engineering processes.

From this study, we recognized the need for understanding complex engineering processes before modeling the data. We reviewed existing tools for functional analysis of processes that would give us that understanding. As a result, we selected the Data Flow model [Gane 79], [Yourdon 79], [De Marco 82], [Batini 92], which was simple and easy to use. Using this model, we began to model the tower engineering process. However, the sheer complexity of engineering processes put additional requirements on the model. Consequently, we developed an extension of the model for facility engineering, named the PARTitioned eNGineering DATA flow model (or PANDA). we applied PANDA to the tower domain and created a detailed set of functional schemata describing the tower engineering process.



**FIGURE 3.1: Research Methodology and Results.**

Using the information in these schemata, we analyzed data-intensive design objects such as tower leg members in the domain. We called the entities representing these design objects “domain entities.” Through studying data of these domain entities, we recognized the criteria of cohesion and reusability, with which we decomposed the domain entities into primitive entities. We then developed a method called DEAL to carry out the decomposition in a systematic manner. Finally, we developed the P-C Data Model and Method for refining and transforming those primitive entities into logical object classes. Using this model and method, we developed a primitive schema of the tower domain. Moreover, we reviewed literature on engineering form, function, and behavior representation, and identified the issues involved in this task. We then defined the positions taken and solutions used in the P-C Approach regarding these issues. Finally, we implemented a database of a selected tower using a commercial object-oriented database management system (ONTOS).

Moving to the third area of development, we defined a method for testing the P-C Approach in the tower domain and tested the approach. In the course of testing it, we recognized the scope of applicability, strengths, and limitations of the approach. We also identified future extensions of this research, including improvements of the approach.

**“Homecoming: The Final Package”** The final “package,” the P-C Approach, is more than just an object-oriented data model. This approach is a structured methodology for analyzing a given domain (i.e., both the engineering process and data used by domain experts) and for designing a primitive schema representing the domain. The next section describes the approach.

---

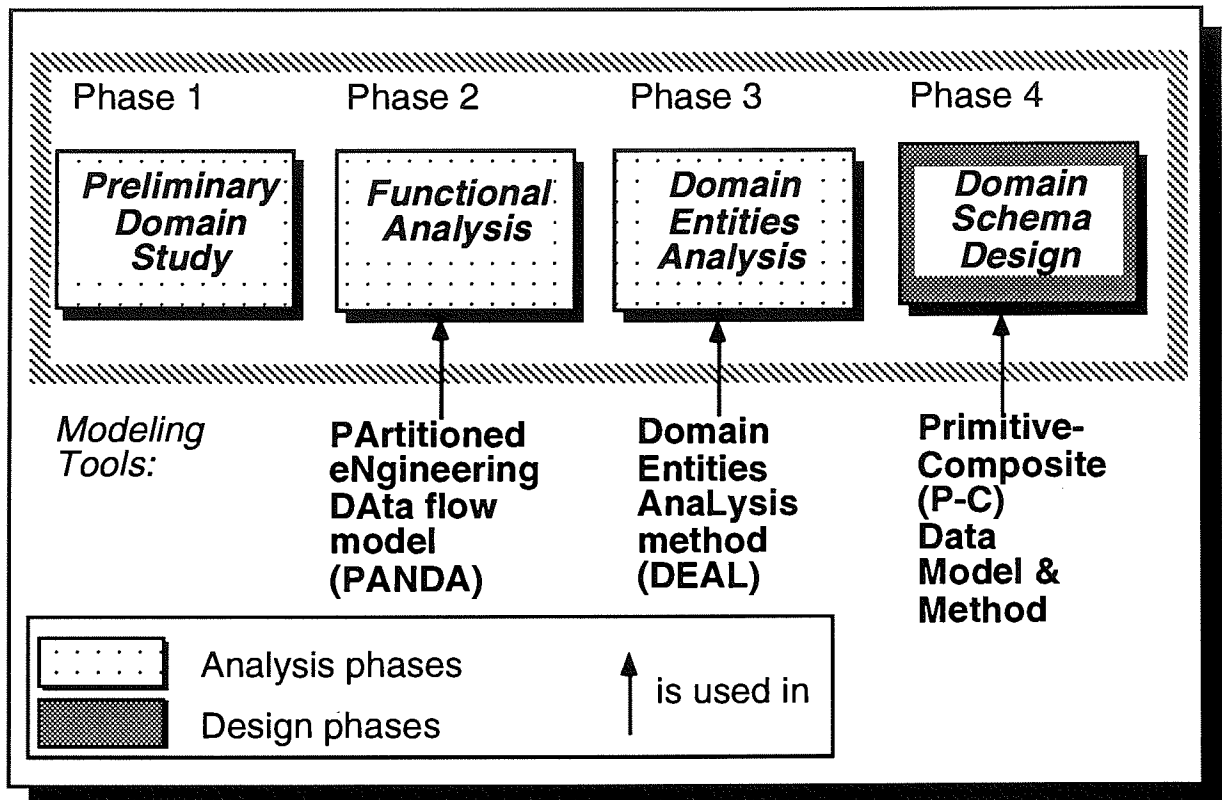
## **3.2 Overview of the P-C Approach**

---

As a methodology, the P-C Approach includes the following:

- Four phases for analyzing the given domain and designing a schema, namely (1) Preliminary Domain Study, (2) Functional Analysis, (3) Domain Entities Analysis and (4) Domain Schema Design. The first three analysis phases lead to the schema design phase. The resulting schema must enable multiple users to share data representations across life-cycle phases in the domain. Specifically, it must support multiple user views and must be extensible.
- The modeling tools used in these phases, which lead to the development of a schema meeting the requirements mentioned above. Specifically, these modeling tools directly incorporate the criteria of cohesion and reusability to design primitive classes of the schema. These modeling tools also provide the elements (i.e., concepts, graphical representations, operations, rules, etc.) necessary for building Computer-Aided Software Engineering (CASE) tools with which a modeler can represent facility data using the P-C Approach.

Figure 3.2 illustrates the overall approach. The following sections explain the requirement and criteria for designing a domain primitive schema and describe the phases and modeling tools of the P-C Approach.



**FIGURE 3.2: Overview of the Primitive-Composite Approach.**

### 3.2.1 Requirements and Criteria

**Sharability as Schema-Level Requirement** In the P-C Approach, sharability is the essential requirement for the development of a domain primitive schema to support data integration. Sharability can be separated into two other component requirements: (sharability) among multiple users and across life-cycle phases. First, the primitive schema must support multiple user views. Using this schema, different users must be able to customize composite classes to represent their own views of complex design objects. Second, the primitive schema must be extensible throughout evolving life-cycle phases. In this case, a modeler must be able to add new primitive classes incrementally to the primitive schema as these phases unfold, without discarding previously defined primitive classes. Moreover, operational measures are defined to test the resulting primitive schema against the sharability requirements. These measures include test variables, measurements, cases and a test procedure and are described in Chapter 8. In short, a domain primitive schema that is designed and tested for this requirement will provide the basis for sharing data representations among multiple users during the various life-cycle phases.

**Cohesion and Reusability as Design Criteria for Primitive Classes** In a facility engineering domain, experts typically design data-intensive objects such as beams, columns, floors, tower panels, etc. A beam, for instance, can be described by its geometry, material, fabrication features, load-resisting functions, bending stresses, etc. Entities representing these design objects are here called “domain entities.” (A domain

entity is a conceptual-level representation of all the data describing a design object. At the next logical level, a composite class represents a view of a design object, which includes only a subset of that data.) The P-C Approach uses cohesion and reusability as the two direct criteria for analyzing those domain entities. The domain entities' analysis leads to the design of a domain primitive schema. In fact, it involves top-down decomposition of the domain entities into basic building blocks called the "primitive entities." The concepts of cohesion and reusability were first introduced in software engineering [Yourdon 79]. Cohesion is defined here as a measurement that indicates how closely the data items of an entity relate to one another. In particular, the P-C Approach identifies five principal dimensions of cohesion: (1) organization/access (how the data is organized and thus how it can be accessed by humans in the work environment), (2) concept (to which concepts the data relates), (3) time (at what time the data is created), (4) source (from which computational sources the data is derived), and (5) use (how the data is used in activities of the engineering process). These dimensions are called "access-cohesion," "concept-cohesion," "time-cohesion," "source-cohesion," and "use-cohesion" respectively. Reusability is another measurement that indicates the extent to which an entity can be reused (i.e., used without modifications) in describing other domain entities. The P-C Approach defines five levels of reusability: reusable (1) for more than one domain entity of a common type, (2) for a single domain, (3) for a single industry, which includes more than one domain, (4) for more than one industry of a common type, and (5) for more than one industry type. These levels are called "domain-entity-type reusability," "domain reusability," "industry (or domain-type) reusability," "industry-type reusability," and "universal reusability" respectively.

The primitive entities identified from the domain entities are conceptual representations that can be implemented at the next logical level using relational tables or object classes. In this approach, these primitive entities are refined and transformed into primitive object classes that constitute the domain primitive schema. Primitive classes are also organized into class hierarchies called "primitive characterization hierarchies." Therefore, *each primitive class is a module of attributes that is "designed" (i.e., first identified, and then refined and transformed) to have maximum cohesion and reusability.*

*In short, while sharability is the requirement of the domain primitive schema, cohesion and reusability are criteria used directly in the design of primitive classes of the schema.* These criteria are directly incorporated into DEAL, a modeling tool provided by the approach. Cohesion and reusability measure the qualities of the resulting primitive classes that ensure sharability of the overall domain primitive schema.

### **3.2.2 Phases and Modeling Tools**

Figure 3.3 summarizes the phases of the P-C Approach, the modeling tools used in these phases, the output of each phase, and the interactions between the phases.

**Phase 1—Preliminary Domain Study** In this phase, a modeler interviews domain experts and collects and organizes relevant information about the domain. Batini et al. [92] discusses this topic in detail and also provides references to background reading material. The P-C Approach assumes that the modeler will consult this wealth of existing work on field study techniques in carrying out Phase 1. By the end of this phase, the modeler must provide the following: general description of the key participants, phases, and computer applications involved in the domain; definition of the terms used in the domain; identification of the objects designed by experts in the domain and the conceptual categories such as geometry, topology, material, fabrication, behavior, etc. of data describing these objects.

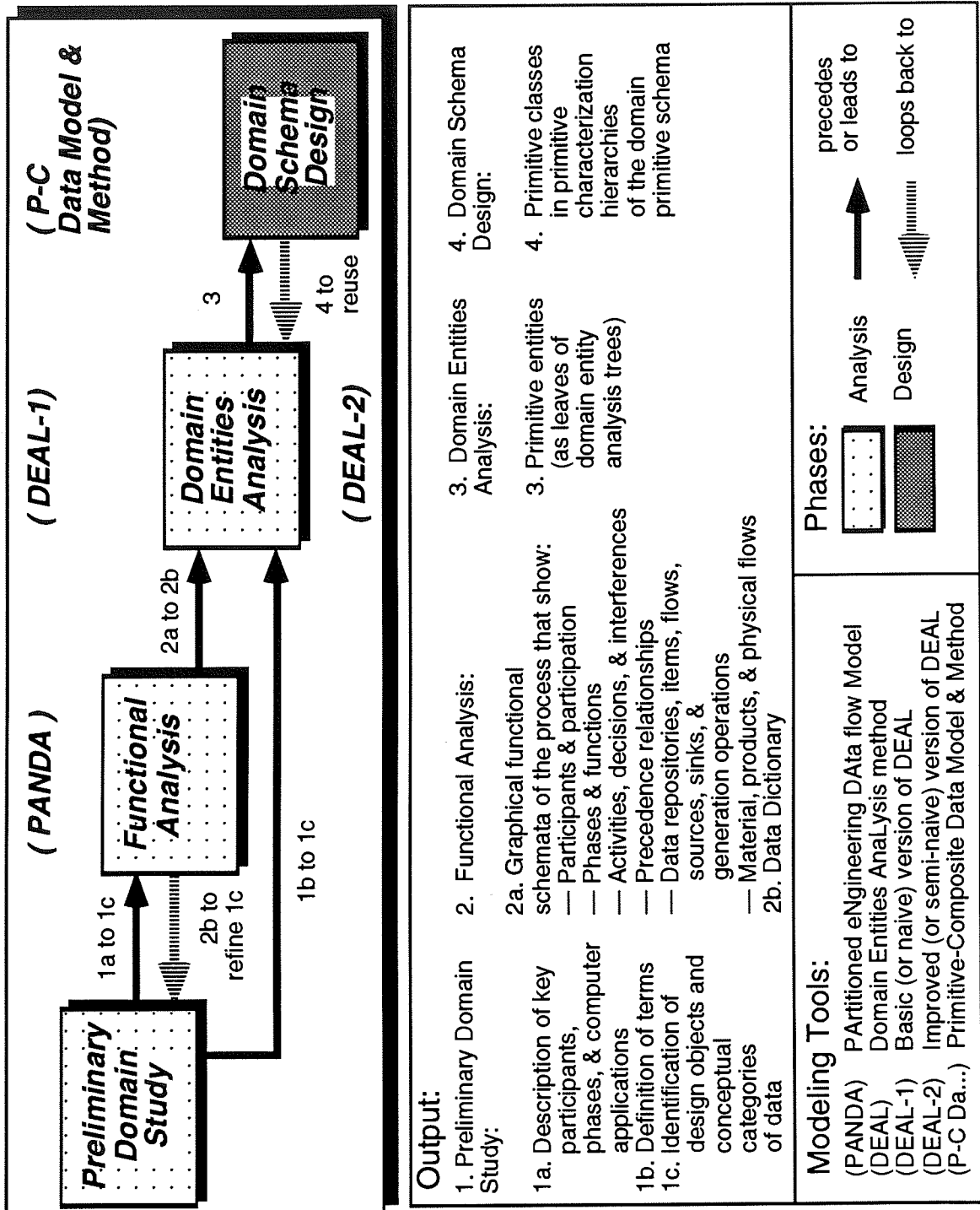


FIGURE 3.3: Phases, Modeling Tools, and Outputs of the P-C Approach.



**Phase 2—Functional Analysis** Having an initial understanding of the domain, the modeler further studies the facility engineering process in the domain and the data used in that process. This is commonly known as “functional analysis.” First, she defines the scope of the functional analysis. The modeler then uses the *PARtitioned eNginEering DAta flow model* (or *PANDA*) to analyze that process. *PANDA* is an extension of the Data Flow model [Gane 79], [Yourdon 79], [De Marco 82], [Batini 92]. *PANDA* provides the concepts needed to represent complicated facility engineering processes. In addition, *PANDA* offers graphical representations, syntactic and semantic rules, schema transformation operations, and a customized method and guidelines for using the model. Moreover, *PANDA* has a partitioned architecture that helps the modeler organize thoughts about complicated engineering processes. At this point, the modeler can refine the conceptual categories of data identified in Phase 1 or continue with Phase 3.

The functional analysis using *PANDA* results in graphical functional schemata of the process that shows the following: the participants and their roles in activities of the process; the decomposition of the process into subprocesses and activities, including the precedence relationships among activities, the design synthesis loops, the decisions and alternatives, and the process interferences; and the data, material and product flow networks. In addition, the modeler builds a data dictionary that includes the definitions of all data items used in the domain. This dictionary must document all existing variations in the naming and representation of data items in the environment. More importantly, this dictionary must also document agreements between the modeler and the domain experts about how those data items will be named and represented in the domain primitive schema. Gane, Sarson [79] and De Marco [79] show how to build a data dictionary based on the Data Flow model.

**Phase 3—Domain Entities Analysis** Once she has acquired a better understanding of the facility engineering process, the modeler can either refine the conceptual categories identified in Phase 1 or proceed to Phase 3. In Phase 3, the modeler uses the *Domain Entities AnaLysis method* (or *DEAL*) to analyze the domain entities, which represent the design objects identified in Phase 1. *DEAL* provides the concepts and terms, graphical representations, rules, procedures, and operations needed for decomposing domain entities into primitive entities using the criteria of cohesion and reusability. The modeler also makes use of the information available from Phase 1 and the output of Phase 2. This analysis using *DEAL* results in the primitive entities that are building blocks of the domain entities.

**Phase 4—Domain Schema Design** In this phase, the modeler designs the domain primitive schema using the *Primitive-Composite* (or *P-C*) *Data Model and Method*. This model provides the concepts of primitive and composite classes and instances, and several relationship types such as generalization, aggregation, and association. The accompanying method includes the steps, rules, and guidelines necessary to design the primitive classes of the primitive schema with the primitive entities from Phase 3 based on the concepts of the model. Chapter 6 explains these steps, rules, and guidelines in detail.

The modeler can iterate between Phases 3 and 4. When analyzing the domain entities for the first time, the modeler must use *DEAL-1*, a basic version of *DEAL*, that uses only cohesion to decompose domain entities. After analyzing one or a few domain entities, the modeler can go to Phase 4 (Domain Schema Design) and design some primitive classes. With those primitive classes in hand, the modeler can go back to Phase 3 to analyze other domain entities. The modeler can now use *DEAL-2*, an improved

version of DEAL that considers both cohesion and reusability. This version recalls and reuses when applicable, the primitive classes available from Phase 4. This modeling option is highly recommended.

As Figure 3.3 shows, feedback loops exist between Phase 1 and Phase 2, and between Phase 3 and Phase 4. However, there are no feedback loops between the tasks of functional analysis and schema design in the P-C Approach since these two tasks are quite involved and need to be handled separately. Future CASE tools automating the approach may enable the modeler to carry out these tasks jointly.

### ***3.3 Evaluation of the P-C Approach***

---

#### ***3.3.1 Scope of Applicability of the Approach***

**Summary of the Characteristics of Applicable Domains** The P-C Approach is a structured methodology that tightly integrates the requirements for sharing a schema within a given domain with the phases and modeling tools that lead to the development of a sharable schema. Because of the number of phases involved and the extent of the work required, this approach should be used mainly for engineering domains that exhibit the following characteristics:

- *The engineering process requires collaboration of multiple participants from different disciplines.* These participants play different roles in the project and, therefore, have different needs and views of the underlying product data.
- *The product life cycle is long, spanning several major phases of development.* Each phase includes a number of functions that, in turn, are divided into several activities.
- *Large amounts of data are generated in an incremental fashion during the life-cycle phases, and this data needs to be communicated among the different participants in all phases.* Moreover, the cooperative use and effective communication of this data among members of the project team is critical to the successful completion of the project.
- *The engineering process involves some level of computer automation.* Computer applications have been used to assist various participants with routine or computation-intensive tasks. These applications need to exchange data.
- *The overall product itself is complicated and consists of several data-intensive design objects.* The data involved delineate various aspects of the design objects, including their physical properties, engineering functions and behavior.

**Further Conditions** In the domains described above, the following conditions must also hold true for the P-C Approach to work properly:

- *The P-C Approach is applied in its entirety (i.e. all four phases are required and all modeling tools are used) to the domain of interest.* This assumption insures that the resulting domain primitive schema can be shared within the domain.
- *In applying this approach, the modeler provides the knowledge about the domain necessary to develop the domain primitive schema.* The P-C Approach provides steps and modeling tools for analyzing the domain and for developing a primitive schema of that domain. However, it lacks the domain knowledge necessary to

generate that schema automatically. In applying this approach, the modeler needs to provide that knowledge. For example, under the successive cohesion criteria considered in the DEAL procedures in Phase 3, the modeler must interactively provide knowledge about the logical access paths, conceptual categories, logical times, computational sources, and primary data uses to which the individual data items relate. The procedures provide the steps, operations and rules that guide the decomposition of entities. In most cases, the modeler has to acquire this knowledge from experts in the domain. The next condition elaborates on this point.

- *Knowledge about the domain can be acquired from the domain experts and can be verified for correctness.* This assumption insures that the modeler can successfully carry out the preliminary domain study (Phase 1) in the P-C Approach. It implies that domain experts must be available for interview during this study and for follow-up consultation in the later phases. It also implies that relevant project documentation (including project design folders, design manuals, engineering drawings, samples of computer applications' input and output files, etc.) must be collected for the purpose of analysis. Specifically, by working with the experts and studying the information collected, the modeler is able to identify the participants, phases, and applications involved in the engineering process in the domain. The modeler is also able to define the terms and domain entities used by the experts and to identify the different categories of data describing the design objects.
- *The engineering process in the domain is correctly modeled in terms of a finite number of related functional units. These units represent functional areas and activities into which the process is decomposed. In addition, the participants involved and data used in each activity can be modeled.* This assumption insures that the modeler can successfully carry out the functional analysis phase in the P-C Approach. This assumption also insures that the resulting schema can represent information used in the engineering process and therefore, can be accepted and used by the domain experts. The resulting model of the process represents the way in which the product is typically planned, designed, and built in the domain. The domain experts can agree upon this model and can verify it. If a single model is not feasible, a fixed number of models representing various scenarios in which the process could occur can be constructed. For example, the tower engineering process can have a scenario in which prototype towers are built and tested before the actual towers are constructed in the field.
- *In the given domain, there exists a fixed number of domain entities. Each domain entity represents a unified view of a design object created by the domain experts and is described by a finite set of unique data items.* This assumption insures that the modeler can complete the domain entities analysis using the P-C Approach. It means that a fixed number of domain entities are analyzed for a given domain. Each entity represents a design object created by experts in the domain. Further, this assumption implies a form of view integration. If two or more domain entities have the same name (e.g., "Transmission Tower Members") but represent different experts' views of the same design object (e.g., "Transmission Tower Members as Designed" and "Transmission Tower Members as Fabricated"), then all the descriptions of those domain entities are integrated into one domain entity description in the analysis. If those entities also have different names (e.g., "Tower Members" and "Transmission Tower Members"), then only one entity name is selected for use (e.g., "Transmission Tower Members"). By the same token, if two or more data items have different names (e.g., "shape size designation" and "shape size identifier") but represent the same property, only one data item is selected for use in the analysis (e.g., "shape size designation"). If two or more data items have

the same name (e.g., “tower member length”) but represent different properties (e.g., “tower member schematic length” and “tower member fabrication length”), they are included in the analysis under distinct names. In fact, the modeler must be able to clearly define the domain entities in Phase 1 of the P-C Approach and to build a data dictionary containing the data items’ definitions in Phase 2.

### **3.3.2 Strengths of the Approach**

The P-C Approach has the following strengths:

- *This approach combines the advantages of object-oriented data modeling with the functionality of primitives and composites in modeling facility data to support data integration.* The object-oriented paradigm offers useful concepts such as classes, attributes, methods, instances, and relationships, as well as powerful abstraction methods such as inheritance, generalization, encapsulation, classification, aggregation, association, etc. These concepts and abstraction methods provide the modeler with convenient tools to represent complex facility data. The notions of primitives and composites enhance this paradigm in the following way: The modeler can define primitive classes to represent the data shared within the domain. Different users can customize composite classes from those primitive classes to define their own views of the shared data.
- *Users sharing a domain primitive schema have the flexibility of defining many complex views of the facility design objects.* Primitive classes defined for a given domain constitute a domain primitive schema. Users sharing this schema are not limited to a fixed number of predefined views of the design objects in the domain. On the other hand, they have the flexibility to define a wide array of complex views of the design objects. Indeed, they can customize a large number of composite classes as combinations of the primitive classes available in the schema. Further, these composite classes need not be defined a priori. Similarly, an application developer can produce highly customized applications from those primitive classes.
- *The modeler can extend a domain primitive schema to accommodate evolving life-cycle phases.* The modeler can do this by adding new primitive classes, from which new composite classes can be defined. As a result, users can define new composite classes as combinations of the primitive classes, both new and old. This extensibility is a strong advantage in supporting design schema evolution and in building large engineering databases.
- *Clean and modular data representations about form, function, and behavior are maintained.* In the P-C Approach, form, function, and behavior are separated into primitive characterization hierarchies. Each hierarchy uses only a single criterion to define the primitive classes and thus provides a clean, homogeneous description of one specific aspect of complex design objects. This enables a modeler to concentrate on representing one aspect of the object at a time and also allows different modelers to work on different aspects simultaneously. This also eliminates the problem of non-homogeneous class hierarchies explained in Chapter 2.
- *Users of a domain primitive schema have complete control over the hierarchical decomposition of a given facility.* The P-C Approach argues that this decomposition should closely reflect the opportunistic way in which facility engineers come up with their designs. In fact, this approach does not impose predefined hierarchical levels (e.g. building, systems, members, and components) by which the user must abide. The user can decompose a facility in two ways: by the functions that it

performs, or by the design artifacts created by the designers. The domain primitive schema provides primitive classes needed to support both types of decomposition.

- *Data modeled following the P-C Approach can be readily exchanged.* First, application developers will no longer have to build special purpose translators. Each application essentially carries the descriptive knowledge needed to support the exchange of common data with other applications. To exchange data between two applications, the data from the first application is transferred into a primitive database that contains only instances of primitive classes, and is then composed into the composite database of the second application. Applications share only the common primitive classes from which their composite classes are assembled. Therefore, in any given domain, a model for data exchange will be the set of primitive classes of that domain, rather than a complex product model that anticipates every possible combination of data in use. Finally, only the data needed by the application is exchanged.

### **3.3.3 Limitations of the Approach**

The P-C Approach has the following limitations:

- *This approach requires that all four phases involved (including three analysis phases and a design phase) are applied to the given domain and that the P-C modeling tools are used in those phases.* Applying all four phases may involve a significant amount of work. However, all these phases are necessary to develop a schema that can be shared by different users throughout the facility life cycle. Each phase needs the results from the previous phases and thus, depends on those phases. In addition, the modeling tools provided by the approach are intended to work together as a set. Although a modeler may use each tool separately, its concepts build on those underlying the tools that were used in the preceding phases. For example, the concept of primitive classes in the P-C Data Model draws heavily upon the concept of primitive entities in DEAL.
- *This approach is currently a manual methodology.* Due to the number of phases involved and the extent of work required, CASE tools are definitely needed to automate the approach and expedite the modeling effort.
- *This approach does not provide yet tools that directly translate data representations between heterogeneous databases and computer applications that already existed in the computing environment.* Unlike the direct data translator approach (explained in Chapter 1 and Chapter 2), this approach is essentially a methodology that can be used to analyze a given domain and to design a common object-oriented schema of that domain. The resulting schema can be used to build new databases and computer applications sharing the same schema.
- *This approach does not offer specific mechanisms for exchanging data between applications.* Like the standard exchange format approach (explained in Chapter 1 and Chapter 2), this approach leads to the development of a domain schema that can also be used as a medium for exchanging data among computer applications. However, specific mechanisms for data exchange between applications using this approach need to be developed. This is recognized to be outside of the scope of the research project but among the future extensions of the project.

- *This approach lacks solutions for the management of data in distributed environments of collaborative facility engineering work.* While research projects such as DICE [Sriram 89] and CEDB [Ullman 91b], [Tiwari 93] are actively seeking new solutions to the facility data management problem, this research project has chosen to focus on the data modeling issues. Without a solution to the modeling of facility data, managing this data is an abstract question. Nevertheless, a logical extension of this research in the future would involve examining issues such as data ownership, integrity, consistency, and concurrency, leading to recommendations of solutions for managing data using this approach in distributed facility engineering work environments.

### **3.4 Chapter Summary**

---

The Primitive-Composite (P-C) Approach is a structured and coherent methodology that can be used to analyze a given facility engineering domain and to design an object-oriented schema of that domain to support data integration. This approach includes the steps, requirements, and criteria that are necessary to design the schema. It also provides the modeling tools used in those steps that lead to the development of the schema meeting the requirements. The following chapters present these modeling tools. Chapter 4 describes the PArtitioned eNginEering DAta flow model (PANDA), Chapter 5 discusses the Domain Entities AnaLysis method (DEAL), and Chapter 6 presents the Primitive-Composite (P-C) Data Model and Method.

The P-C Approach should be used mainly for engineering domains that involve multiple participants, a long product life cycle, computer applications, complicated engineering processes, complex data describing engineering design objects, and a critical need for data communication. This approach offers advantages in modeling complex facility data. The modeler can build a domain primitive schema that includes primitive classes representing the data shared within the domain. Different users have the flexibility to represent their own complex views of the facility design objects. To accommodate evolving life-cycle phases, the modeler can extend the schema by adding new primitive classes, from which new composite classes can be defined. Therefore, the schema flexibility and extensibility help the modeler overcome the deficiencies of existing facility engineering data models that were pointed out in Chapters 1 and 2. Clean and modular data representations about form, function, and behavior are maintained. Moreover, users of the schema have complete control over the hierarchical decomposition of a facility. Data modeled following this approach can be readily exchanged between applications. The P-C Approach also has limitations. Applying all of its four phases may involve a significant amount of work. However, these phases are necessary to develop a schema that can be shared within the domain. CASE tools automating this approach do not exist at the present time. Further, this approach does not provide yet tools that directly translate data representations between heterogeneous databases and computer applications. Currently, it does not include specific mechanisms for data exchange between applications. Finally, it needs solutions for the management of data in distributed environments of collaborative facility engineering work.



# Chapter 4

---

## Functional Analysis Using PANDA

---

### **Chapter Abstract:**

The PARTitioned eEngineering DATA flow model (or PANDA) is an extension of the Data Flow model. It is used for functional analysis of facility engineering processes in the P-C Approach. This chapter first describes the development of PANDA: the requirements we defined, the background study we conducted, and the reasons we selected the Data Flow model. It then provides an overview of PANDA and describes the concepts, graphical representations, syntactic and semantic rules, and schema transformation operations of PANDA. Next, this chapter explains how to apply PANDA to a domain and gives an example from transmission tower engineering. It ends with a summary of and lessons learned from this development work.

### **Organization:**

- 4.1 *Introduction to Functional Analysis in the P-C Approach*
- 4.2 *Development Perspective*
  - 4.2.1 *Required Capabilities and Properties*
  - 4.2.2 *Background Study*
  - 4.2.3 *Evaluation of Selected Models*
- 4.3 *The Extended Model: PANDA*
  - 4.3.1 *Overview of PANDA*
  - 4.3.2 *Key Concepts and Graphical Representations*
  - 4.3.3 *Syntactic and Semantic Rules*
  - 4.3.4 *Schema Transformation Operations*
- 4.4 *Using PANDA*
  - 4.4.1 *Method for Using PANDA*
  - 4.4.2 *Additional Guidelines for Using PANDA*
  - 4.4.3 *Illustrative Example in Transmission Tower Engineering*
- 4.5 *Chapter Summary*



## ***4.1 Introduction to Functional Analysis in the P-C Approach***

---

Functional analysis plays a significant role in developing databases and applications that operate on those databases. Functional analysis is the study of information flow among the activities of a process or processes in an enterprise. Its purpose is to understand what information is used in each activity and how that information is exchanged among the activities [Batini 92]. Functional analysis does not consider the social, economic or environmental impacts of the process, the duration and coordination of activities, the allocation of resources, or the costs or quality of products that result from the process. For this reason, functional analysis clearly differs from the process modeling that has been studied in other fields. For this reason, we specifically use the term “functional analysis” instead of “process modeling.”

Generally speaking, functional analysis is important to any effort that requires an understanding of how an enterprise operates and how information is used to support the enterprise’s operation. In the P-C Approach, functional analysis must be undertaken prior to the modeling of data. By focusing on the activities and information flow of the process, functional analysis helps the modeler understand what information is needed by the activities, how it is used, and who the users are. This understanding is critical to designing a domain primitive schema that will be accepted and shared by all users. The P-C Approach provides a modeling tool, the PArtitioned eNginEering DAta flow model (PANDA), for functional analysis of complex facility engineering processes. Functional analysis using PANDA yields “functional schemata” of the process that are used in the subsequent domain entities analysis. This analysis in turn leads to the identification of the domain’s primitive entities and the design of the domain primitive schema.

In the development of PANDA, my objective was to build a reference model suitable for functional analysis of facility engineering processes. By “reference model,” we mean a set of concepts, rules and operations needed to do the analysis, as well as the method and the guidelines needed to use those concepts, rules and operations. The next section provides a perspective on the development of such a reference model.

## ***4.2 Development Perspective***

---

### ***4.2.1 Required Capabilities and Properties***

**Capabilities** The reference model should have the following abilities:

- *It should be capable of representing the participants and their roles (i.e., the capacities in which they are involved) explicitly in the functional schema.* In a typical facility engineering process, multiple participants from various disciplines are involved in different capacities. The reference model must be capable of representing these participants, which are as important to the process’ description as the activities or the data.
- *It should be capable of representing complex, non-linear facility engineering processes.* A facility engineering process is typically complex. During the facility life-cycle, the engineering process consists of several subprocesses, which in turn include several activities. Moreover, the process is not linear. Subprocesses can take place concurrently since multiple participants are involved. At different times, subprocesses can be activated, suspended, resumed or terminated. In addition, design synthesis involves the typical “Propose-Evaluate-and-Revise” loop. The

more accurately the reference model represents such a process, the more effective the design of the domain primitive schema will be.

- *It should be capable of representing complex data and flow of material and products within the process as well as be able to represent relationships among data as it is generated in the process.* In the facility engineering process, data is actually stored in and retrieved from repositories such as vendors' standard-part catalogs, companies' design manuals and program input and output files. This data can be used in activities, which in turn produce new data. A single activity can use data items from more than one source. Consequently, the flow of data among activities in the process is complex. As the process evolves, the amount and complexity of the data increases over time. New data is generated from existing data drawn from several sources. In addition, real-life processes utilize material and products to build the facility. Thus, the physical flow of material and products is an important part of the process, especially for construction, operations, and maintenance. In short, the reference model must be able to represent complicated data and physical flows that are an essential part of the process. Data generation is also very important: The way in which data is actually generated in the process should directly affect the way in which it is represented in the domain primitive schema.

**Required Properties** The reference model should also have the following properties:

- *It should be more formal than natural languages.* The model should include a finite set of concepts that have clear and precise definitions. Without this property, the modeler might produce process descriptions or specifications that are ambiguous, verbose or inaccurate.
- *It should be graphical.* The model should have graphical representations that can be used to create drawings of the process functional schemata. These drawings provide a pictorial, highly descriptive and concise way to depict the process. They are called "graphical functional schemata." As Ceri [86] points out, the popularity of models such as the Data Flow model attests to the importance of graphical representations.
- *It should be capable of producing highly readable graphical functional schemata.* The model should have built-in features that automatically yield highly readable graphical functional schemata. The schema readability is measured not only by the ease with which users can interpret it ("graphical readability"), but also the extent to which they can comprehend it ("conceptual readability") [Batini 92]. Without this property, the modeler would have the freedom to draw schemata in any way he or she chooses, but could produce graphical functional schemata for complex processes that would be very difficult to read and comprehend.
- *It should be as simple and easy to use as possible.* The model should be designed so that a novice modeler, with minimal learning, could apply its basic features. An experienced modeler would use the more advanced features to be more skillful and efficient. According to Ceri [86], many suggested models have not been used because they are complicated and difficult to learn.

## 4.2.2 Background Study

**General Process Models** We first looked at existing models that could be applied to several domains for all process modeling purposes. We called these “general process models.” A large number of these models have been proposed. Those that are relevant to this study came from many areas of research and development:

- *Software engineering*: This area has contributed the popular Data Flow model [Gane 79], [Yourdon 79], [De Marco 82], [Batini 92] and the Structured Analysis Design Technique (SADT) [Ross 77a], [Ross 77b].
- *Information System Design*: Work done in this area includes the Information Systems Work and Change Analysis (ISAC) approach [Lundeberg 82], the Conceptual Information Analysis Methodology (CIAM) [Gustavsson 82] and the Integrated Computer Aided Manufacturing Definition (IDEF) methodologies [Bravoco 85a], [Bravoco 85b], [Mayer 92], including IDEF0 and IDEF3, to name a few.
- *Database Design in the Field of Databases*: Models developed in this area include the DATAID Database Design Methodology [Ceri 86], Nijssen’s Information Analysis Method (NIAM) [Verheijen 82] and the Active and Passive Component Modelling (ACM/PCM) [Brodie 82] to name a few. In addition, a number of so-called Conceptual Modeling Languages (CML), defined in [Borgida 85], are primarily used for conceptual database design, but also make possible the formal specifications of processes. These specifications can be automatically interpreted and verified. These languages include TAXIS [Mylopoulos 80], Galileo [Albano 85] and Requirements Modeling Language (RML) [Greenspan 86].
- *Theory of Nets*: This theory has resulted in process models such as Petri Nets [Peterson 77], Information Control Nets (ICN) [Ellis 79] and the Information Management Language Inscribed High-Levels Petri Nets (IML) [Richter 82]. These models have a firm mathematical underpinning and are commonly used to model asynchronous process events.
- *“Enterprise Integration” in the Field of Artificial Intelligence (AI)*: More recently, research and development on “Enterprise Integration” in the field of Artificial Intelligence (AI) has focused on acquiring an understanding of how an enterprise operates and how information plays a role in supporting the enterprise’s operation. Work in this area is under way at several universities [Fox 92], [Jagannathan 92], [Srinivasan 92] and industry-funded research centers [Billmers 92], [Bradshaw 92b], [Grosos 92], [Gruber 92] across the country. The general direction here is to explore advanced AI techniques to support enterprise modeling, automation and integration.

In short, the general process models differ vastly in origin, development objectives and emphases, underlying concepts, representational means, method, degree of formalization and software support tools. They also address various analysis and design tasks and phases in the information system life-cycle.

**In Facility Engineering** Less work has been done in facility engineering. What has been proposed can be found in [Sanvido 84], [EPRI 87], [Vanegas 87], [Sanvido 90], [Luiten 91a], and [Luth 91]. Most of what has been done emphasized the accurate depiction of the process itself rather than the development of a reference model for

analyzing and representing the process. The majority of this work consists of functional schemata or verbal descriptions of facility engineering processes and not reference models as we defined them earlier. In other words, no “reference model” has been developed for facility engineering processes.

### **4.2.3 Evaluation of Selected Models**

Based on the background study, we selected five models for evaluation: the Data Flow model<sup>†</sup> [Gane 79], [Yourdon 79], [De Marco 82], [Batini 92], the Structured Analysis Design Technique (SADT) [Ross 77], the Information Systems Work and Change Analysis (ISAC) [Lundeberg 82], Petri Nets [Peterson 77] and Information Control Nets [Ellis 79]. There were three reasons for this decision. First, all five models were suggested in earlier reviews by Ceri [86] and Batini et al. [92]. Second, these five models are representative of the first three categories of general process models discussed above. Finally, we believed them to be the strongest candidates for this study.

We evaluated these models using four criteria drawn from the required capabilities and properties stated earlier. The criteria are listed here in order of importance: (1) supporting the features of the required capabilities, (2) having graphical representations, (3) graphical readability and conceptual readability of functional schemata and (4) simplicity and ease of use. The reader can refer to [Phan 92] for more detailed information about this evaluation. The evaluation showed that all five models have advantages and shortcomings. However, the Data Flow model is the best all-around performer. It provides the basic features of process, data flow, data repository, and data source or sink. It also has graphical representations and is simple to learn and use. Therefore, we selected this model and extended it to meet the requirements stated earlier. The extension, the “PARTitioned eNginEering DATA flow model” (abbreviated PANDA), is presented next.

## **4.3 The Extended Model: PANDA**

---

### **4.3.1 Overview of PANDA**

PANDA has a multi-leveled partitioned architecture. With PANDA, the modeler can functionally decompose a process into several subprocesses, which in turn include many activities. Therefore, a process may have many hierarchical description levels. As Figure 4.1 shows, the data flow diagram at the detailed level has three major partitions:

- *Participants*: This partition presents the people involved in the process in different capacities. These capacities are clearly annotated on the diagram.
- *Process*: This partition presents the process and its subprocesses and activities. Boundary nodes clearly delimit subprocesses and depict their states as “activated,” “suspended,” “resumed” or “terminated.” Decisions and alternatives are part of the process description. Interferences are special occurrences that disrupt the smooth execution of the process. Activities, decisions, interferences and boundaries can have precedence relationships to each other.

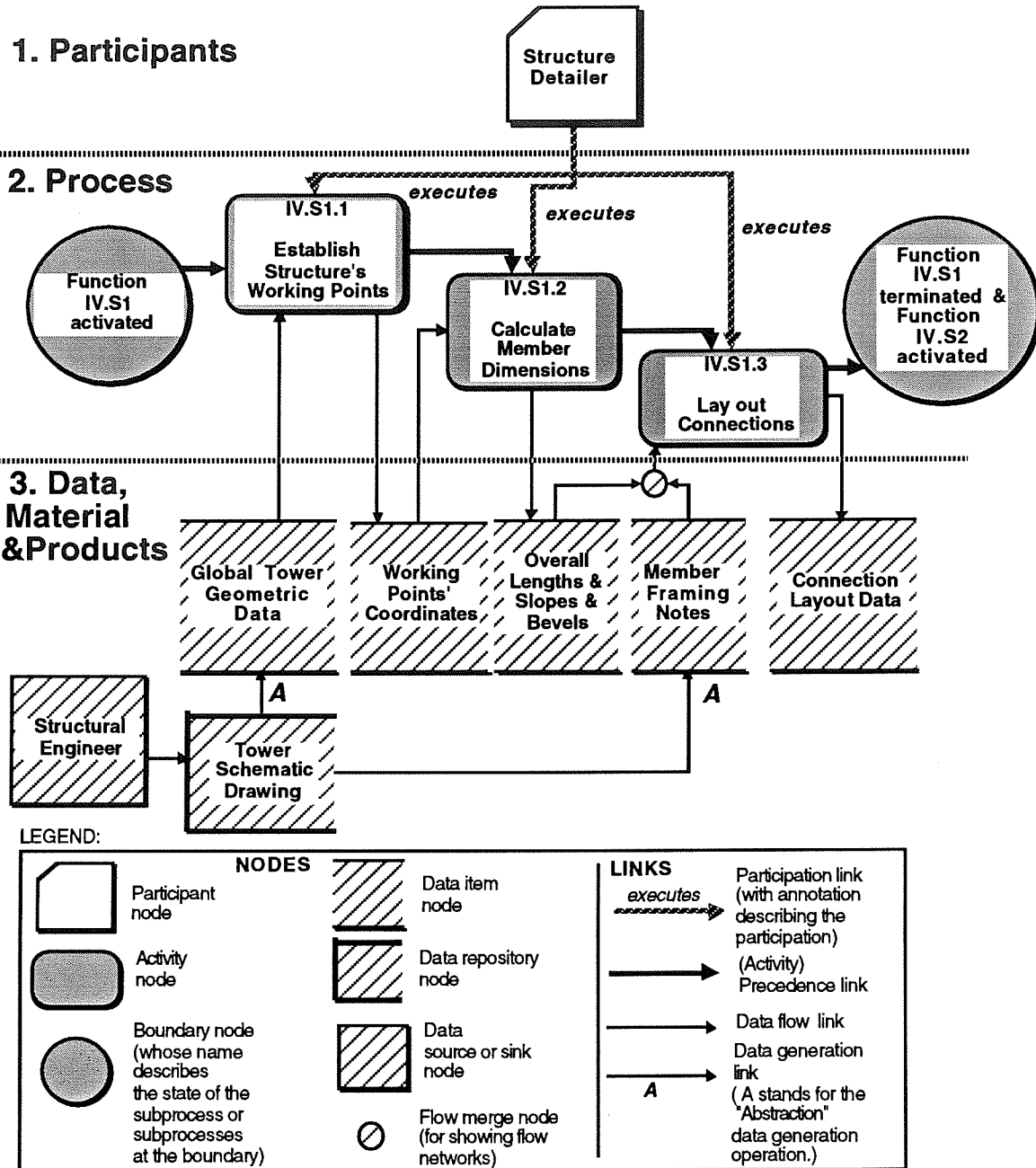
---

<sup>†</sup> The Data Flow model has at least three variations: [Gane 79], [Yourdon 79] and [De Marco 82]. The one considered here comes from [Gane 79] and used by [Batini 92] for conceptual data modeling.

### 1. Participants

### 2. Process

### 3. Data, Material & Products



**FIGURE 4.1: A Sample Partitioned Data Flow Diagram Using PANDA.**

- Data-Material-Products (abbreviated DMP):** This partition presents the data items, data repositories and data flow. It also shows the material and products and their physical flow. The data flow and physical flow are graphically represented as networks of data, material and products circulating in and out of activities. This partition shows the relationships (e.g., is-derivation-of, is-previous-version-of, is-stored-into) among data items and data repositories as they are generated in the process. The partition also presents the people or things that are originators or receivers of data items or data repositories.

This partitioned architecture serves two main purposes. It helps the modeler organize his or her thinking about a complicated engineering process by focusing attention on different aspects of the process during the functional analysis. In addition, the architecture enhances both the conceptual and graphical readability of the process' functional schema. Each partition encloses only the concepts that are relevant to it. With these partitions built into the schema, users can easily review different aspects of the process: the participants; subprocesses and activities; or data, material and products.

PANDA also provides graphical representations of its concepts. As a result, the modeler can draw graphical functional schemata of a process. Due to the three built-in partitions, such a schema is also called a "Partitioned Data Flow diagram" (or "P-diagram"). Figure 4.1 is a sample P-diagram. In this example involving transmission tower engineering, the structure detailer establishes the working points (i.e., the reference points on the tower structure used in the next step to calculate the member dimensions), determines the member lengths, slopes and bevels, and lays out the connections. The input and output data for each activity is shown in the diagram. In addition to the concepts and graphical representations, PANDA provides syntactic and semantic rules that govern the use of its concepts. The modeler can use several basic schema transformation operations in PANDA to develop a functional schema incrementally. A method accompanying the model guides the modeler in applying the concepts to his or her problem domain. Moreover, PANDA provides additional guidelines for using specific concepts of the model and for drawing Partitioned Data Flow diagrams. [Phan 92] provides a detailed description of PANDA.

### 4.3.2 Key Concepts and Graphical Representations

The concepts of PANDA are arranged according to the three major partitions. These concepts are explained next. Tables 4.1 to 4.3 at the end of this section show the graphical representations of these concepts.

**Partition I: Participants** The two concepts in this partition were not included in the original Data Flow model [Batini 92] and are part of my extension. They are:

1. **Participant**—A participant represents a class of personnel that takes part in activities of a process. Each participant can be involved in more than one activity in the process. For instance, the structure detailer, structural engineer and electrical engineer are three different participants in the tower engineering process. The graphical representation of this concept is the "participant node." In Figure 4.1, there is one participant node labeled "Structure Detailer."

2. **Participation**—The concept of participation represents the capacity in which a participant is involved in an activity of a process. There are two possible pre-defined roles for the participant: carrying out an activity and being directly responsible for its successful completion (executive role), or being involved in other indirect capacities and having no direct responsibility or authority (supporting role). In the example shown in Figure 4.1, the structural detailer carries out three activities and plays the executive role in each case. Each participation is represented graphically by a "participation link" from a participant node to an activity node in the next partition. The link is annotated with a clear description of the capacity in which the participant is involved.

**Partition II: Process** Only the first concept in this partition was included in the Data Flow model. The others are part of my extension. The concepts here are:

1. **Activity**—An activity is defined as “an organizational unit of a process for performing a specific task” [Webster 86]. Figure 4.1 shows three different activities. A process actually includes many activities. The graphical representation of an activity is the “activity node.”

2. **Precedence Relationship**—Activities have precedence relationships. These relationships place temporal constraints on the execution of the activities. *An activity, A, precedes (or has precedence over) another activity, B, if B cannot be started until A is finished.* The graphical representation is the directed “precedence link.” Each link connects a pair of activity nodes. Its arrow goes from the predecessor activity to the successor activity. For example, as Figure 4.1 shows, the activity of establishing the structure’s working points precedes that of calculating member dimensions.

3. **Decision and Alternative**—A decision is a special activity that involves answering an important question by considering one or more alternatives and choosing among them. Each “alternative” leads to a unique solution. A decision is graphically represented as a “decision node.” The alternatives are represented as annotations to the precedence links coming out of the decision nodes. Each link points to an activity node representing the action that needs to be carried out when choosing that alternative. A decision node can be connected to an activity node by precedence links.

4. **Interference**—An interference is a special occurrence that interrupts the successful execution of the process. An interference can lead to activities required to remedy the situation. It can also put the current process into a “suspended” or “terminated” state. It can create a loop that takes the process back to some earlier activity. An interference is graphically represented as an “interference node.” Interference nodes can be connected to activity nodes and even decision nodes by precedence links.

5. **Subprocess**—A subprocess is a fixed set of activities, decisions, interferences and delimiting boundaries, which have precedence relationships to each other. A subprocess can include other child subprocesses. Thus, the definition of a subprocess covers the intermediate functional units of a process as well as the process itself. A process is the overall set of functional units that has no parent. A subprocess is graphically represented by its components: activities, decisions, interferences and boundaries. The subprocess in Figure 4.1 represents the subprocess of dimensioning members and laying out connections in the tower engineering process.

6. **Boundary**—A boundary marks the beginning or end of a subprocess or the borderline between a subprocess and another activity or subprocess, as shown in Figure 4.1. Each subprocess has at least two boundaries. Boundaries actually serve three purposes: (1) they delimit subprocesses; (2) they connect subprocesses; and (2) they allow subprocesses to be mapped back to their parent process. Boundaries are graphically represented as “boundary nodes.” The name of the boundary node describes the state of the subprocess or subprocesses at the boundary. The possible states are: “activated” (or “started”), “suspended,” “resumed” and “terminated” (or “ended”).

7. **Process Non-Linearity**—Each node in this partition can have more than one outgoing precedence link with other nodes. This allows the modeler to represent parallel activities or subprocesses of non-linear processes. Moreover, by naming its boundary nodes, the modeler can designate a subprocess as “activated,” “suspended,” “resumed” or “terminated.” Subprocesses that are suspended during the execution of other subprocesses can be represented. As a result, multiple concurrent subprocesses and activities conducted by different participants can be delineated. *Reciprocally, any node can have more than one incoming precedence link from other nodes.* This allows the modeler to represent loops and thus, iterative subprocesses such as the common “Propose-Evaluate-and-Revise” loop in design synthesis.

**Partition III: Data-Material-Products (DMP)** This partition includes many concepts. Of these, only data repository, data flow, and data source or sink were included in the original Data Flow model [Batini 92]. The others are parts of my extension. The concepts are:

1. **Data Repository**—A data repository is a permanent storage of data in paper or electronic format. Examples include files, permanent records, paper or electronic forms, electronic databases, vendors' standard parts catalogs, standard design codes, companies' design manuals, engineering drawings, and program input and output printouts. The corresponding graphical representation is the "data repository node." In Figure 4.1, the "Tower Schematic Drawing" node represents a data repository.

2. **Data Item**—A data item represents a single piece of data or a collection of data that is created by activities of the process and used as input by other activities. A data item may or may not later be stored into a data repository. This concept was not included in the original Data Flow model. In that model, data items can be shown only as annotations (of data flow links) in the data flow diagram. By contrast, data items are represented explicitly in PANDA and have their own graphical representation, the "data item node." Figure 4.1 shows several data items such as global tower geometry data, working points' coordinates, and connection layout data.

3. **Data Flow**—The concept of data flow indicates that a data item or data repository flows into or out of an activity. Data flow is graphically represented by the directed "data flow link" between a data item node or data repository node and an activity node. The direction of the arrow indicates whether the data item or data repository serves as input to or output from the activity. In Figure 4.1, data flow links show how data is used in and exchanged between the three consecutive activities of dimensioning members and laying out connections.

4. **Data Source or Sink**—A data source or sink represents the person or thing that is the prime originator or receiver of data repositories or data items. The graphical representation is the "data source or sink node." In Figure 4.1, the data source is the structural engineer who provides the tower schematic drawing. Data source or sink nodes can be connected to data source nodes or data item nodes by data flow links.

5. **Data Generation**—During the life-cycle phases of the facility, new data is generated from existing data from several sources. The concept of data generation captures the special relationships that exist between data items or repositories as the process unfolds. This concept enables the modeler to show better how data is actually generated and is evolved in the process. Data generation is graphically represented as directed "data generation links." These are special links that exist only among data items and data repositories. The direction of the arrow goes from the source data to the resulting data. The link is also annotated with an abbreviation showing the type of data generation relationship involved. The six main types of data generation relationships are:

- "is-abstracted-to" (Type A): A relationship of this type indicates that a source data item or repository is used to abstract a certain data item or repository of interest (and suppress others). This usually involves reducing the volume of the source data. In Figure 4.1, the global tower geometry data and member framing notes are abstracted from the tower schematic drawing.
- "is-derivation-of" (Type D): A relationship of this type indicates that an existing data item or repository is used to derive or compute a new data item or repository.
- "is-previous-version-of" (Type V): A relationship of this type indicates that an existing data item or repository is modified to create a new version.



- “is-stored-into” (Type S): A relationship of this type indicates that an existing data item is placed into permanent storage (i.e., into a data repository).
- “is-combined-into” (Type C): A relationship of this type indicates that an existing data item or repository is put together with other data items or repositories to form a new data item or repository.
- “is-presented-in” (Type P): A relationship of this type indicates that an existing data item or repository is presented under a different format in another data item or repository.

**6. Material or Product**—The concept of material or product represents the resources, as well as intermediate or final results, of the process. Its graphical representation is the “physical node.”



**7. Physical Flow**—The concept of physical flow here indicates that a material or product flows into or out of an activity. Physical flow is graphically represented by the directed “physical flow link” between a physical node and an activity node. The direction of the arrow indicates whether the material or product is a resource for or result of the activity. For example, raw steel material is a resource used in the fabrication of the tower parts, the result of which are the galvanized tower parts.

**8. Mixed Flow**—In the facility construction stage, an activity can use not only data, but also material and products. For example, the fabrication of the tower parts needs raw steel material and the data describing the fabrication features from the tower detailed drawing. The concept of mixed flow indicates that a data item or repository and a material or product flow together into or out of an activity. The graphical representation is the directed “mixed flow link.” (The terms “physical flow” and “mixed flow” were introduced in the ISAC model [Lundeberg 82].)

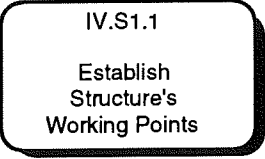

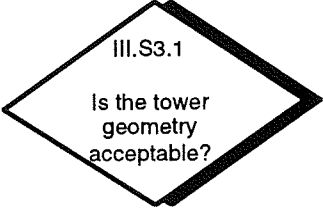
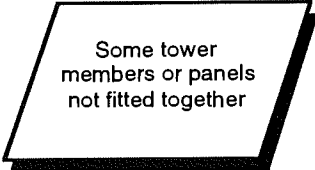
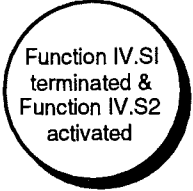
**9. Flow Network**—The concept of a flow network represents an aggregated way of showing complex data flow among activities: Several different data items and data repositories can be used in a single activity in order to generate more data. This concept applies to physical and mixed flow. A flow network can be shown graphically using “flow merge nodes.” Each node represents a point where the data flow, physical flow, or mixed flow among activities merge. The flow merge node in Figure 4.1 shows the combination of two data items that are used in the third activity.

The following tables 4.1 to 4.3 summarize the concepts and their graphical representations.

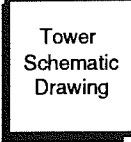


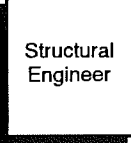





**TABLE 4.1: Concepts and Graphical Representations in Partition I.**

CONCEPTS	GRAPHICAL REPRESENTATIONS
Participant	Participant Node 
Participation	Participation Link 

**TABLE 4.2: Concepts and Graphical Representations in Partition II.**

<b>CONCEPTS</b>	<b>GRAPHICAL REPRESENTATIONS</b>
<i>Activity</i>	Activity Node 
<i>Precedence Relationships</i>	Precedence Link 
<i>Decision</i>	Decision Node 
<i>Interference</i>	Interference Node 
<i>Subprocess</i>	The graphical representation of a subprocess includes those of the activities, decisions, interferences and boundaries that make up the subprocess.
<i>Boundary</i>	Boundary Node 
<i>Process Non-linearity</i>	<ul style="list-style-type: none"> <li>• Any of the nodes in Partition II can have more than one outgoing or incoming links.</li> <li>• By naming its boundary nodes, a subprocess can be designated as “activated” (or “started”), “suspended,” “resumed,” and “terminated” (or “ended”).</li> </ul>

**TABLE 4.3: Concepts and Graphical Representations in Partition III.**

<b>CONCEPTS</b>	<b>GRAPHICAL REPRESENTATIONS</b>	
<i>Data Repository</i>	Data Repository Node	
<i>Data Item</i>	Data Item Node	
<i>Data Flow</i>	Directed Data Flow Link	
<i>Data Source or Sink</i>	Data Source or Sink Node	
<i>Data Generation</i>	Directed, Annotated Data Generation Link	 <p>(Annotation is the abbreviation of the type of data generation operation involved.)</p>
<i>Material or Product</i>	Physical Node	
<i>Physical Flow</i>	Directed Physical Flow Link	
<i>Mixed Flow</i>	Directed Mixed Flow Link	
<i>Flow Network</i>	Flow Merge Node	

### 4.3.3 Syntactic and Semantic Rules

PANDA provides a number of syntactic and semantic rules. The syntactic rules insure that the nodes are connected properly, that the links are used in the right places, and consequently, that the resulting schema conforms to my definition of the model. These rules form the underlying grammar of the model. However, a syntactically valid schema may represent a process that would not make sense in real life. Thus, the semantic rules further constrain the use of the elements so that they produce meaningful schemata.

**Syntactic Rules** The matrix shown in Table 4.4 summarizes the syntactic rules for nodes and links. The column headings across the table designate the types of nodes from which a directed link can originate. The row headings on the left hand side of the table designate the types of nodes to which a directed link can go. A filled cell in the matrix indicates that a node of type designated by the column heading can be linked to a node of the type designated by the row heading. The types of link that may be used are designated by the cell entry. An empty cell indicates that no linkage is permissible. Due to space constraints, we use the following abbreviations: Repository for data repository, Item for data item, participation for a participation link, precedence for a precedence link, data flow for a data flow link, generation for a data generation link, physical for a physical flow link, and flophys/mix for a data flow or physical flow or mixed flow link. All of these links are directed links.

**Semantic Rules** The first two rules given below apply to all nodes in Partition II. In these rules, the so-called “process event” refers to an activity, decision, interference, or a boundary node that marks the initiation of a subprocess. The semantic rules are:

- **Conjunctive Precedence Rule**—*A process event cannot occur unless all its precedent events are completed.* This rule elaborates the precedence relationships for those nodes that have more than one incoming precedence link.
- **Disjunctive Precedence Rule**—*If the node representing a process event is marked with the special symbol “+” enclosed in a circle, then the event can occur as soon as one of its precedent events has been completed.* This rule presents an exception to the conjunctive precedence rule. For example, an activity can have two incoming precedence links, one of which loops back from an activity that happens downstream. In this case, the activity can begin as soon as one of its precedent events has been finished.
- **Bounded Subprocess Rule**—*Each subprocess must contain at least one activity and must be delimited by at least two distinct boundary nodes. One of the boundary nodes must have an activated state and another must have a terminated state. This rule insures that each subprocess is defined properly and has a beginning and an end. As a result, the overall process always has a beginning and an end.*
- **State Closure Rule**—*A subprocess that enters an “activated” or “resumed” state must eventually reach a “terminated” or “suspended” state; similarly, a subprocess that enters a “suspended” state must eventually reach a “resumed” or “terminated” state.* This rule extends the previous rule by specifying the possible states that can occur in between the beginning and the end of a subprocess. It also insures that no subprocess has open ends in the schema.

Partition	I				II								
	Participant	Activity	Decision	Interference	Boundary	Repository	Item	Flow Merge	Physical	Source sink			
Participant													
Activity	participation	precedence	precedence	precedence	precedence	data	data	flophy/imix	physical				
Decision	participation	precedence	precedence	precedence	precedence	data	data	flophy/imix	physical				
Interference	participation	precedence	precedence	precedence	precedence								
Boundary		precedence	precedence	precedence	precedence								
Repository		data flow	data flow			generation	generation			data flow			
Item		data flow	data flow			generation	generation			data flow			
Flow Merge		flophy/imix	flophy/imix			data flow	data flow	flophy/imix	physical				
Physical		physical	physical										
Source sink						data	data						

**TABLE 4.4: Matrix of Permissible Node Linkages in PANDA Using the Appropriate Link Types.**

#### 4.3.4 Schema Transformation Operations

PANDA provides several basic operations to support incremental transformations of a functional schema. (They are actually “types” of operations, for the purpose of generality and conciseness. A type can have many variations.) The definition of these operations closely follows the syntactic and semantic rules stated earlier. Therefore, the modeler can produce valid and meaningful functional schemata using these operations. These operations are incorporated into the method accompanying the model, which is presented next. They are labeled for future reference. The label consists of a roman numeral such as I, II, III indicating the applicable partition and an abbreviation such as C for Creation (i.e., creating new nodes) and M for Modification (i.e., modifying the way in which the nodes are connected). The abbreviation is followed by an integer indicating the ordering of the operation. The basic schema transformation operations are:

- I-C1**      **Creating a Participant**—This operation creates a participant node in Partition I. It draws the new node and links it to an existing activity, decision or interference node in Partition II using a participation link. The link’s annotation describes the participation (i.e., capacity in which the participant is involved).
- I-M1**      **Adding a Participation Link**—This operation adds a participation link connecting an existing participant node in Partition I to another activity, decision or interference node in Partition II.
- I-M2**      **Modifying a Participant**—This operation changes the name or the participation’s description of an existing participant node in Partition I, or reconnects that node to a different activity, decision or interference node in Partition II using a new participation link. This operation has three variations: modifying name, modifying participation, and modifying an existing node connectivity.
- II-C1**      **Creating a Subprocess**—This operation creates a subprocess in Partition II. It draws one activating boundary node, one activity node, and one terminating boundary node and connects them in that order using precedence links.
- II-C2**      **Creating a Functional Unit (i.e., activity, decision, interference, or a child subprocess) in an Existing Subprocess**—This operation creates a functional unit and adds it to a subprocess already defined. The unit can be an activity, decision, interference, or even a child subprocess. The corresponding new node (or nodes in the case of a subprocess) is drawn and then connected to one existing node in Partition II. The connection is done using precedence links and following the appropriate syntactic rules for activity, decision, interference and boundary nodes.
- II-M1**      **Doing Functional Decomposition**—This operation spawns an existing activity node at one level of functional decomposition, say Level A, into a child subprocess at the next lower level, Level B. The subprocess at Level B is created in the same way as the one described in Rule II-C1. The decomposed activity at Level A is also changed to a subprocess by adding boundary nodes that delimit the subprocess. (Both subprocesses have boundary nodes with identical names for later reconnection.)
- II-M2**      **Modifying a Functional Unit**—This operation changes the name of a node (i.e., activity, decision, interference or boundary) in Partition II or reconnects that node to a different node using a new link. The appropriate syntactic rules for nodes in Partition II and links apply. This operation has

two main variations: modifying a name and modifying an existing node connectivity.

- III-C1** *Creating a Data Repository or Data Item*—This operation creates a data repository or data item node in Partition III. It draws the new node and connects it to one activity or decision node in Partition II or to a flow merge node in Partition III using a data flow link.
- III-M1** *Adding a Data Flow Link from a Data Item or Repository to a Process Node*—This operation adds a data flow link connecting an existing data item or data repository node in Partition III to an activity or decision node in Partition II or to another flow merge node in Partition III.
- III-M2:** *Adding a Data Generation Link*—This operation adds a data generation link connecting an existing data item or data repository node in Partition III to another data item or data repository node. The operation has two variations: generating a data repository or a data item.
- III-C3** *Creating a Data Source or Sink*—This operation creates a data source or sink node in Partition III. It draws the new node and connects it to one data item or data repository node in Partition III using a data flow link.
- III-M3:** *Adding a Data Flow Link from a Data Source or Sink to a Data Item or Repository*—This operation adds a data flow link connecting an existing data source or sink node in Partition III to another data item or data repository node.
- III-C4** *Creating a Material or Product*—This operation creates a physical node in Partition III. It draws the new node and connects it to one activity or decision node in Partition II or to one flow merge node in Partition III using a physical flow link.
- III-M4:** *Adding a Physical Flow Link*—This operation adds a physical flow link connecting an existing physical node in Partition III to an activity or decision node in Partition II or to another flow merge node.
- III-C5** *Creating a New Flow Network*—This operation creates a new network from several existing flows that have a common destination. It has three variations: creating a data flow network, a physical flow network, or a mixed flow network. By adding a flow merge node and a data flow link, the first variation combines flows into a network from data item or data repository nodes. The new flow merge node is connected to the data item or repository nodes. The new data flow link goes from the flow merge node to the destination node. Similarly, the second variation combines flows into a network from physical nodes by adding a flow merge node and a physical flow link. The third variation combines several flows, some from data item or data repository nodes and others from physical nodes, by adding a flow merge node and a mixed flow link.
- III-M5** *Merging with an Existing Flow Network*—This operation merges an existing flow or flow network with another existing flow network. It differs from the above operation in that no new flow network is created here. This operation has three variations. In the first variation, a data flow from a data item or data repository node is merged with a flow merge node by connecting the first node to the latter node. The second variation is similar to the first, except that it involves a physical node and a flow merge node. The third variation is again identical, except that it involves two flow merge nodes. These variations can be used over and over again

to build elaborate flow networks. In all three cases, the type of link coming out of the flow merge node must be checked using the syntactic rules for links.

**III-M6** *Modifying a Node in Partition III*—This operation changes the name of a node (i.e., data item, data repository, data source or sink, or flow merge node) in Partition III or reconnects that node to a different node using a new link. The appropriate syntactic rules for nodes in Partition III and for links apply. This operation has two variations: modifying a name and modifying an existing node connectivity.

---

## 4.4 Using PANDA

---

### 4.4.1 Method for Using PANDA

**Scope Definition Before Functional Analysis** Before functional analysis of the process begins, the scope of the analysis must be defined. This is crucial to ensure satisfactory results. Defining the scope involves:

- *Stating the breadth of the analysis effort:* Making general statements about which disciplines, participants, phases, data and data sources and sinks are to be included in and excluded from the analysis.
- *Stating the depth of the analysis effort:* Specifying as much as possible the extent to which the modeler should carry out the analysis of the above categories. As a result, the modeler can put different emphases on different parts of the analysis.

The resulting specifications will guide the modeler throughout the entire effort.

**Mixed Two-Pass Method** This method guides the modeler in applying PANDA to different facility engineering processes. The method uses a mixed two-pass strategy, which involves top-down functional decomposition and bottom-up functional refinement in two successive passes. This method also takes advantage of PANDA's partitioned architecture by giving different priorities to the partitions at different stages of analysis. The resulting set of functional schemata describes the engineering process in several hierarchical levels. The top-level functional schema shows the overall process' decomposition into phases. This schema is also called a "skeletal functional schema." A phase is a subprocess of the overall process, which corresponds to a major identifiable stage of development in the facility life-cycle. The functional schemata at the next level depict the phases' breakdown into functions. Each function is a group of coherent activities that together help achieve a distinct objective or short-term purpose. The low-level functional schema shows the activities of the functions. The method is as follows:

**PASS 1: TOP-DOWN FUNCTIONAL DECOMPOSITION WITH PROCESS-PARTICIPANTS-DMP PRIORITY** The first pass involves top-down functional decomposition of the process, with the following order of priority: (1) process, (2) participants, and (3) data, material and products. It includes the following steps:

- 1.1 *Produce the top-level skeletal functional schemata of the phases and functions of the facility engineering process:* First, identify the major phases of the process and then the functions of which they are constituted. (Guidelines for doing this are given in the next section.) Next, using the graphical representations in PANDA, produce



the “skeletal functional schemata” for those phases and functions. At this top level, represent the phases using the concept of subprocess. Model functions as activities of the subprocesses representing the phases. Use mainly the schema transformation operations II-C1 and II-C2 to create subprocesses and add functional units (i.e., an activity, decision, interference, or even child subprocess) to those subprocesses.

- 1.2 *Perform top-down functional decomposition using the skeletal functional schemata with highest priority on Partition II (Process):* Using the above skeletal functional schemata, decompose the functions into their component activities. (The modeler has complete control over the total number of functional decomposition levels.) Use the schema transformation operations II-C1 and II-C2 to create subprocesses and to add functional units, and especially operation II-M1 for functional decomposition. Because of the potentially overwhelming complexity of the overall engineering process, concentrate at this point mainly on the process. Identify the key participants when possible. The modeler does not have to be very specific about or thorough with the elements of Partition III. The order of priority here is Partition II, Partition I and Partition III.
- 1.3 *Stop functional decomposition at the level of activities and augment the other partitions:* Stop functional decomposition at the level of individual activities. (A rule of thumb for doing this is given in the next section.) Augment Partition I (Participants) and Partition III (Data-Material-Products) as much as possible by specifying the details to be included in those partitions. For instance, be more specific about the participants and their participation, the input and output data of the activities, the data repositories used, the potential flow networks, the data generation operation performed, and the data sources and sinks needed. Use mainly the schema transformation operations I-C1 and III-C1 to C5.

In this pass, the modeler does not have to complete Partitions I and III.

**PASS 2: BOTTOM-UP FUNCTIONAL REFINEMENT WITH DMP-PARTICIPANTS-PROCESS PRIORITY** The second pass involves bottom-up functional refinement of the process, with the priority placed on: (1) data, material and products, (2) process and (3) participants in that order. It includes the following steps:

- 2.1 *Revise low-level functional schemata and complete Partition III (Data-Material-Products) and Partition I (Participants):* Revise each functional schema by first reviewing Partition III, making any necessary changes, and then completing this partition. Then, using (the nodes and links shown) in Partition III, revise Partition II by reviewing and modifying it if necessary. Go back and forth between these two partitions until no more changes are needed. Using Partition II, review, modify and complete Partition I. Use the schema transformation operations II-C1 to C2, I-C1, and III-C1 to C5 to add new nodes to the three partitions, and especially I-M1 and III-M1 to M5 to modify Partitions I and III.
- 2.2 *Take an inventory of the important data sources and sinks, data repositories and data items involved in the process and represented in Partition III, and trace them:* First, step back and take an inventory of all key elements that are involved in the process. Those elements include data sources and sinks, data repositories, and data items that are represented in Partition III. Then, see where those elements are represented in the functional schemata produced from the previous pass. If they have not already been included, revise the appropriate functional schemata to include them in Partition III. This may involve revising Partitions II and I of those functional schemata.

- 2.3 *Take an inventory of the participants involved in the process and trace them:* First, take an inventory of all participants involved in the process. Then, see where those participants are represented in Partition I of the functional schemata. If they have not already been included, revise the appropriate functional schemata to include them in Partition I. This may involve revising Partitions II and III of those functional schemata.
- 2.4 *Revise the higher level functional schemata using a bottom-up refinement approach:* Use the functional schemata modified in the previous step to revise the functional schemata of higher levels, including those skeletal schemata produced in Step 1.1. The objective here is to integrate the low-level functional schemata with higher level schemata and produce consistent functional schemata at all levels.

All three partitions should be completed by the end of this second pass. The modeler must then build a data dictionary that includes the definitions of all data items used in the domain and considered in this functional analysis. This dictionary must document all existing variations in the naming and representation of data items in the environment, and more importantly, agreements between the modeler and the domain experts about how those data items will be named and represented in the domain primitive schema. Gane [79] and De Marco [79] show how to build a data dictionary based on the Data Flow model.

#### **4.4.2 Additional Guidelines for Using PANDA**

**Guidelines for Identifying Phases of a Facility Engineering Process** A phase is a subprocess that corresponds to a major identifiable stage of development in the facility life cycle. Use the following guidelines to identify the phases of a process:

- *As a short cut, consult a domain expert.* A domain expert can usually identify a phase in terms of the following: time, people involved, place, work involved, goal, important decisions made and end results.
- *Otherwise, look for major milestones in the process where a transfer of responsibilities and important deliverables between participants from two major disciplines takes place.* The phases can be separated at these turning points.
- *Consider the informational differences between various periods in the process.* The informational differences between the phases normally lie in the amount and granularity of detail of the facility description. For example, the structural conceptual design phase produces the global geometric data of a structure and the general geometric data of the systems and members. By contrast, the structural detailed design phase adds more detailed design data, including member sizes, cross-sectional properties, stresses and deflections, and connection data.
- *Finally, consult existing work on standard process description for different facility types.* This work includes: [Vanegas 87] for the early phases of general building design, [Sanvido 84] for the later construction process, [EPRI 87] for electrical power plants, [Sanvido 91] for buildings in general, [Luth 91] for high-rise commercial office buildings, and [Phan 92] for electrical utility transmission towers.

**Guidelines for Identifying Functions of a Phase** A function is a group of coherent activities that together help achieve a distinct objective or short-term purpose. Figure 4.4 in the next section shows the functions of the Tower Construction Planning phase. A phase can be decomposed into several functions using the following guidelines:

- *Identify and define functions as groups of coherent activities that together help achieve a distinct objective or short-term purpose.*
- *Divide the major functions of the phase according to the discipline involved. A function is normally carried out by experts of one discipline.*
- *Examine the project control hierarchy.* Luth [91] suggests that divisions of a process are subprocesses that interact, communicate and also impose constraints on one another. A project control hierarchy is needed to anticipate and coordinate these subprocesses. Such a hierarchy exists at three levels: global, regional and local. At the global level, Luth [91] classifies functions into three general categories: owner, design and construction. The breakdown in the control structure at the next lower levels may help identify the functions of the process.
- *Finally, keep each function small—no larger than the single sheet on which the schema is drawn.* Larger functions that span several sheets of paper drawing need to be re-examined and possibly decomposed further.

**Guidelines for Identifying Activities of A Function** A function in turn can be decomposed into several activities. Figure 4.5 in the next section shows the activities of the first function of the Tower Construction Planning phase. As rules of thumb, stop the functional decomposition when one of the following becomes true:

- *Further decomposition would require specific domain knowledge about the design.*
- *Further decomposition would not reveal new information of interest about the participants involved or data used.*
- *A single software module can be built to handle the activity.*

**Other Guidelines** Examples of these guidelines are:

- *No annotation is needed for the executive role.* By default, a link from a participant node to an activity node without annotation means that the participant is carrying out the activity. However, for the supporting role, the link must be annotated with a clear description of the capacity in which the participant is involved. In addition, *any link between a participant node in Partition I and an activity node in Partition II always assumes a downward direction.* Both of these defaulting guidelines reduce the amount of work a modeler has to do and improve the graphical readability of the resulting P-diagram.
- *The modeler can duplicate participant nodes in the P-diagram to minimize the number of link crossings and improve the graphical readability of the diagram.* Duplicated nodes should be clearly denoted using the standardized graphical notations that come from Gane's version of the Data Flow model [Gane 79].

Phan and Howard [92] describe additional guidelines of PANDA such as those for using the concepts of decision, boundary, data item, and flow network; for drawing the partitioned data flow diagrams; and for validating the resulting functional schemata.

#### **4.4.3 Illustrative Example in Transmission Tower Engineering**

In this section, we give an example of applying PANDA to the domain of transmission tower engineering. Transmission towers are large lattice structures supporting wires that transmit electrical power. As Figure 1.2 in Chapter 1 illustrates, the tower engineering process can be decomposed into six phases: (I) Transmission Line Analysis and Design, (II) Tower Structural Conceptual Design, (III) Tower Structural Detailed Design, (IV) Tower Construction Planning, (V) Tower Construction Execution and (VI) Tower Facility Management. In this example, we focus on the Tower Construction Planning phase. We first describe the phase in terms of its time occurrence, participants, work involved, goal, functional decomposition, and end results. We then show the Partitioned Data flow diagrams of this phase that result from using PANDA.

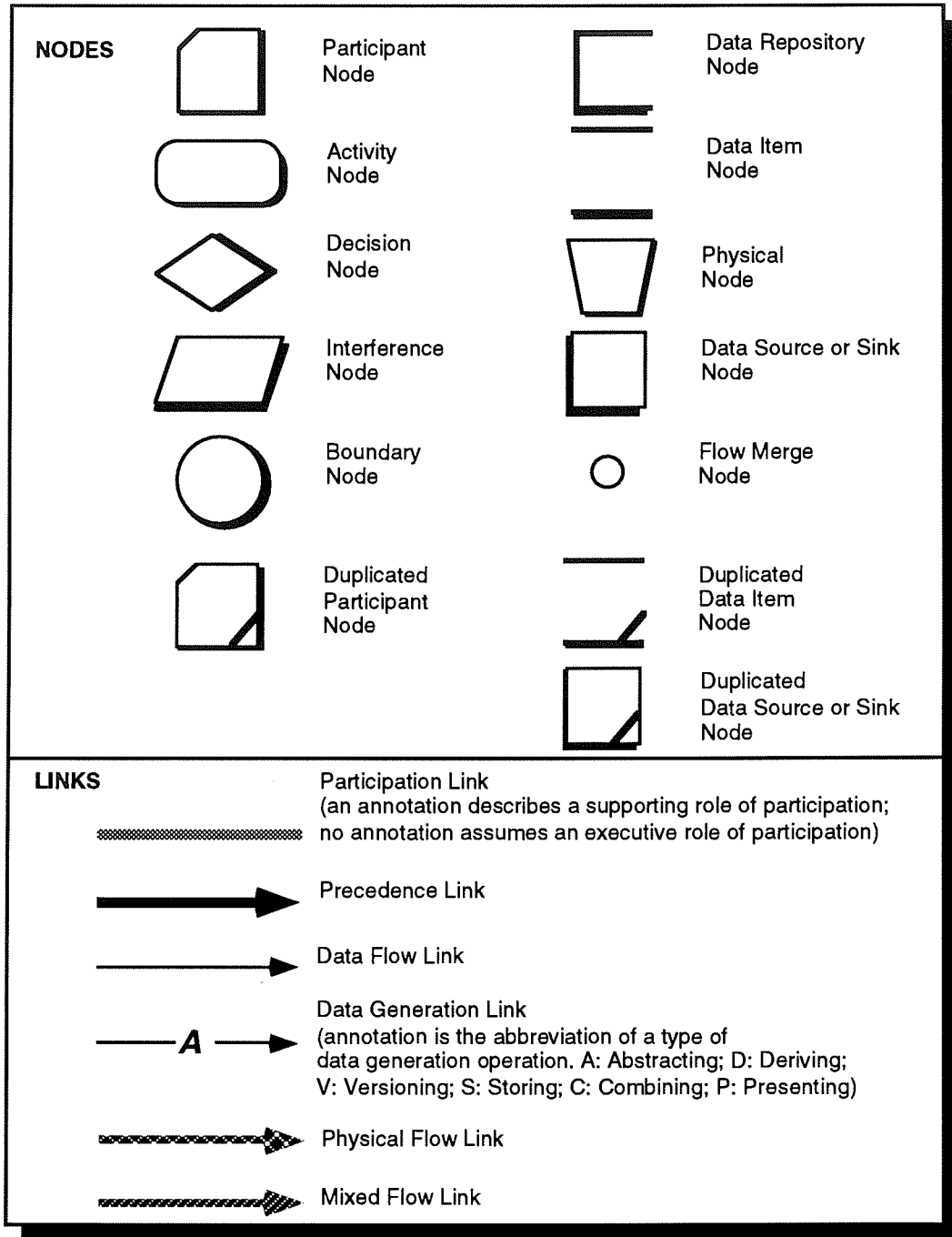
**Description of the Tower Construction Planning Phase** This phase occurs once a tower schematic drawing had been finished and can be used to work out all the necessary construction details. The key participants are structure detailers, which can be either company or contract employees. This phase involves the development of the fabrication and erection details in preparation for tower construction. The goal is to plan the construction phase in order to minimize errors and maximize efficiency. This phase consists of four functions:

- *Determining the Dimensions of Members and Laying out Connections:* The main objective of this function is to calculate the overall length, slope and bevel of all members and determine the shape and size of the connection plates, their hole patterns, and required ringfills and bolt lengths.
- *Detailing Fabrication Parts:* The detailer determines the number of fabrication parts needed and details each part by specifying its fabrication features, such as fabrication dimensions, gage lines, hole patterns, hole diameters, edge clippings, etc.
- *Generating Detailed Drawing:* The detailed drawing of the tower includes the information necessary to fabricate and construct the structure: detailed sketches of the tower structure, foundation setting plan, a bolt schedule listing all the hardware (e.g., bolts, nuts, ringfills) needed, the raw and galvanized weights of the tower structure, and all details of the fabrication parts.
- *Compiling Erection Bill of Material and Bundling List:* The detailer compiles a detailed list of all fabrication parts and groups them into separate bundles according to their mark number, size, length, quantity and weight. The purpose of this function is to facilitate shipment from the fabrication shop to the site, as well as to aid site handling by the construction crew.

All detailed design data for the tower is generated by the end of this phase. The deliverables are the detailed drawing and the erection bill of material and bundling list.

**Partitioned Data Flow Diagrams of the Construction Planning Phase** Due to space constraints, Figures 4.2 and 4.3 first show the legend and diagram notes for all the P-diagrams that follow. The legend includes the graphical symbols for the concepts of PANDA. Diagram notes are general notes that apply to an entire diagram. Figures 4.4 to

4.6 then show the partitioned data flow diagrams. The first diagram is the phase's skeletal functional schema. The remaining diagrams are the detailed schemata of the functions.

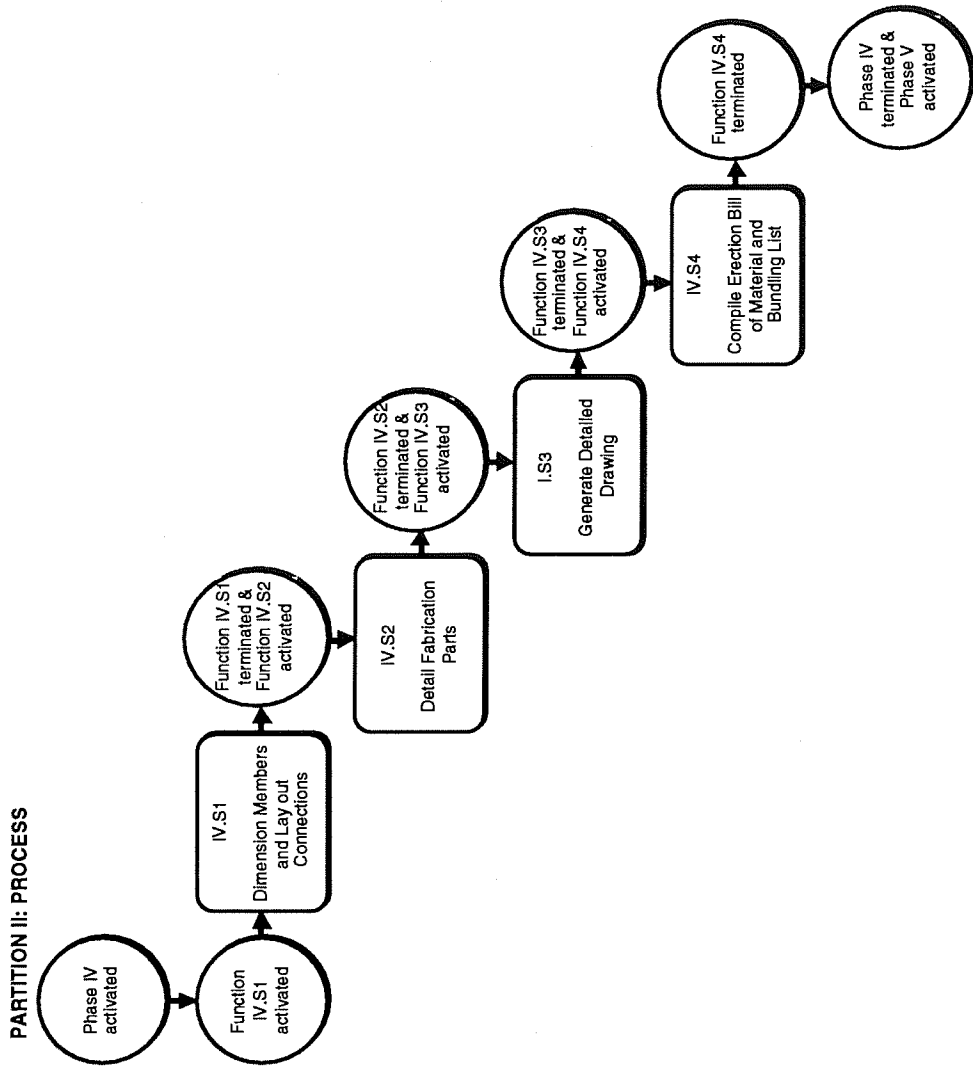


**FIGURE 4.2: Legend for the Partitioned Data Flow Diagrams that follow.** graphical representations of duplicated nodes were inspired by Gane's and Sarson's version of the Data Flow model [Gane 79].

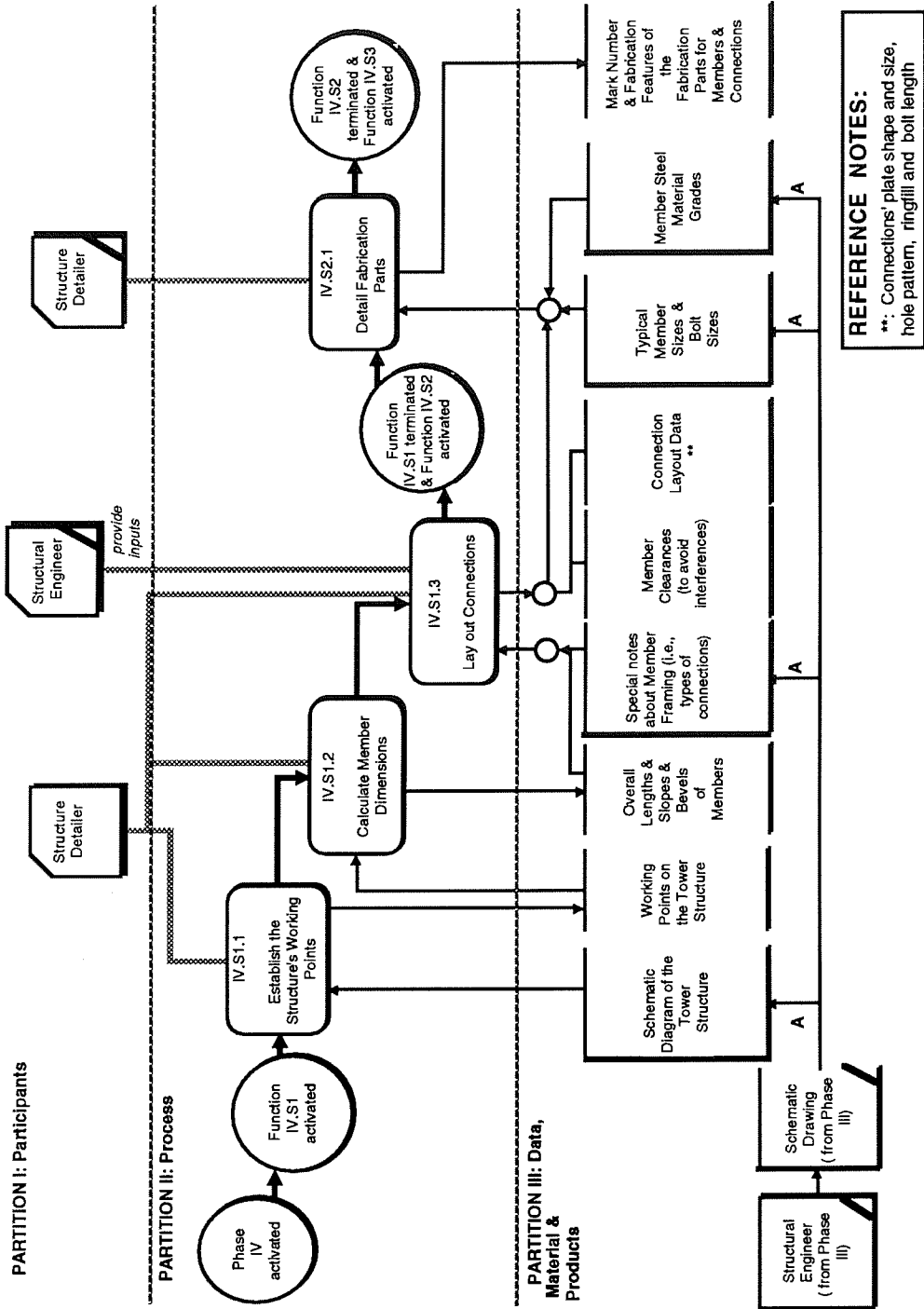
## DIAGRAM NOTES

- A. This diagram is an intermediate skeletal graphical functional schema of a phase. It illustrates mainly the breakdown of a phase into several functions. This schema results from the first step, 1.1, of the method presented in Section 4.4.1. At this level, the three partitions do not reveal all the details about the participants, activities, data, material and products. The diagrams that follow will reveal those details for the individual functions of this phase.
- B. This diagram is a detailed skeletal graphical functional schema of one or more functions. It illustrates the breakdown of the function or functions into several activities. It shows all the details in three partitions: (1) Participants, (2) Process and (3) Data, Material and Products. This schema results from completing all the steps in the two passes of the method described in Section 4.4.1.

**FIGURE 4.3: Diagram Notes for the Partitioned Data Flow Diagrams that follow.**

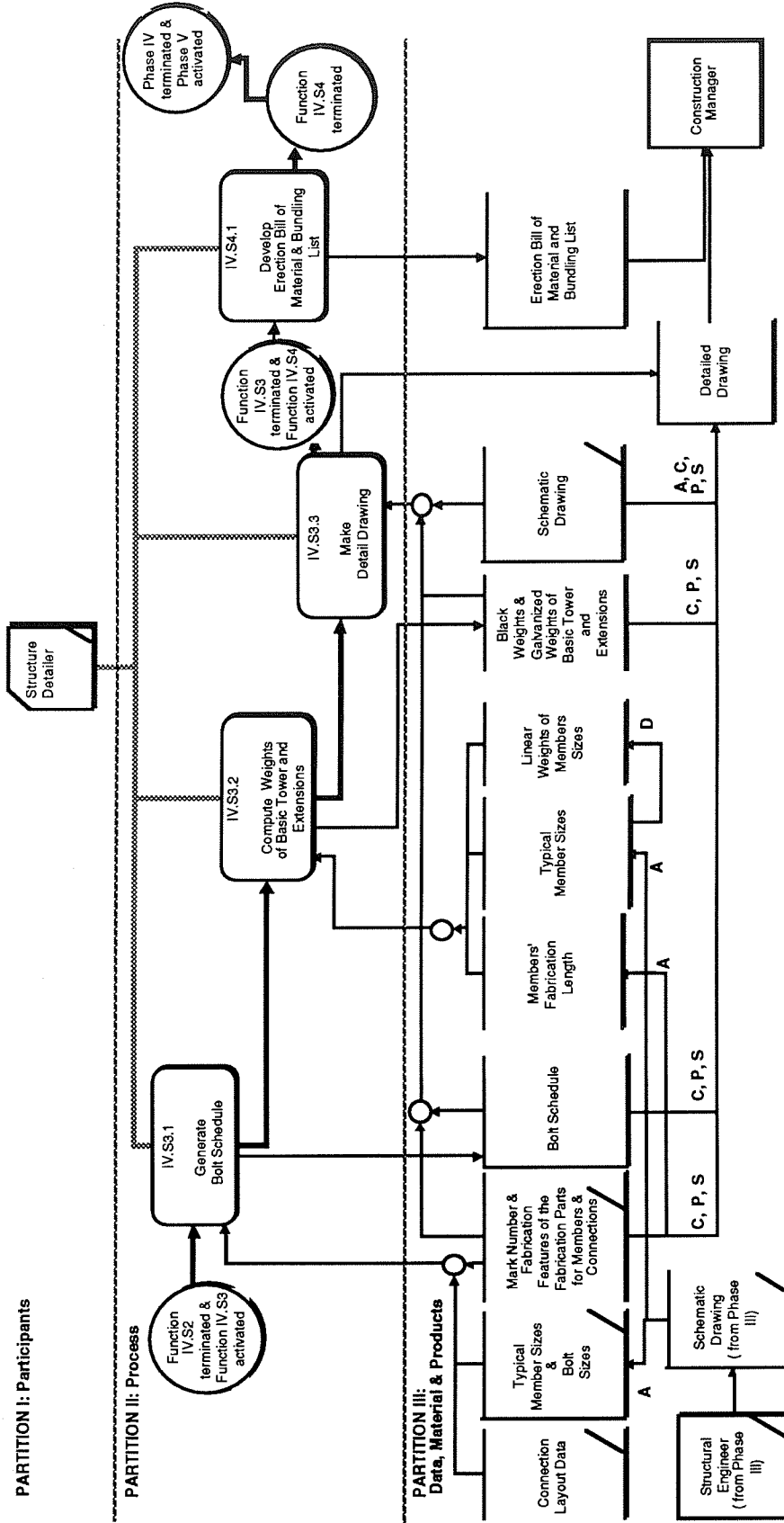


**FIGURE 4.4: Intermediate Skeletal of Phase IV, Tower Construction Planning.**  
(See Diagram Note A)



**FIGURE 4.5: Function IV.S1, Dimensioning Members and Laying out Connections, and Function IV.S2, Detailing Fabrication Parts, of Phase IV (Tower Construction Planning).** (See Diagram Note B)





**FIGURE 4.6: Function IV.S3, Generating Detailed Drawing, and Function IV.S4, Compiling Erection Bill of Material and Bundling List, of Phase IV (Tower Construction Planning).** (See Diagram Note B)

## **4.5 Chapter Summary**

---

The P-C Approach provides the PARTitioned eNginEering DATA flow model (or PANDA), an extension of the Data Flow model. PANDA can assist the modeler in the functional analysis of the complex facility engineering process in the domain. This chapter described the development of PANDA. It also presented the concepts, graphical representations, syntactic and semantic rules, and schema transformation operations of PANDA. The graphical functional schemata that result from using PANDA shows the participants and their roles in activities of the process; the decomposition of the process into subprocesses and activities, including the precedence relationships among activities, the design synthesis loops, the decisions and alternatives, and the process interferences; and the data, material and product flow networks. Having a better understanding of the engineering process, the modeler analyzes data used by domain experts in the next phase. This analysis, the “domain entities analysis,” makes use of the information from the process functional schemata. The next chapter describes the analysis and the Domain Entities AnaLysis method (or DEAL) provided by the P-C Approach to carry out the analysis.



# Chapter 5

---

## Domain Entities Analysis Using DEAL

---

### **Chapter Abstract:**

In the P-C Approach, domain entities analysis involves decomposing entities used by experts in a given domain (or “domain entities”) into conceptual “primitive entities” based on the criteria of cohesion and reusability. Cohesion measures how closely data items of an entity relate to one another. Reusability measures the extent to which an entity can be reused in describing domain entities. The evaluation of cohesion is based on the information of how data is generated and used in the engineering process modeled in the previous functional analysis using PANDA. The primitive entities are later used to design the domain primitive schema. This chapter first explains the need for doing domain entities analysis prior to schema design. It then gives an overview of the Domain Entities AnaLysis method (or DEAL) for doing the analysis. It also presents DEAL’s concepts, terms, assumptions, graphical representation, procedures (in two versions, DEAL-1 and DEAL-2), operations, and rules. Next, this chapter gives an example of using DEAL. Finally, it summarizes DEAL.

### **Organization:**

- 5.1 “What Should An Entity Be?”: The Domain Entities Analysis in the P-C Approach
- 5.2 An Overview of the Domain Entities AnaLysis method (or DEAL)
- 5.3 A Formal View of DEAL
  - 5.3.1 Basic Concepts and Terms
  - 5.3.2 Assumptions
  - 5.3.3 Graphical Representation: Domain Entity Decomposition Tree
  - 5.3.4 Procedures and Operations in DEAL-1 and DEAL-2 Versions
  - 5.3.5 Rules
- 5.4 Example of Using DEAL-1
- 5.5 DEAL as a Medium for Mediating Data Representations
- 5.6 Chapter Summary

## **5.1 “What Should An Entity Be?”: The Domain Entities Analysis in the P-C Approach**

---

Conceptual data modeling in general, and conceptual database design in particular, involves describing the content of the database or information system being developed. The resulting “conceptual schema” provides a high-level description of that content for use in the later implementation of the database or information system. According to Batini et al. [92], the Entity-Relationship (or E-R) model described in [Chen 76] has been the leading model for conceptual data modeling. In the E-R model, entities represent distinct “things,” abstract or concrete, in the domain of interest. Relationships define the ways in which the entities are associated with one another in that domain. Attributes represent properties of entities and relationships. However, what should an entity be? Or to be specific, how should the modeler go about identifying entities in the given domain? Which attributes should be included in an entity definition? Indeed, identifying and defining the proper entities in facility engineering domains is not a trivial task, as we learned in this research project. This task becomes even more difficult when the goal is to build a schema supporting data integration. First, the modeler can be overwhelmed with the size and complexity of the domain and particularly, with the large amount of data that is generated during the facility life-cycle and exchanged among activities. Second, experts in the domain typically design objects such as beams, columns, and floors that are highly data-intensive. A beam, for instance, may be described by geometry, material, fabrication features, load-resisting functions, bending stresses, etc. Entities representing these data-intensive design objects are called “domain entities.” In short, as Wimmer and Wimmer [92] point out, methods clearly need to be developed to aid the modeler with the conceptual data modeling of large complex engineering domains.

To respond to this need, the P-C Approach incorporates an analysis phase, the Domain Entities Analysis, prior to the schema design phase. This analysis involves decomposing the domain entities into “primitive entities” using the criteria of cohesion and reusability. The input domain entities can be easily identified by interviewing domain experts. Cohesion then helps the modeler decide which attributes should be included in an entity definition. By evaluating an entity’s cohesion, the modeler must consider five principal dimensions: (1) how the data is organized and thus how it can be accessed by humans in the work environment, (2) to which concepts the data relates, (3) at what time the data is created, (4) from which computational sources the data is derived, and (5) how the data is used in activities of the engineering process. The evaluation of cohesion is based on the information of how data is generated and used in the engineering process of the domain. This information comes from the process functional schemata that result from the previous functional analysis (Phase 2) using PANDA. Reusability ensures that each entity will be reused as much as possible and thereby improves modeling efficiency. The resulting primitive entities are designed into object classes of the domain primitive schema in the subsequent phase. The P-C Approach also provides a method for doing the analysis, the Domain Entities AnaLysis method (or DEAL). This method can aid the modeler with the complex conceptual modeling task described in Chapter 1.

## **5.2 An Overview of the Domain Entities AnaLysis method (or DEAL)**

---

We first need to introduce some basic terminology. An “entity” here represents either an abstract concept or a group of similar objects in the real world. Each entity is described by several “data items.” A data item (a simpler term for “attribute”) corresponds to a piece of data that delineates a single property of an entity. Cohesion and

reusability are familiar concepts in software engineering [Yourdon 79]. These concepts are applied here to facility data modeling, and each of them is given a new definition. “Cohesion” is defined as a measurement that indicates how closely the data items of an entity relate to one another. The P-C Approach considers five principal dimensions of cohesion: (1) *organization/access* (how the data is organized and thus accessed by humans in the work environment), (2) *concept* (to which concepts the data relates), (3) *time* (at what time the data is created), (4) *source* (from which computational sources the data is derived), and (5) *use* (how the data is used in activities of the engineering process). We call these dimensions “access-cohesion,” “concept-cohesion,” “time-cohesion,” “source-cohesion,” and “use-cohesion” respectively. “Reusability” is another measurement that indicates the extent to which an entity can be reused (i.e., used without modifications) in describing other domain entities. The P-C Approach incorporates five levels of reusability: reusable (1) for more than one domain entity of a common type, (2) for a single domain, (3) for a single industry, which includes more than one domain, (4) for more than one industry of a common type, and (5) for more than one industry type. We name these five levels: “domain-entity-type reusability,” “domain reusability,” “industry (or domain-type) reusability,” “industry-type reusability,” and “universal reusability.”

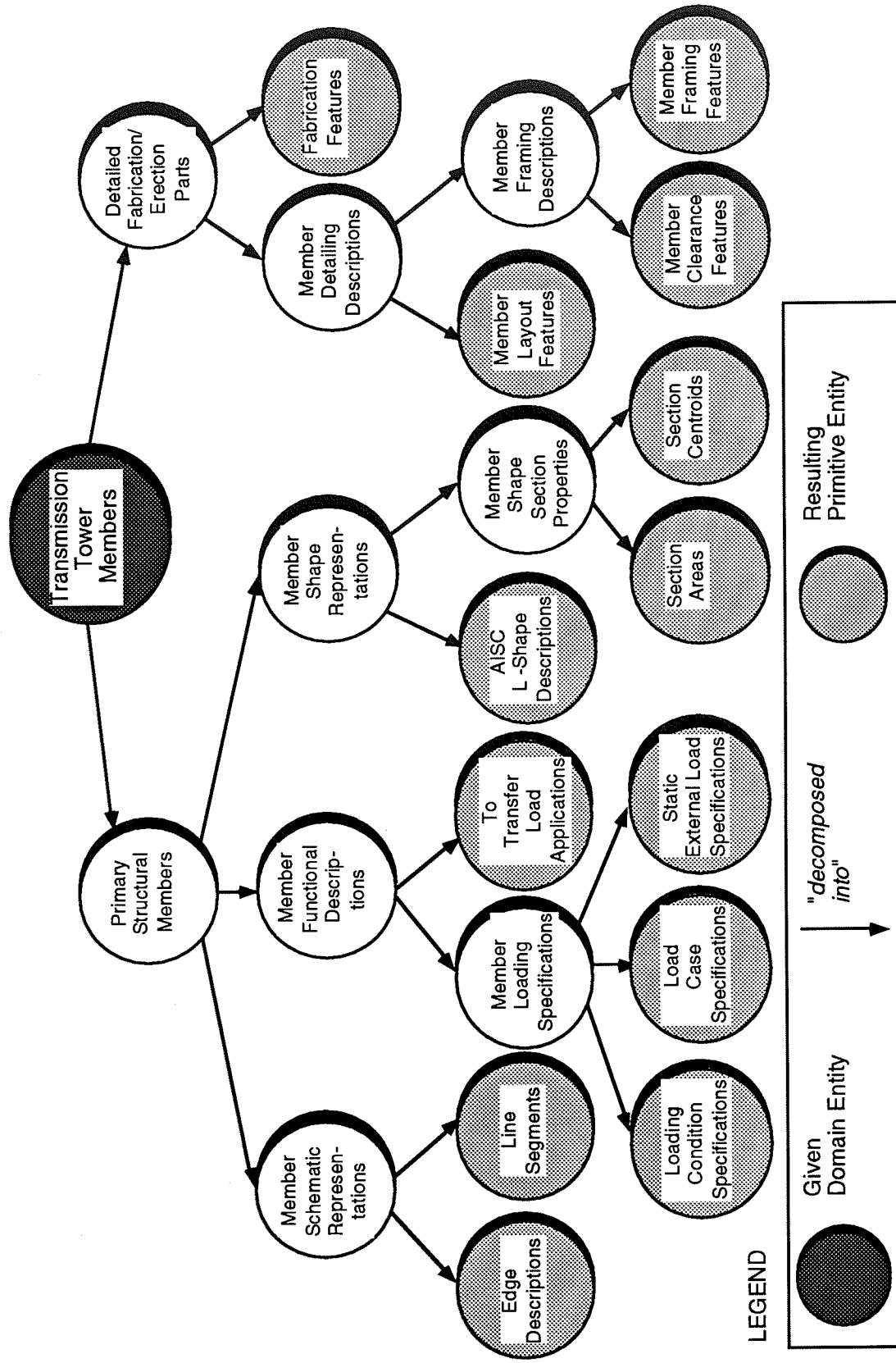
As mentioned before, domain entities represent complex facility design objects in the domain. These domain entities are typically described by a large set of data items. However, these entities are not cohesive or reusable. For instance, the data items of a “Transmission Tower Members” domain entity relate to various conceptual categories concerning form, function, and behavior. Thus, by definition, this entity is not concept-cohesive. These data items are also instantiated at different times in the facility life-cycle and, therefore, the entity is not time-cohesive either. The entire entity description cannot be reused in describing another domain entity. Thus, cohesion and reusability provide the main criteria for analyzing domain entities and decomposing them into cohesive and reusable entities: Entities that are not cohesive will be decomposed in the domain entities analysis. The end result is the primitive entities. Reusability is taken into account when more than one domain entity is analyzed and several primitive entities are identified.

In short, as Figure 5.1 illustrates, *the domain entities analysis using DEAL involves a step-by-step, top-down decomposition of domain entities into increasingly cohesive and reusable entities, ultimately resulting in the primitive entities of the domain.* Section 5.4 will explain the analysis shown in this figure.

DEAL includes concepts, terms, assumptions, graphical representation, procedures, operations, and rules. The DEAL procedures correspond to two versions:

**DEAL-1**—a basic version that considers only cohesion. This version successively examines the five cohesion criteria. Under each criterion, it carries out four operations. When an operation is carried out, the appropriate rules are also checked. This version is primarily used when the modeler has no previous experience with decomposing domain entities. In addition, it employs an elaboration technique that allows the inexperienced modeler to introduce gradually data items that are necessary to describe the domain entity being analyzed. For these reasons, this version is also called the “naive” version.

**DEAL-2**—an improved (or semi-naive) version that takes full advantage of the previous decomposition of domain entities and the primitive entities thus produced. This version considers both cohesion and reusability. It is similar to, but more efficient than its basic counterpart since the original operations are modified in order to enable the modeler to reuse the available primitive entities as much as possible. DEAL-2 also includes new operations that give the experienced modeler maximum control over where and when the entities’ decomposition terminate.



**FIGURE 5.1: Analysis of A Given Domain Entity Using DEAL.**

DEAL also provides a medium with which a modeler can gradually elicit knowledge about the data used in the domain from a domain expert. In return, the modeler can incrementally show the domain expert what the representations of that data will look like.

### **5.3 A Formal View of DEAL**

---

#### **5.3.1 Basic Concepts and Terms**

##### **5.3.1.1 Entity, Instance, and Data Item**

**Entity**—An entity represents either an abstract concept or a group of similar objects in the real world.

*Example:* In Figure 5.1, “To Transfer Load Applications” represents an abstract concept, whereas “Transmission Tower Members” represents a group of real-world objects.

**Instance**—An instance corresponds to a unique occurrence of an entity.

*Example:* “leg member 23” is an instance of the “Transmission Tower Members” entity.

**Entity Description & Data Item**—An entity is “described” in terms of data items. Each data item corresponds to a single piece of data that delineates a property of the abstract concept or real-world objects represented by the entity, or a relationship to another entity. (“Data item” is a simpler term for “attribute.” We use the former for entities in the analysis phases, including the previous functional analysis, and the latter for classes in the schema design phase.)

*Example:* “fabrication length,” “material yield strength,” “approximate weight,” and “derived section properties” are among the data items that form a description of the “Transmission Tower Members” entity.

**Data Item Value**—Each data item of an instance has a specific value.

*Example:* The “fabrication length” data item of the “leg member 23” instance has a value of 6.5 feet.

**Data Item Value Type**—Each data item value is associated with a data type. This data type can be scalar (i.e., integer and real numbers), character (single characters or character strings), Boolean (i.e., true and false), enumeration, user-defined data structures or “abstract data types” (e.g., “Coordinate,” “Direction,” “Date,” “Time”), aggregate (e.g., arrays, lists, bags, sets), or reference (i.e., logical pointers to other entities).

##### **5.3.1.2 Conceptual Categories**

**Conceptual Category**—Conceptual categories constitute a taxonomy for classifying entities’ data items to evaluate their concept-cohesion. The conceptual categories of an entity are the superset of the conceptual categories of its data items.

*Example:* From the preliminary study of the transmission tower domain, we identified conceptual categories such as Spatial Reference Form, Geometry Form, Topology Form, Shape Representation Form, Material Form, Part Detailing/Fabrication Form, Strength Behavior, Serviceability Behavior, Functions, Requirements, etc. We will explain these categories in detail in Chapter 7.



### 5.3.1.3 Domains and Their Classifications

(The following definitions will be used later to define levels of reusability.)

**Domain**—A domain corresponds to a particular product that provides an intended use or service.

*Example:* Electrical Utility Transmission Towers, Steel-Framed Warehouses, Reinforced-Concrete Buildings, and Cable-Suspended Steel Bridges.

**Domain Type (or Industry)**—A domain can be classified into “domain types.” The term “industry” is also used interchangeably with domain types. An industry represents a group of profit-making enterprises that produce and supply a particular product type [Webster 86].

*Example:* Buildings, Bridges, Roads, Process Plants, Ships, and Utilities.

**Industry Type**—Industries can be further classified into “industry types.” The classifications of domains into domain types and industry types are designed to be compatible with the three STEP layers of entities (i.e., Product-type layer, Industry-type layer and General STEP layer) [Geilingh 88].

*Example:* A/E/C, Mechanical Products, Electrical and Electronic Products, Computer Software/Hardware Products, Aerospace, and Automobile (as identified by STEP [Geilingh 88]).

### 5.3.1.4 Cohesion

**Cohesion**—Cohesion is a measurement that indicates how closely the data items of an entity relate to one another. This (composite) measurement consists of five dimensions: “access-cohesion,” “concept-cohesion,” “time-cohesion,” “source-cohesion,” and “use-cohesion.” All these dimensions have nominal values of yes or no. DEAL uses cohesion as a criterion for analyzing domain entities such as “Transmission Tower Members.”

**Access-cohesion**—This dimension is useful and most evident to consider in the domain entities analysis. The way in which people organize and gain access to information in the work environment suggest ways in which data should be represented. A primitive entity should not include data items that are organized in different places (e.g., project design files, structure drawings, program input files, program output files) and that therefore, must be accessed in different ways. Formally, *an entity is access-cohesive if all its data items can be accessed by the same logical access path.* A “logical access path” specifies a way to access the data and not a physical path. Each specification includes a sequence of items: a data repository name, and a key identifier, or a keyword, separated by a dot. Data repository names are shown in brackets and keywords are shown in quotes. *The information needed to define a logical access path and to evaluate access-cohesion comes from the process functional schemata using PANDA.* Access-cohesion can be evaluated by reviewing the repositories from which the data items under consideration are abstracted. In the domain entities analysis, if an entity is not access-cohesive, it is decomposed.

*Example:* In Figure 5.1, the “AISC L-Shape Descriptions” entity, whose description consists of “dimension d1,” “dimension d2,” and “thickness t” data items, is access-cohesive. Its logical access path is [AISC Manual]. “Angles.” Designation. This path includes a data repository name followed by a keyword and an identifier.

*Counterexample:* “Transmission Tower Members” is not access-cohesive since its data items are organized in various places in the tower engineering work environment and thus, must be accessed in many different ways.

**Concept-cohesion**—Concept-cohesion is the next important dimension to consider. Minsky [75] emphasizes that in solving a complex problem, people generally work with a chunk of the problem at a time. Moreover, a designer typically deals with different concepts at different times in the design process. For example, during the tower conceptual design, a structural engineer considers engineering functions, geometry and topology. Analysis element types, section properties, material properties, and element behavior are concepts essential to the structural components’ analysis and design during the tower detailed design. Therefore, a primitive entity should not include data items that pertain to different conceptual categories (defined earlier in Section 5.3.2.1). Formally, *an entity is concept-cohesive if all its data items relate to the same conceptual categories. The conceptual categories of the data were identified in the previous preliminary domain study (Phase 1).* However, the modeler must provide additional knowledge about the categories to which the data items under consideration relate. In the domain entities analysis, if an entity is not concept-cohesive, it is decomposed.

*Example:* The “AISC L-Shape Descriptions” entity described above is concept-cohesive. All of its data items relate to the Shape Representation Form conceptual category.

*Counterexample:* The “Tower Member Schematic Representations” entity is not concept-cohesive since its description into edges (topology) and lines (geometry) encompasses two conceptual categories: Topology Form and Geometry Form.

**Time-cohesion**—This dimension is essential to the domain entities analysis. The representation of data in a given domain should reflect the way in which activities of the engineering process in the domain generate data. A primitive entity should not include data items that are instantiated at different times in that process. By “instantiated,” we mean that the value of the data item is specified. Formally, *an entity is time-cohesive if all its data items are instantiated at the same logical time.* A logical time is not a physical clock time. Instead, each logical time is defined in two parts:

- a “time reference”: a time symbol that refers to a particular activity of the engineering process in which the data item is instantiated. A time reference can also be another logical time.
- a “relative time”: an integer that indicates the order in which the data item is instantiated relative to other data items that have the same time reference. For instance, both geometric data items (points and lines) and topological data (edges) are instantiated in the same activity of configuring the tower geometry. Both have the same time reference. However, the geometric data can be instantiated before the topological data and thus, can be given an earlier relative time.

The logical time is specified using this format: Time reference | Relative time. For instance, t1|1, where t1 refers to Activity “Determine Load Paths and Structural Systems” (labeled II.S2.3 in Appendix A). *The information needed to define a logical time comes from the process functional schemata using PANDA.* Time-cohesion can be evaluated by reviewing the activities that generate as output the data items under consideration. In the domain entities analysis, if an entity is not time-cohesive, it is decomposed.

*Example:* “AISC L-Shape Descriptions” is time-cohesive: All four data items have values specified when an instance is created.

*Counterexample:* The “Tower Member Loading Specifications” entity is not time-cohesive since its data items describing the loading conditions, load cases, loads, and load trees are instantiated in four separate activities of the tower engineering process (labeled II.S1.4.B, II.S1.4.C, II.S1.5 and II.S1.6 respectively in Appendix A).

**Source-cohesion**—This dimension helps separate representations of data whose value is derived from other data. The derivation normally takes time. The derived data is typically generated after the source data. Therefore, a primitive entity should not combine the derived data with the source data. Formally, *an entity is source-cohesive if all its data items are computed using sources outside of the entity, or simply if no data item of the entity can be used to compute other data items of the entity.* A subset of the entity’s data items that can be used to compute other data items of the entity is called an “internal computation source.” *The information about how the data is computed in the engineering process comes from the process functional schemata using PANDA.* Source-cohesion can be evaluated by reviewing the data repositories or other data items from which the data items under consideration are derived. In the domain entities analysis, if an entity is not source-cohesive, it is decomposed.

*Example:* “AISC L-Shape Descriptions,” whose description consists of “dimension d1,” “dimension d2,” and “thickness t” data items, is source-cohesive since none of its data item can be a computation source for other data items of the entity.

*Counterexample:* The “Rectangle Dimensions and Properties” entity, whose description consists of “length,” “width,” “area,” “moment of inertia,” and “centroid,” is not source-cohesive since the first two data items can be used to compute the last three data items. In this example, “length” and “width” form an internal computational source of this entity.

**Use-cohesion**—Data should be represented according to the way in which activities of the engineering process in the domain use data. For a given activity, a data representation is considered to be efficient if it does not include additional data items that are not used by that activity. In the domain entities analysis, such an entity is not “use-cohesive” and therefore, is decomposed. Formally, *an entity is use-cohesive if all or none of its data items are used in the primary data uses of the domain.* A data use is defined here as an occurrence in which data items of one or more entities are used in an activity. In fact, each data use corresponds to a single activity of the engineering process modeled in Phase 2. As an example, analyzing the tower structure (activity labeled II.S1.4 in Appendix A) uses data describing the analysis elements’ types, boundary conditions, lengths, cross-sectional areas, material properties, etc. The modeler can select a set of data uses in the domain, called the “primary data uses,” with which she optimizes the entities considered in the domain entities analysis. *The information about how the data is used in the engineering process comes from the process functional schemata using PANDA.* Use-cohesion can be evaluated by reviewing the activities that use as input the data items under consideration.

*Example:* “AISC L-Shape Descriptions” is use-cohesive in the domain of transmission towers since the primary data uses that we selected in the domain need all three data items describing the L-shape’s parametric dimensions.

*Counterexample:* The “Tower Member Shape Section Properties” entity, whose description includes “area,” “moment of inertia  $I_x$ ,” “moment of inertia  $I_y$ ,” “x-centroid,” “y-centroid,” “shear center,” etc. is not use-cohesive in the domain of transmission towers since structural analysis of the tower members does not need the data about the centroids and shear center.

### 5.3.1.5 Reusability

**Reusability**—*Reusability is a measurement that indicates the extent to which an entity can be reused (i.e., used without modifications) in the description of domain entities. This (ordinal) measurement is based on the five levels defined below. These levels are based on the definition and classifications of domains given earlier. DEAL uses reusability as another criterion for analyzing entities.*

**Domain-entity-type-reusable**—*An entity is domain-entity-type-reusable if it can be reused for more than one domain entity of a common type.*

*Example:* “AISC L-Shape Descriptions” is domain-entity-type-reusable since it can be reused in the description of “Transmission Tower Members,” “Transmission Tower Stub Angles,” and “Transmission Tower Base Shoes” domain entities, all of which are of the same “Tower Components” type. (We keep using “AISC L-Shape Descriptions” as example to conclude later that it is a primitive entity.)

**Domain-reusable**—*An entity is domain-reusable if it can be reused for more than one domain entity type in a single domain.*

*Example:* For the transmission tower domain, another domain entity type is “Tower Systems.” The “Edge Descriptions” entity is domain-reusable for that domain since it can be reused in the description of domain entities of both “Tower Components” and “Tower Systems” types.

**Industry-reusable (or domain-type-reusable)**—*An entity is industry-reusable if it can be reused for a single industry that includes more than one domain.*

*Example:* The “Line Segments” entity is industry-reusable since it can be reused for the Utilities industry, which includes the domains of Electrical Utility Transmission Towers and Electrical Utility Substations.

**Industry-type-reusable**—*An entity is industry-type-reusable if it can be reused for more than one industry of a common type.*

*Example:* The “Material Strength Properties” entity is industry-type-reusable since it can be reused for industries such as Buildings, Bridges, Ships, and Utilities of a common A/E/C type.

**Universe-reusable**—*An entity is universe-reusable if it can be reused for more than one industry type.*

*Example:* The “Cartesian Points” entity is universe-reusable since it can be reused for industry types such as A/E/C, Mechanical Products, Aerospace, and Automobile.

### 5.3.1.6 Domain Entity and Primitive Entity

**Domain Entity**—*A domain entity represents a facility design object created by experts in a domain. The description of a facility design object typically includes a large number of data items pertinent to several conceptual categories. To distinguish, a composite class represents a single user view about a facility design object. Thus, a composite class includes only the subset of data items that are of interest to that user. A domain entity can be seen as the union of all user views (or a unified view) about a facility design object.*

*Example:* For transmission towers, domain entities include “Transmission Tower Members,” “Transmission Tower Connections,” “Transmission Tower Base Shoes,” and “Tension-Compression Panel Systems.” Composite classes that correspond to the “Transmission Tower Members” domain entity can be “Transmission Tower Members

As Analyzed,” “Transmission Tower Members as Designed,” “Transmission Tower Members As Delivered,” and “Transmission Tower Members As Assembled” to name a few.

**Primitive Entity**—A primitive entity is a cohesive set of related data items that is reusable in the description of domain entities. Each primitive entity must be at least access-cohesive, concept-cohesive, time cohesive, source-cohesive, and domain-entity-type-reusable. In fact, access-cohesion, concept-cohesion, time-cohesion, and source-cohesion are necessary requirements of primitive entities. On the other hand, use-cohesion allows the modeler to further optimize primitive entities in terms of how they will be used. Such optimization can improve efficiency in certain cases, but it can also reduce the efficiency in others. Therefore, it is up to the modeler to decide for which primary data uses primitive entities are optimized.

*Example:* The entity “AISC L-Shape Descriptions” as described previously is a primitive entity.

*Counterexample:* Domain entities are not primitive. Also, the entities used in the previous counterexamples (i.e., “Tower Member Schematic Representations,” “Tower Member Loading Specifications,” and “Tower Member Shape Section Properties”) are not primitive.

### 5.3.2 Assumptions

The following assumptions are necessary for DEAL to work properly:

**Bounded Domain Assumption**—In the given domain, there exists a fixed number of domain entities.

*Implication:* In a given domain, a fixed number of domain entities will be analyzed. These are domain entities that interest the users of the database or information system being developed and, thus, the modeler. The modeler must identify those domain entities in the preliminary domain study (Phase 1) prior to the domain entities analysis.

**Finite Domain Entity Assumption**—Each domain entity is described by a finite, non-empty set of data items. (An entity description is only a simplified picture of the concept or objects that the entity represents.)

*Implication:* The analysis of a given domain entity will terminate.

**Unique Data Items Assumption**—All data items considered in the analysis are unique.

*Implication:* If two or more data items have different names but represent the same property, only one data item name will be selected for use in the analysis. If two or more data items have the same name but represent different properties, they will be included in the analysis under distinct names. The modeler must clearly define all data items in the data dictionary that is built during the functional analysis (Phase 2) preceding the domain entities analysis.

**Unified Domain Entities Assumption**—Each domain entity represents a unified view of a concept or of real-world objects as seen by all experts in the domain.

*Implication:* This assumption implies a form of view integration. If two or more domain entities have the same name but represent different experts’ views of the same facility design object, then all the descriptions of those domain entities will be integrated into one domain entity description in the analysis. If those entities also have different names, then only one entity name will be selected for use. Literature on view integration can be found in [Yao 78], [Ozsu 91], and [Gotthard 92]. In the course of identifying domain entities in

Phase 1 and building the data dictionary in Phase 2, the modeler must identify and resolve any conflicting domain entity definitions prior to the domain entities analysis.

**No-Knowledge Procedures Assumption**—The modeler provides the knowledge necessary to complete the DEAL procedures.

**Implication:** The procedures (in both DEAL-1 and DEAL-2 versions as presented later in Section 5.3.4) require a modeler's interaction and cooperation. Under the successive cohesion criteria considered in the analysis, the modeler must provide knowledge about the logical access paths, conceptual categories, logical times, computational sources, and primary data uses to which the individual data items relate. (The modeler either has this knowledge herself or acquires it from domain experts.) The procedures provide the steps, operations, and rules necessary to decompose entities.

### 5.3.3 Graphical Representation: Domain Entity Decomposition Tree

DEAL uses a graphical representation, the "domain entity decomposition tree," to carry out the analysis of a given domain entity. This section defines this tree and its elements, all of which will be used later to define the procedures and operations of DEAL. These definitions are also necessary for the development of a CASE tool automating DEAL.

**Tree Definition** A "domain entity decomposition tree" of a given domain entity  $e_d$ , noted as  $DT(e_d)$ , is a directed tree [Flores 70], where:

- the root (i.e., the vertex with no entering arcs) graphically represents the entity  $e_d$ ;
- other vertices graphically represent entities into which  $e_d$  is decomposed;
- an arc (or directed edge) from a vertex  $v_i$  to a vertex  $v_j$  denotes a "decomposition" relationship between the entity  $e_i$  represented by  $v_i$  and the entity  $e_j$  represented by  $v_j$ . The vertex  $v_j$  is a "child" of the vertex  $v_i$ .

Figure 5.1 earlier shows a sample domain entity decomposition tree.

**Decomposition Relation** The decomposition relation among entities in a domain entity decomposition tree, noted as  $R_D: E \rightarrow E$ , where  $E$  is the set of all entities in  $DT(e_d)$ , is defined as follows:

*An entity  $e_i$  is decomposed into an entity  $e_j$  if the description of  $e_i$  requires and includes the description of  $e_j$ .*

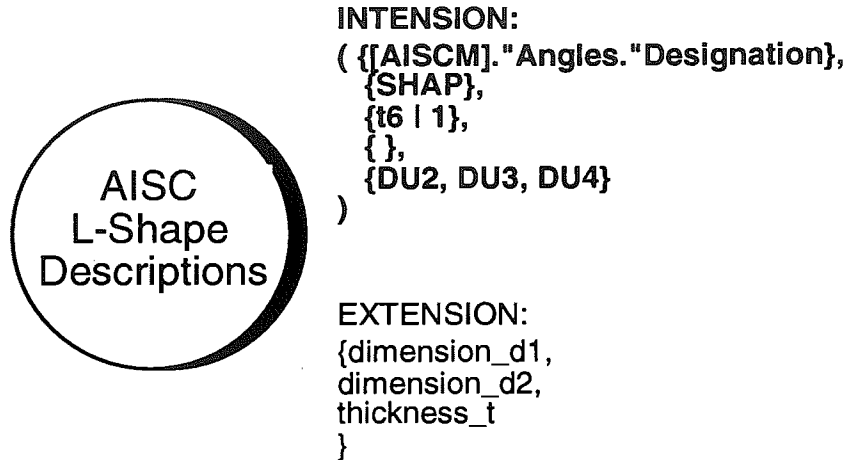
*Notation:*  $e_i \rightarrow e_j$  :  $e_i$  is "decomposed into"  $e_j$ .

This relation is a partial ordering: It is reflexive, asymmetric and transitive.

**Vertex Definitions** Each vertex of a tree, which graphically represents an entity, can be defined in two ways:

- **Vertex Extension**—An extension of a vertex  $v_i$  is the fixed set of data items of the entity that  $v_i$  represents.

$$\text{EXT}(v_i) = \{ d_i \mid d_i \text{ belongs to } e_i \text{ and } v_i \text{ represents } e_i \}$$



**FIGURE 5.2: Vertex Intension and Extension.** SHAP is an abbreviation of the Shape Representation Form conceptual category. DU2, DU3 and DU4 are symbols representing primary data uses.

- **Vertex Intension**—An intension of a vertex  $v_i$  is the tuple of elements  $(A, C, T, S, U)$  pertinent to the entity that  $v_i$  represents, where  $A$  is a set of logical access paths,  $C$  is a set of conceptual categories,  $T$  is a set of logical times,  $S$  is a set of internal computation sources, and  $U$  is a set of data uses. Logical access paths, conceptual categories, logical times, internal computation sources, and data uses were already explained in Section 5.3.1.4 on Cohesion.

$$\text{INT}(v_i) = (A, C, T, S, U)$$

Figure 5.2 shows the intension and extension of a vertex representing the “AISC L-Shape Descriptions” entity. As a convention, the vertex intension is shown above the vertex extension in the decomposition tree.

### 5.3.4 Procedures and Operations in DEAL-1 and DEAL-2 Versions

#### 5.3.4.1 The Basic Version, DEAL-1

DEAL-1 successively considers the access-cohesion, concept-cohesion, time-cohesion, and use-cohesion criteria. Under each criterion, four consecutive operations are performed: ELABORATE, ASSIGN, COHERE, and DECOMPOSE. The first operation is optional, whereas the others are required. These operations correspond to four steps of the procedure as shown below. The procedure is as follows:

**INPUT:** A domain entity  $e_d$ , which is initially described as a set of data items.  $e_d$  is the root and only vertex of the decomposition tree  $DT(e_d)$  at this point.

**OUTPUT:** Primitive entities  $e_p$ .

**PROCEDURE:****Step 1—Access-cohesion Criterion**

- 1.1 **ELABORATE:** For each leaf  $v_j$  of the tree, “elaborate” the entity  $e_j$  represented by  $v_j$  by replacing the current data items with many more data items that are also more specific than the current ones. *Elaboration is an important problem-solving technique in which the problem solver can add new assertions about the problem* [VanLehn 89]. These assertions reflect the problem solver’s current understanding of the problem. DEAL-1, the “naive” version, uses this technique to enable the modeler to “elaborate” gradually all the data items necessary to describe the domain entity being analyzed. (Check the safe elaboration rule.)
- 1.2 **ASSIGN:** For each leaf  $v_j$  of the tree, determine the unique logical access paths or combinations of paths for accessing the data items contained in  $v_j$ . The separate paths or path combinations to which data items are assigned are generically called “binders.” Assign each data item to its proper binder.
- 1.3 **COHERE:** For each leaf  $v_j$  of the tree, find the subsets of  $e_j$  in which all data items “cohere” to the same binder (i.e., can be accessed by the same logical access paths). These subsets, called “cohesive sets,” partition the entity  $e_i$ .
- 1.4 **DECOMPOSE:** For each leaf  $v_j$  of the tree, decompose  $v_j$  into child vertices  $v_k$ , each of which corresponds to a single cohesive set. Also check the non-loss, disjoint decomposition rule.

(optional) If new logical access paths that are either more specific or more direct than those of  $e_j$  can be determined for the newly created entities  $e_j$ , repeat Steps 1.1 to 1.4.

At the end of this step, the leaves of the tree represent access-cohesive entities.

**Step 2—Concept-cohesion Criterion**

- 2.1-4 Repeat the same operations as in 1.1 to 1.4, except using the concept-cohesion criterion. The ELABORATE operation has a restriction: The resulting entity must still be access-cohesive. In the elaboration technique, the problem-solver cannot invalidate any previous assertion when introducing a new one. The ASSIGN operation must be based on the pre-defined conceptual categories to which the data items relate.

At the end of this step, the leaves of the tree represent access-cohesive and concept-cohesive entities.

**Step 3—Time-cohesion Criterion**

- 3.1-4 Repeat the same operations as in 1.1 to 1.4, except using the time-cohesion criterion. The ELABORATE operation has a restriction: The resulting entity must still be access-cohesive and concept-cohesive. The ASSIGN operation must be based on the logical times at which the data items are instantiated.



Alternatively, the tree can be “time-ordered.” Given two logical times  $t_a$  and  $t_b$ ,  $t_a < t_b$  if:

- $t_a$  refers to an activity that precedes the one that  $t_b$  refers, or
- $t_a$  and  $t_b$  refer to the same activity, but  $t_a$ 's relative time is less than  $t_b$ 's relative time.

As an example,  $t_1 < t_2$  since  $t_1$  refers to the activity of configuring the tower geometry, and  $t_2$  refers to a later activity of the tower's structural analysis. To time-order a tree, arrange the leaves from left to right in the increasing time order as defined above. Arrange non-leaf vertices in a similar fashion using their children's smallest logical time for each vertex.

At the end of this step, the leaves of the tree represent access-cohesive, concept-cohesive and time-cohesive entities.

#### **Step 4—Source-cohesion Criterion**

4.1-4 Repeat the same operations as in 1.1 to 1.4, except using the source-cohesion criterion. The ELABORATE operation has a restriction: The resulting entity must still be access-cohesive, concept-cohesive and time-cohesive. The ASSIGN operation must be based on the computational sources from which the data items are derived.

At the end of this step, the leaves of the tree represent access-cohesive, concept-cohesive, time-cohesive and source-cohesive entities.

#### **Step 5—Use-cohesion Criterion**

5.1-4 Repeat the same operations as in 1.1 to 1.4, except using the use-cohesion criterion. This final step also requires all the leaves of the tree to be elaborated fully. The ELABORATE operation has one restriction: The resulting entity must still be access-cohesive, concept-cohesive, time-cohesive and source-cohesive. The ASSIGN operation must be based on the primary data uses for which the data items of  $e_j$  are used.

At the end of this step, the leaves of the tree represent access-cohesive, concept-cohesive, time-cohesive, source-cohesive and use-cohesive entities.

**PSEUDO-CODE:** The following pseudo-code shown below summarizes this procedure. Appendix B contains the pseudo-codes for the four operations.

```
procedure DEAL-1(vertex ed )
// This version only considers cohesion. CRIT is the set of
// criteria considered. The criterion A in CRIT stands for
// access-cohesion criterion, C for concept-cohesion, T for
// time-cohesion, S for source-cohesion, and U for use-
// cohesion.
begin
  for each criterion I in CRIT = { A, C, T, S, U } do
  begin

    L1:

    I.1  for each leaf vi in DT(ed) do
          optional ELABORATE(vi ) ;
    I.2  for each leaf vi in DT(ed) do ASSIGN(vi, I);
    I.3  for each leaf vi in DT(ed) do COHERE(vi, I);
    I.4  for each leaf vi in DT(ed) do DECOMPOSE(vi, I);

    // (optional) If the current criterion is access-
    // cohesion, choose the option to repeat steps I.1 to
    // I.4, this time using more specific and direct logical
    // access paths.
    if I == A then optional goto L1;
  end
end
```

### 5.3.4.2 The Improved Version, DEAL-2

**INPUT:** A domain entity  $e_d$ , which is initially described as a set of data items  $d_k$ .  
Previously analyzed domain entities  $e_d'$  and their domain entity decomposition trees  $DT(e_d')$ .  
Previously identified primitive entities  $e_p'$ .

**OUTPUT:** Primitive entities  $e_p$ .

**PROCEDURE:**

DEAL-2 follows the same procedure as DEAL-1, except for the following:

- A new optional operation, **RESTART\_AT**, can restart the procedure at one of the earlier steps. This operation allows the modeler to repeat an earlier step upon discovering that he or she has done something incorrectly. It is designed to give the modeler maximum control over the procedure.
- An operation, **SAVE\_TREE\_VERSION**, is automatically triggered at the end of each step to save the version of the tree that results from that step. This tree version will be retrieved for use if the modeler decides to **RESTART\_AT** the end of that step.
- A new optional operation, **TERMINATE**, allows the user to declare a node terminated. A terminated node will no longer be considered in the remaining steps of the procedure and will remain as a leaf. This operation gives the modeler more control over where and when the procedure terminates. For instance, after Step 4 of the procedure, the modeler may decide that an entity meets the requirements to be considered primitive (i.e., access-cohesion, concept-cohesion, time-cohesion, and source-cohesion) and needs no further analysis. The modeler can then declare the corresponding vertex as terminated.
- A new optional operation, **RECALL**, takes advantage of the previously analyzed domain entities  $e_d'$  by enabling the modeler to recall any vertices of the existing decomposition trees  $DT(e_d')$ . Therefore, when the modeler identifies that a vertex  $v_i$  of the tree under consideration is the same as a vertex  $v_j$  of a previously analyzed tree  $DT(e_d')$ , she can recall  $v_j$  to replace  $v_i$ . In that case,  $v_j$  and all its descendants  $v_k$  in  $DT(e_d')$  will replace  $v_i$ . Those  $v_k$  that are leaves are automatically terminated since they need not be considered in the remaining steps of the procedure. **RECALL** is designed to improve the efficiency of the procedure.
- The **ELABORATE**, **ASSIGN**, **COHERE**, and **DECOMPOSE** operations are modified to enable the modeler to reuse previously identified primitive entities ( $e_p'$ ). This is where DEAL-2 takes reusability into account. When those primitive entities are reused, their vertices are also automatically declared to be terminated.
- A new optional operation, **UNDO**, allows the user to roll back what has been done to a vertex. In that case, the extension and intension of the vertex will be reset to the ones that existed at the end of the previous step. (**UNDO** is a special case of **RESTART\_AT**, which enables the user to restart the procedure at any one of the earlier steps.)

**PSEUDO-CODE:** The pseudo-code shown on the next page summarizes the DEAL-2 procedure. Pseudo-codes for ELABORATE-2, ASSIGN-2, COHERE-2, and DECOMPOSE-2 are shown in Appendix B.

```

procedure DEAL-2(vertex  $e_d$ , trees  $D_T'$ , entities  $e_p'$  )

//This version considers both cohesion and reusability. The
criterion A in CRIT stands for access-cohesion, C for concept-
cohesion, T for time-cohesion, S for source-cohesion, and U for
use-cohesion.

begin

L0:  for each criterion I in CRIT = {A, C, T, S, U} do begin

      for each non-terminated leaf  $v_i$  in  $DT(e_d)$  do begin
          if (RECALLABLE ( $v_j$ ,  $D_T'$ )) then RECALL( $v_j$ ,  $v_i$ );
      end

      L1:
      I.0  for each non-terminated leaf  $v_i$  in  $DT(e_d)$  do
          optional ELABORATE-2( $v_i$  );
      I.1  for each non-terminated leaf  $v_i$  in  $DT(e_d)$  do
          ASSIGN-2( $v_i$ , I);
      I.2  for each non-terminated leaf  $v_i$  in  $DT(e_d)$  do
          COHERE-2( $v_i$ , I);
      I.3  for each non-terminated leaf  $v_i$  in  $DT(e_d)$  do
          DECOMPOSE-2( $v_i$  , I);

      // (optional) If the current criterion is access-
      cohesion, choose the option to repeat steps I.1 to
      I.4, this time using more specific and direct logical
      access paths.

      if I == A then optional goto L1;

      optional UNDO();

      if (user-input restart_procedure?) RESTART_AT (user-
      input I);

      end

end

```

### 5.3.5 Rules

The DEAL procedures are governed by two rules:

**Non-Loss, Disjoint Decomposition Rule**—If an entity  $e_i$  is decomposed into  $n$  entities  $e_j$ , then each non-key data item  $d_i$  of  $e_i$  also belongs to the extension of **one and only one** of the entities  $e_j$ . On the other hand, key data items with which users in the domain identify instances can be kept in more than one  $e_j$  in order to preserve information.

*Purpose:* As in normalization [Codd 71a], [Ozsu 91], this important rule insures that no data items will be lost during decomposition. It also insures that each entity is decomposed into other disjoint entities. In addition, it enables the reconstruction of entities represented in the tree: The extension and intension of a vertex  $v_i$  can be directly composed from those of its children  $v_j$  using the set union operator **U**. Consequently, this rule enables each step of the DEAL procedures to consider only the leaves of the tree. *This rule is checked after each decomposition of an entity.*

**Safe Elaboration Rule**—If two leaves  $v_i$  and  $v_j$  of the tree are elaborated and as a result, contain duplicate data items, they must be decomposed into three other child vertices, each of which has a distinct extension. One of those children gets the duplicate data items.

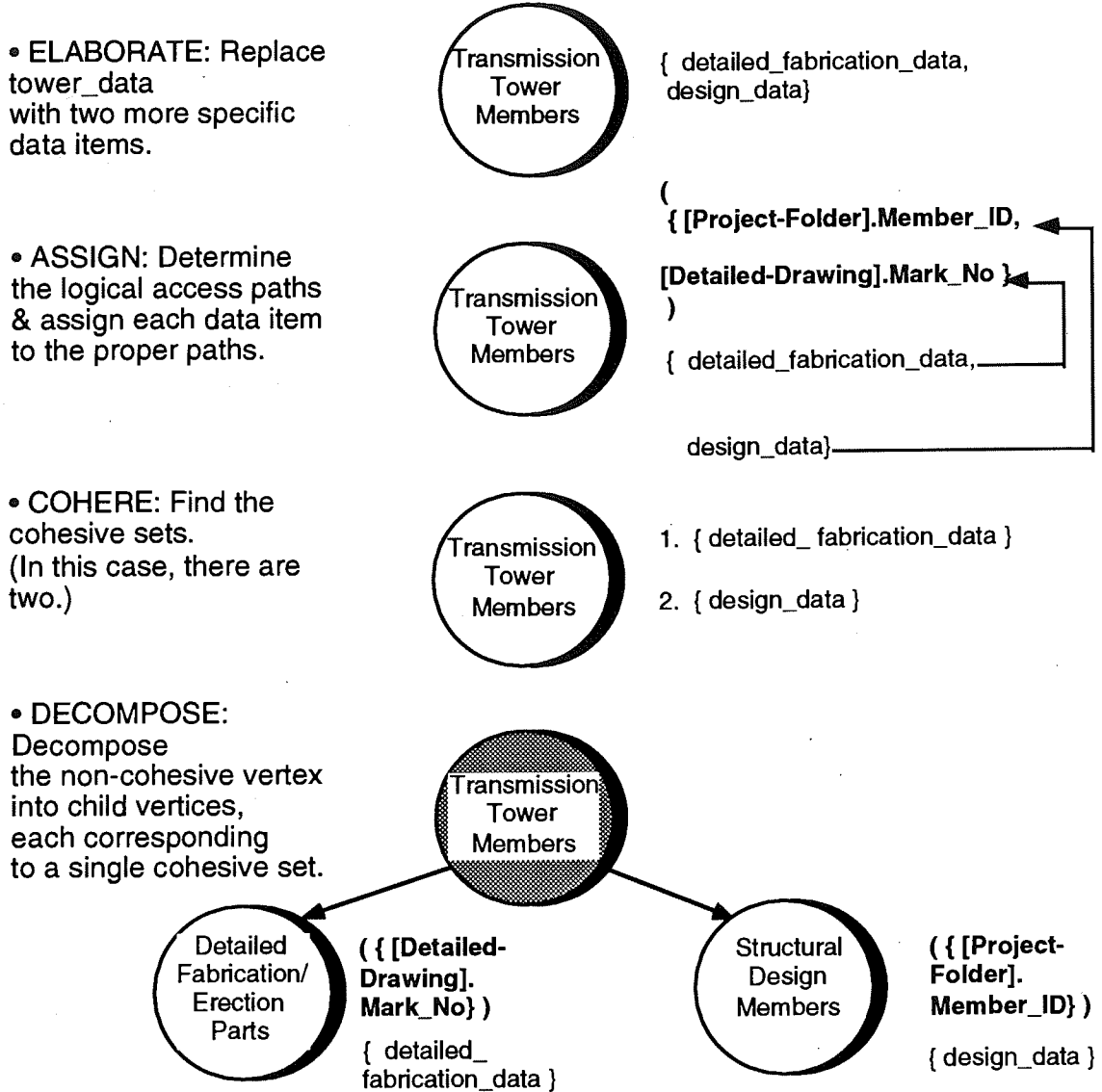
*Purpose:* The elaboration of entities in DEAL may introduce duplicate data items in different leaves of the tree. Therefore, this rule insures that no two leaves of the tree will contain duplicate data items. *This rule is checked after each elaboration.*

## 5.4 Example of Using DEAL-1

---

This section illustrates the use of DEAL-1 by giving a brief analysis of a domain entity called “Transmission Tower Members.” In this analysis, the domain entity is initially described by one data item, `tower_data`. Also, the intensions of the vertices are shown above their extensions in the decomposition trees, and data items whose name ends in “\_data” will be elaborated in a later step. A more detailed analysis is given in Appendix B.

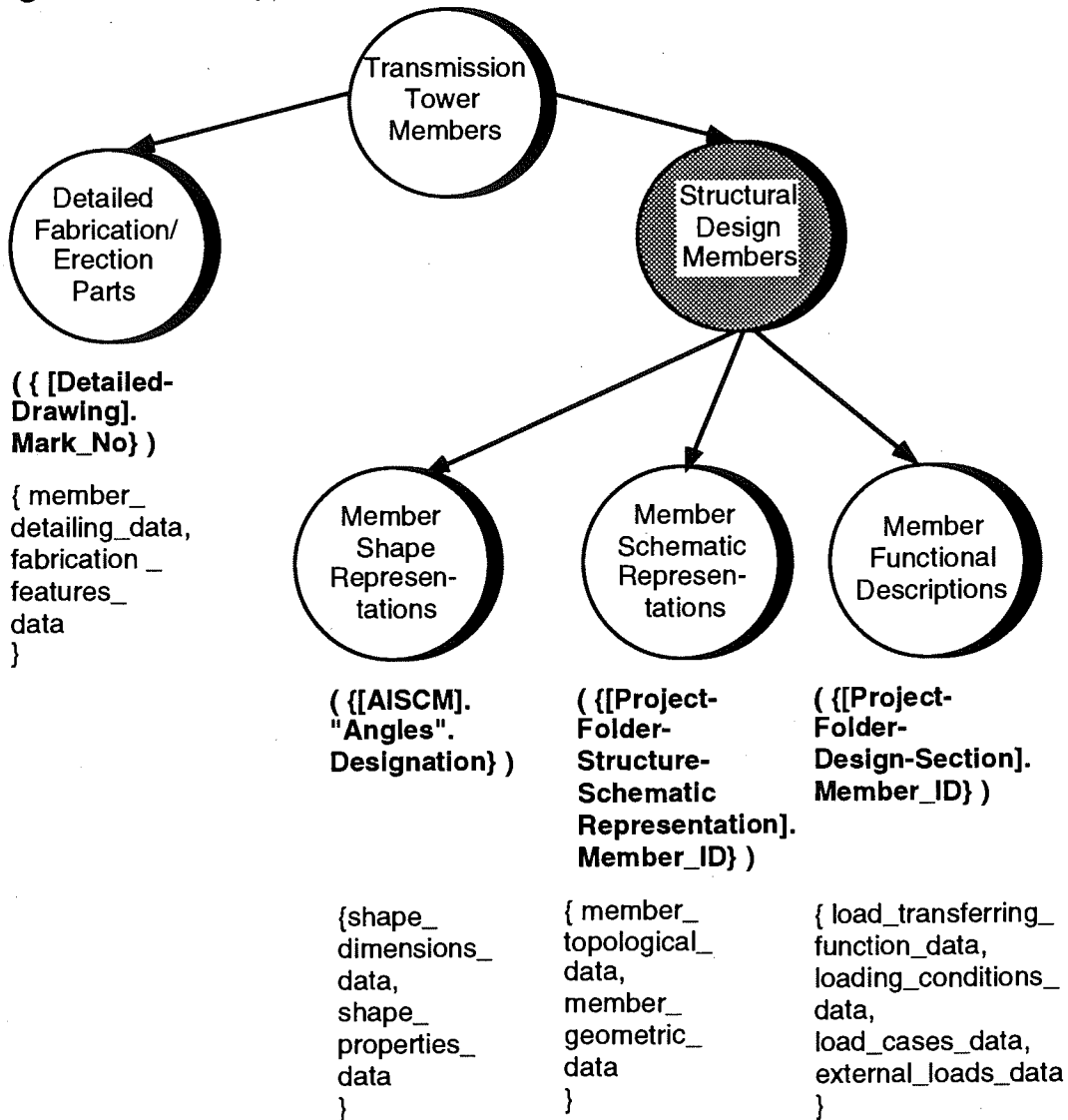
**Step 1: Considering Access-Cohesion** Figure 5.3 illustrates the first step that considers the access-cohesion criterion and involves the four operations (i.e., ELABORATE, ASSIGN, COHERE, and DECOMPOSE).



**FIGURE 5.3: Step 1 of DEAL-1 Considering Access-Cohesion.** The shaded vertex in the last operation is the non-access-cohesive one that gets decomposed.

The first step is repeated, this time using logical access paths that are either more specific or more direct than those shown in the previous figure. The newly introduced logical access paths are shown as part of the intension of the new leaves in Figure 5.4. Due to space constraints, this figure shows only the decomposition tree that results from this repetition. In addition, many new data items describing the transmission tower members appear in the extension of those leaves as the result of repeating the ELABORATE operation. These data items may come from the process functional schemata that result from the functional analysis in Phase 2.

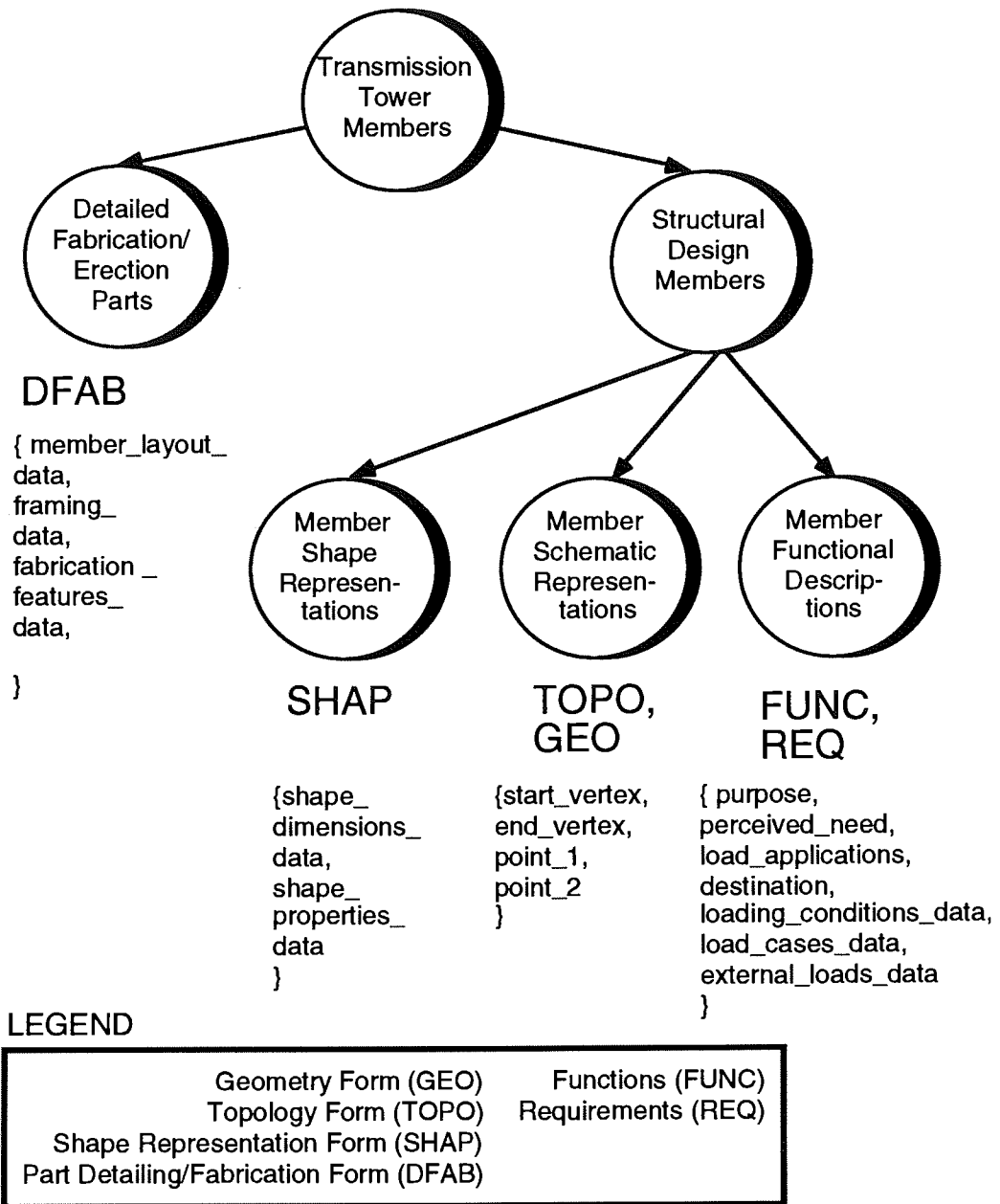
**STEP 1 (repeated):  
ACCESS-COHESION  
using direct and specific  
logical access paths**



**FIGURE 5.4: Repeating Step 1 of DEAL-1 Using Direct and Specific Logical Access Paths.** The shaded vertices are the non-access-cohesive ones that get decomposed.

**Step 2: Considering Concept-Cohesion** This step involves the same sequence of operations as the previous step. Figure 5.5 shows the conceptual categories assigned to the leaves of the decomposition tree during the ASSIGN operation. Again, new data items shown in the extension of the leaves were introduced as a result of the earlier ELABORATE operation in this step.

**STEP 2:  
CONCEPT-COHESION**

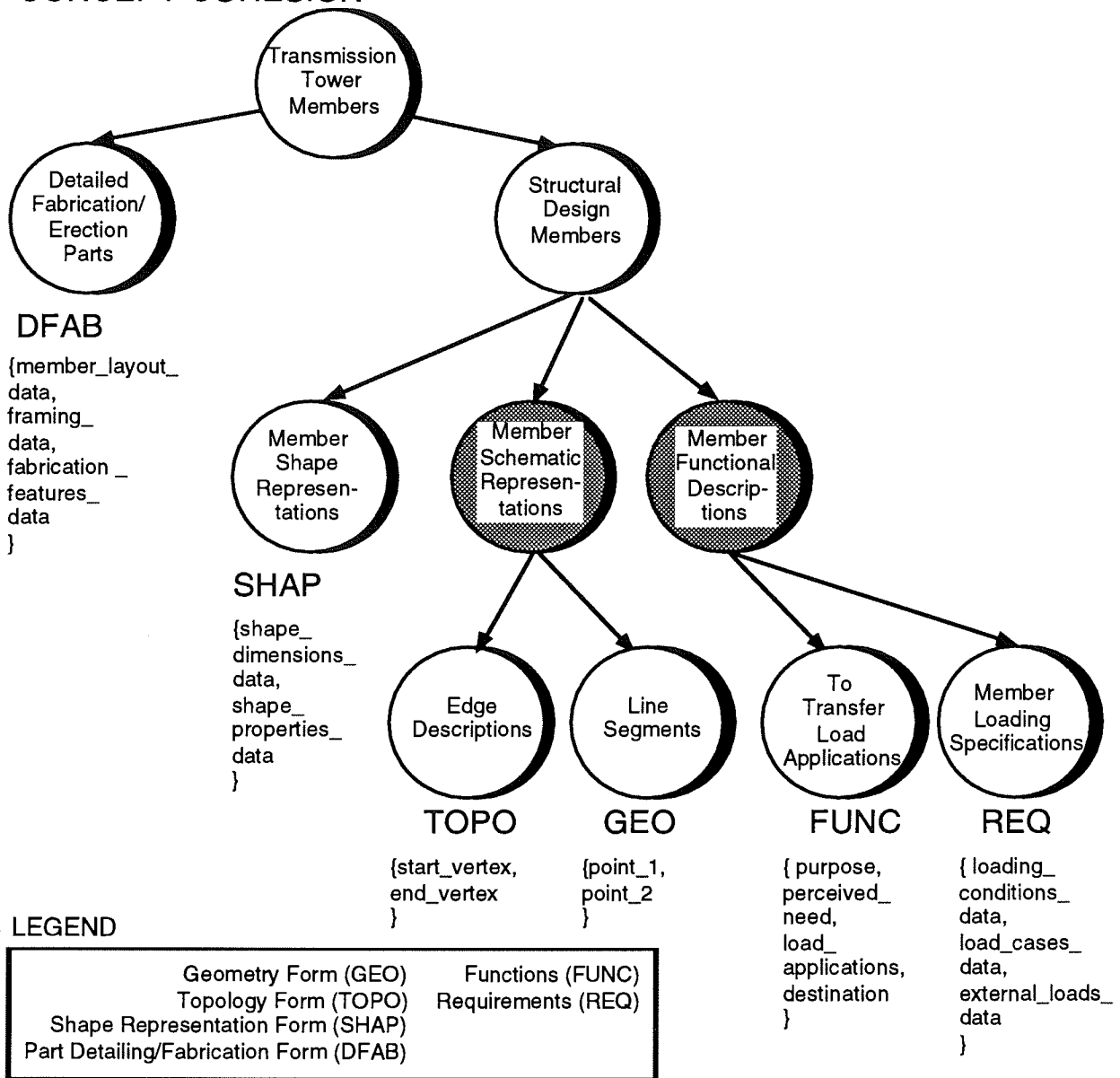


**FIGURE 5.5: Conceptual Categories Assigned to the Leaves of the Tree during the ASSIGN Operation in Step 2.**



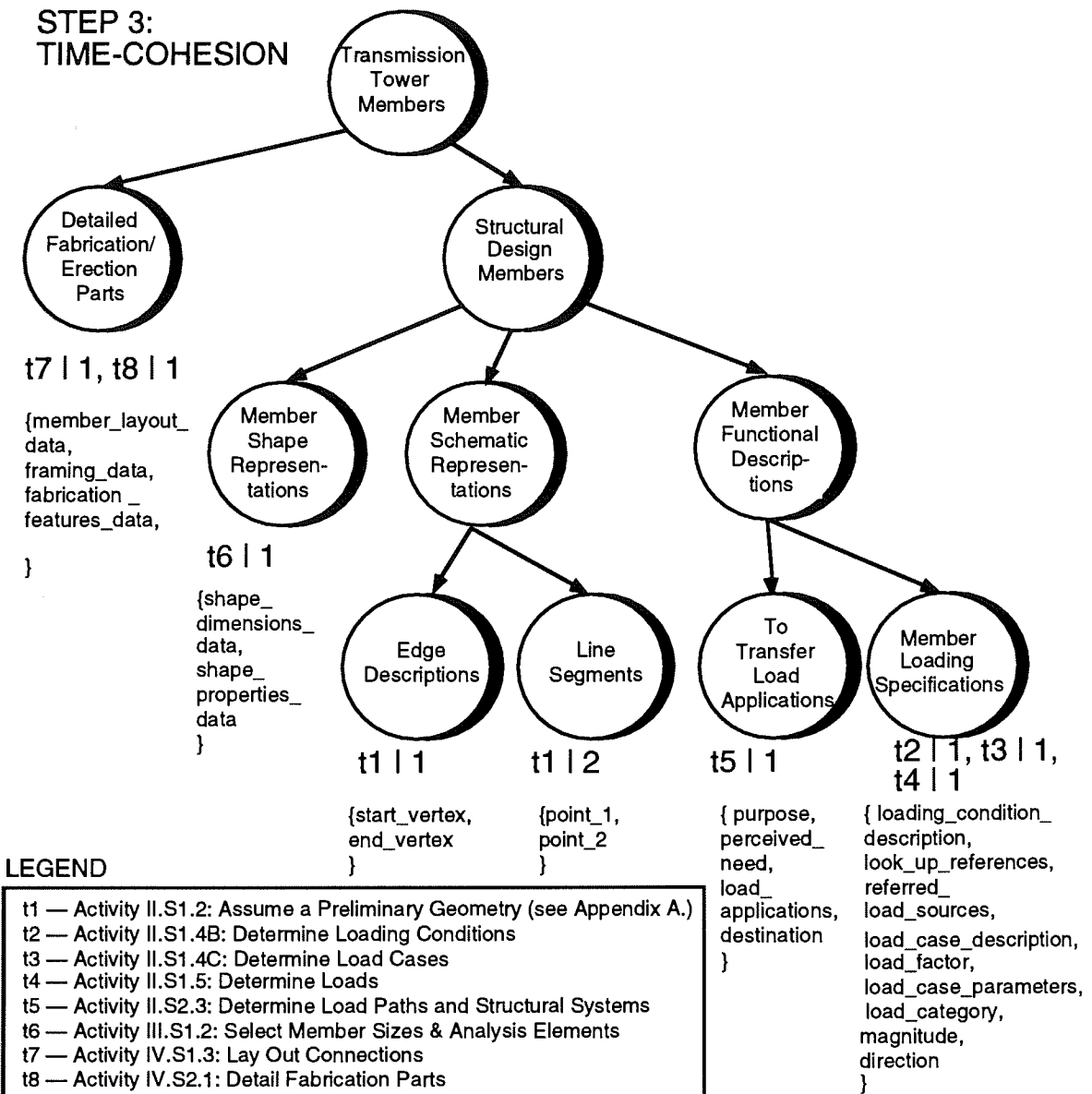
Figure 5.6 illustrates the tree after the DECOMPOSE operation based on the concept-cohesive criterion.

**STEP 2:  
CONCEPT-COHESION**



**FIGURE 5.6: Decomposition Tree at the End of Step 2 Considering Concept-Cohesion.** The shaded vertices are the non-concept-cohesive ones that get decomposed.

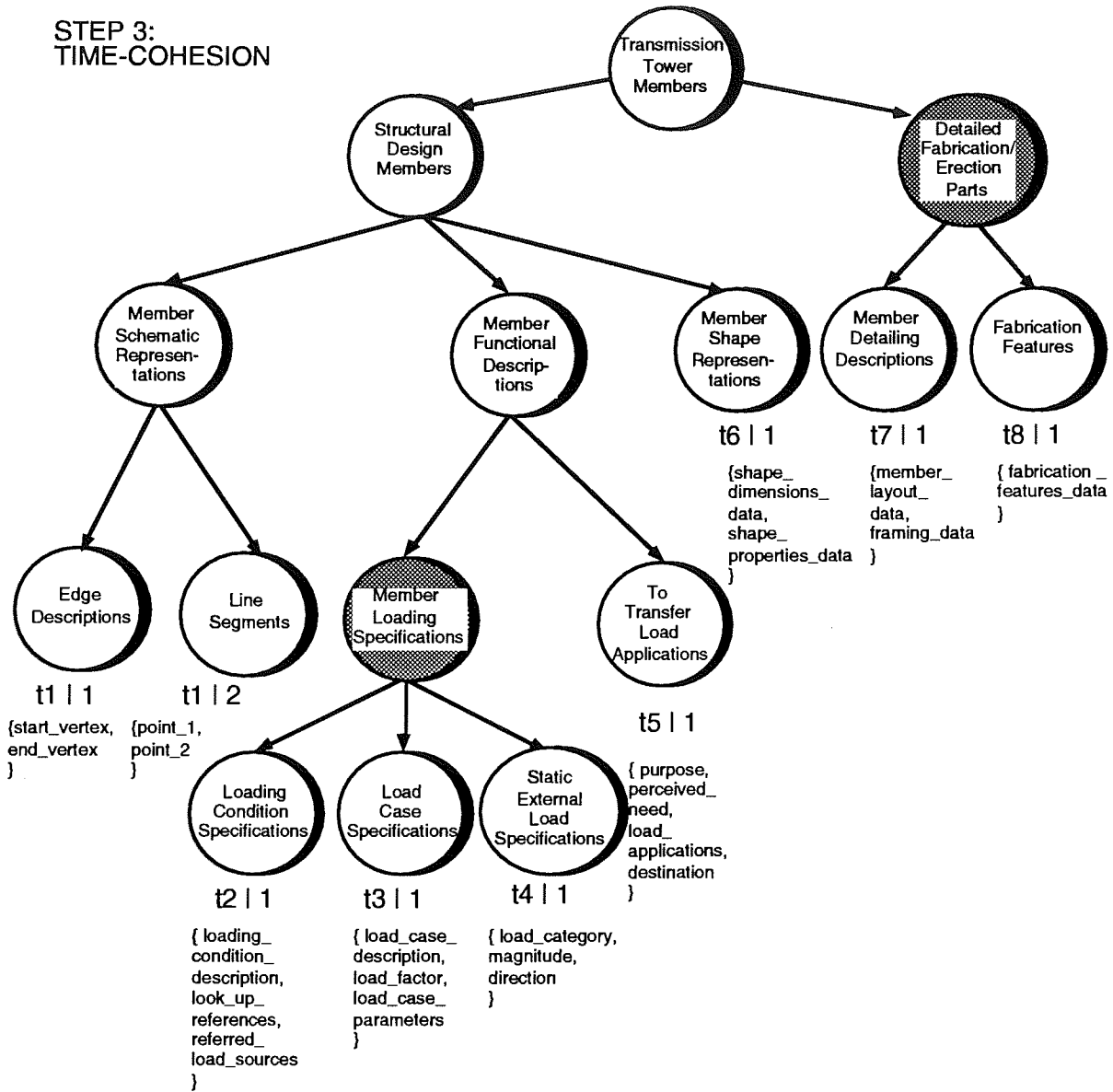
**Step 3: Considering Time-Cohesion** This step carries out the same successive operations as the previous two steps. Figure 5.7 shows the logical times assigned to the leaves of the decomposition tree determined during the ASSIGN operation. These logical times refer to tower engineering activities as shown in graphical functional schemata in Appendix A. Again, new data items shown in the extension of the leaves were introduced as a result of the earlier ELABORATE operation in this step.



**FIGURE 5.7: Logical Times Assigned to the Leaves of the Tree during the ASSIGN Operation in Step 3.**

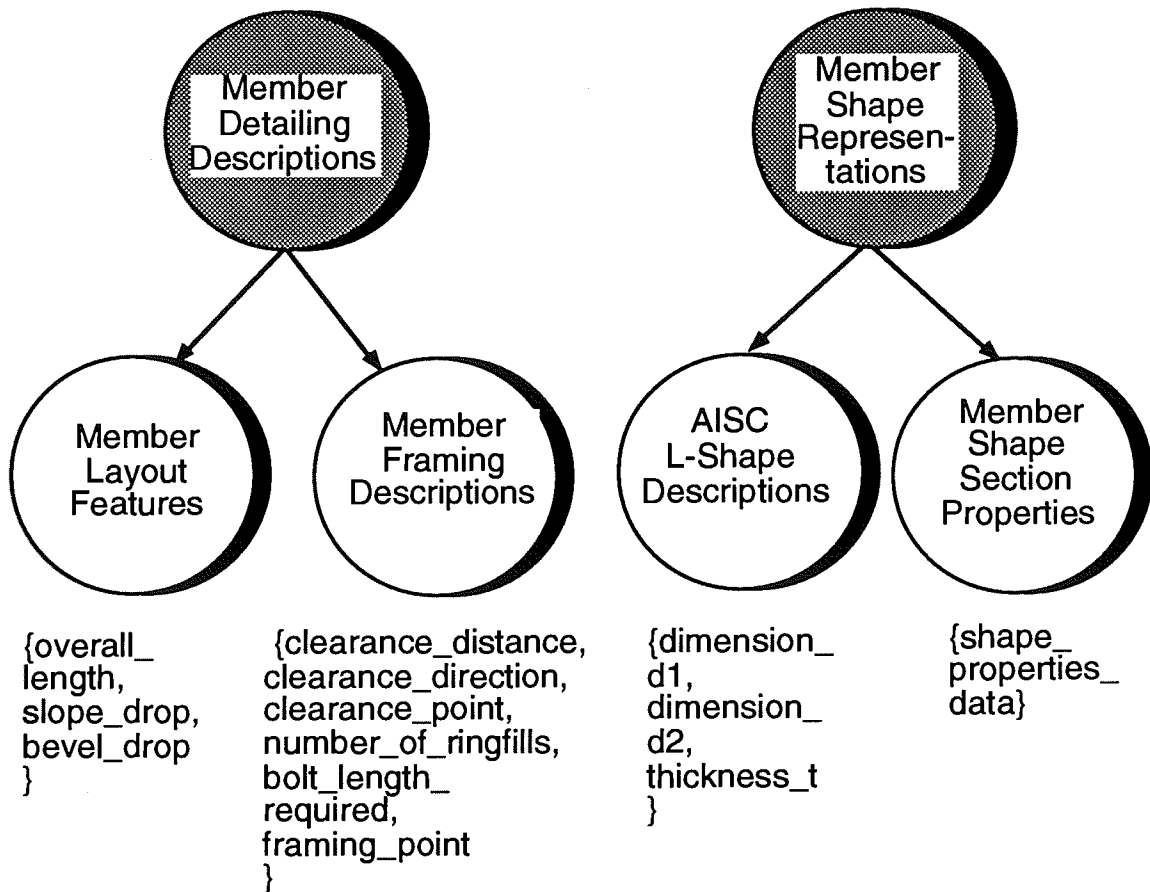
Figure 5.8 shows the decomposition tree resulting from this step. This tree is also time-ordered.

STEP 3:  
TIME-COHESION



**FIGURE 5.8: Time-Ordered Decomposition Tree at the End of Step 3 Considering Time-Cohesion** The shaded vertices are the non-time-cohesive ones that get decomposed.

**Step 4: Considering Source-Cohesion** Due to space constraints, Figure 5.9 shows only the two non-source-cohesive vertices that are decomposed in Step 4. Both vertices contain data items that are derived from other data items in the same vertex. For instance, shape section properties are derived from shape dimensions. New data items appear due to the ELABORATE operation carried out in this step.

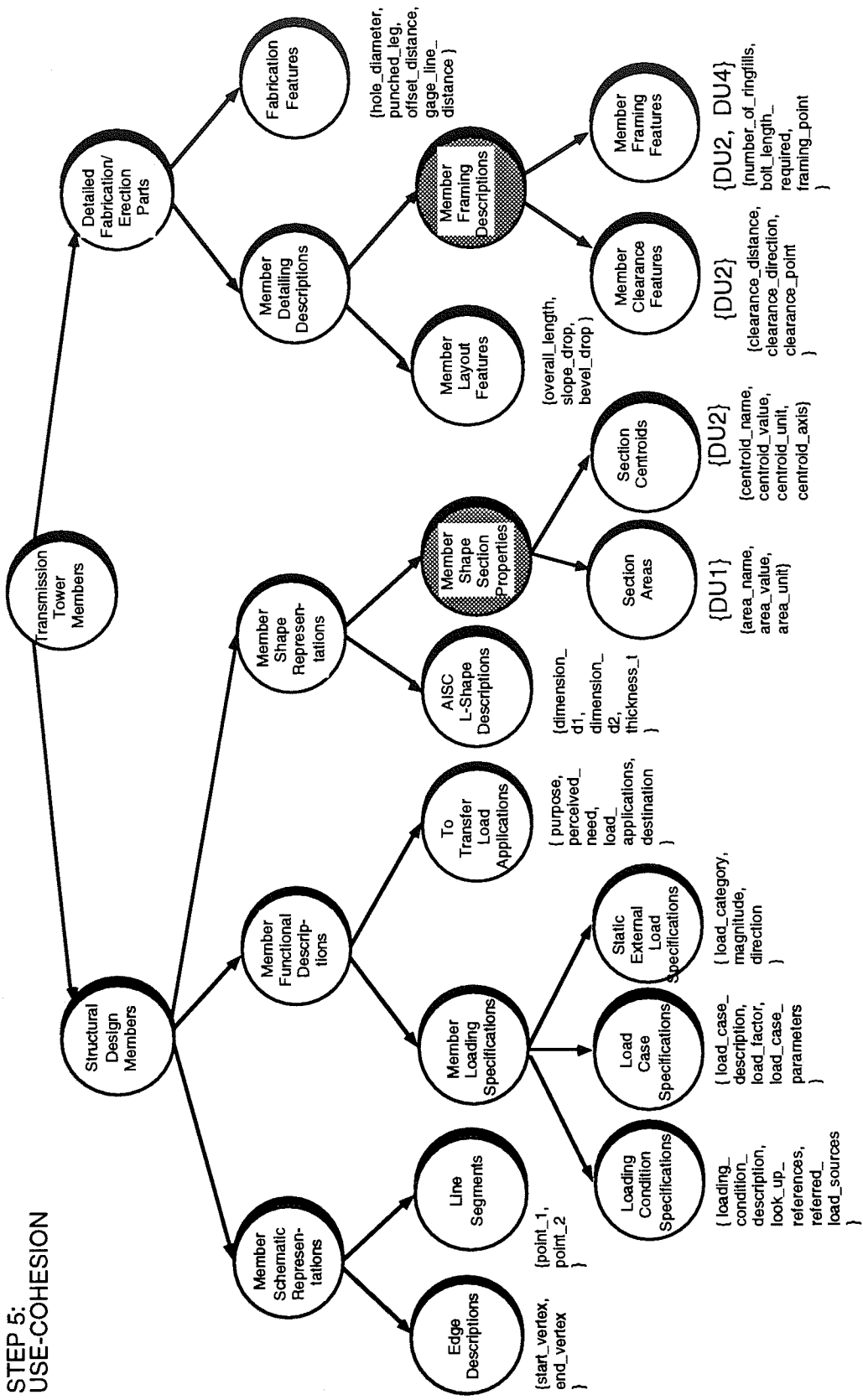


**FIGURE 5.9: Vertices Decomposed at the End of Step 4 Considering Source-Cohesion.** The shaded vertices are the non-source-cohesive ones that get decomposed.

**Step 5: Considering Use-Cohesion** To keep the example simple, let us consider four selected primary data uses that involve the following activities: (1) the structural engineer doing structural analysis of the tower (Activity III.S1.4), (2) the structure detailer laying out connections (Activity IV.S1.3), (3) the fabricator fabricating the tower parts (Activity V.F1.4), and (4) the construction crews assembling the tower systems on the ground (Activity V.C3.1). If these primary data uses are considered, the “Member Framing Descriptions” vertex will be decomposed since not all of its data items are used in the second and fourth primary data uses. Figure 5.10 shows the final tree that results from this decomposition.

### **5.5 DEAL as a Medium for Mediating Data Representations**

One important value of DEAL is that it provides a medium with which modelers and domain experts can “mediate” data representations. The notion of “mediating representations” is explained in [Bradshaw 92a]. In fact, modelers and domain experts can use DEAL to define a conceptual schema collaboratively without dealing with low-level implementation codes. With DEAL, the modeler can gradually elicit knowledge about the data used in the domain from the domain expert. In return, the modeler can incrementally show the domain expert what the representations of that data will look like. Using a CASE tool automating DEAL, the modeler can quickly develop a prototype of these representations, and the domain expert can instantly review them for correctness and accuracy. This type of communication is vital to the development of a schema that will be used by domain experts. A similar CASE tool that is enhanced with a knowledge-assisted, interactive user interface could be used directly by domain experts. In that case, the expert could create his or her own data representations with the help of this CASE tool. Different experts could use this tool to explore different data representations and can discuss among themselves about the sharability of those representations. Finally, with the criteria of cohesion and reusability, DEAL provides a basis for explaining and justifying a database schema design.



**FIGURE 5.10: Decomposition Tree at the End of Step 5 Considering Use-Cohesion.** The shaded vertices are the non-use-cohesive ones that get decomposed. DU1 stands for the second primary data use, DU2 stands for the second primary data use, and so on.

## **5.6 Chapter Summary**

---

DEAL is a method that aids the modeler in the conceptual modeling of a given facility engineering domain as mentioned in Chapter 1. DEAL's key idea is the decomposition of domain entities using the criteria of cohesion and reusability to obtain primitive entities. DEAL provides the terms and concepts, assumptions, graphical representation, procedure, operations, and rules.

In the P-C Approach, the domain entities analysis using DEAL follows the preliminary domain study and functional analysis phases and makes use of the results from those phases. Using the basic DEAL-1 version of the procedure, the modeler can decompose domain entities by considering how their data items can be accessed, to which conceptual categories they relate, at which logical time they are instantiated, from which source they are derived, and how they are used in activities of the engineering process. These five dimensions (i.e., access-cohesion, concept-cohesion, time-cohesion, source-cohesion, and use-cohesion) indicate how well the data items of an entity relate to one another. Using the improved version, DEAL-2, the modeler can take advantage of previously analyzed domain entities and can reuse primitive entities identified from those domain entities. Reusability measures the extent to which an entity can be reused (i.e. used without modifications) in the description of other domain entities. The P-C Approach defines five levels of reusability: domain-entity-type reusability, domain reusability, industry reusability, industry-type reusability, and universe reusability. In both versions, a CASE tool automating the procedure is needed to assist the modeler in analyzing highly data-intensive domain entities in real life.

The conceptual primitive entities resulting from the domain entities analysis are designed into logical object classes of the subsequent design of the domain primitive schema in Phase 4. To aid the modeler with this design, the P-C Approach provides an object-oriented data model, the Primitive-Composite (P-C) Data Model, and an accompanying method, the P-C Data Modeling Method, for using this model. The next chapter describes the domain schema design using both the model and method.

Finally, DEAL provides a medium with which a modeler can gradually elicit knowledge about the data used in the domain from a domain expert. In return, the modeler can incrementally show the domain expert what the representations of that data will look like.

# Chapter 6

## Domain Schema Design Using the P-C Data Model and Method

### **Chapter Abstract:**

In the P-C Approach, the design of a domain primitive schema involves refining those primitive entities identified in the preceding domain entities analysis, transforming them into primitive classes, and building primitive characterization hierarchies with these classes. To aid the modeler with this design, the P-C Approach provides an object-oriented data model, the Primitive-Composite (P-C) Data Model, and an accompanying method, the P-C Data Modeling Method, for using this model. The model includes concepts such as primitive and composite classes and instances, and several relationship types such as generalization, instantiation, aggregation, association, and derivation. The method provides the steps for the design of a domain primitive schema and a composite schema based on the concepts of the model. In addition, the method provides rules and guidelines for the design of domain primitive schemata or composite schemata. This chapter describes both the model and method in detail.

### **Organization:**

#### *6.1 Introduction*

#### *6.2 The P-C Data Model*

##### *6.2.1 Building Blocks of the Model*

##### *6.2.2 Direct Extensions to Object-Oriented Concepts*

##### *6.2.3 Relationships and Relationship Types*

#### *6.3 The P-C Data Modeling Method*

##### *6.3.1 Overview of the Method*

##### *6.3.2 Entity Dependencies and Dependency Types*

##### *6.3.3 Designing a Domain Primitive Schema*

##### *6.3.4 Designing a Composite Schema*

#### *6.4 Chapter Summary*



## 6.1 Introduction

The object-oriented paradigm provides useful concepts and techniques for representing entities in the real world. However, it leaves up to the modeler the task of deciding how to represent the problem domain in terms of object classes; how to organize object classes into class hierarchies; how to determine the attributes, number of attributes, and methods of each class; and so on. While some rules for making these decisions are imposed by the nature of the modeling problem at hand, others are dictated by the available constructs in the programming language or system in use. In any case, additional requirements and methods that can assist the modeler in meeting these requirements and thus insuring good design are clearly needed. Without them, the resulting object schema may be poorly defined and, consequently, difficult to maintain and upgrade.

In the P-C Approach, the domain primitive schema includes primitive classes that come from primitive entities identified in the domain entity analysis. Therefore, the design of this schema is optimized using the criteria of cohesion and reusability: Each primitive class is a module of attributes designed to have maximum cohesion and reusability. Moreover, primitive classes are organized into separate class hierarchies called "primitive characterization hierarchies." Each hierarchy involves a single concept about form, function, or behavior. This separation yields clean and modular data representations for describing facility design objects. Users can combine primitive classes from these hierarchies to customize composite classes representing their own views about the facility design objects. In addition, primitive classes can be added incrementally to the schema and, as a result, more composite classes can be defined from primitive classes, both old and new. This graceful extension of the schema can accommodate evolving life-cycle phases.

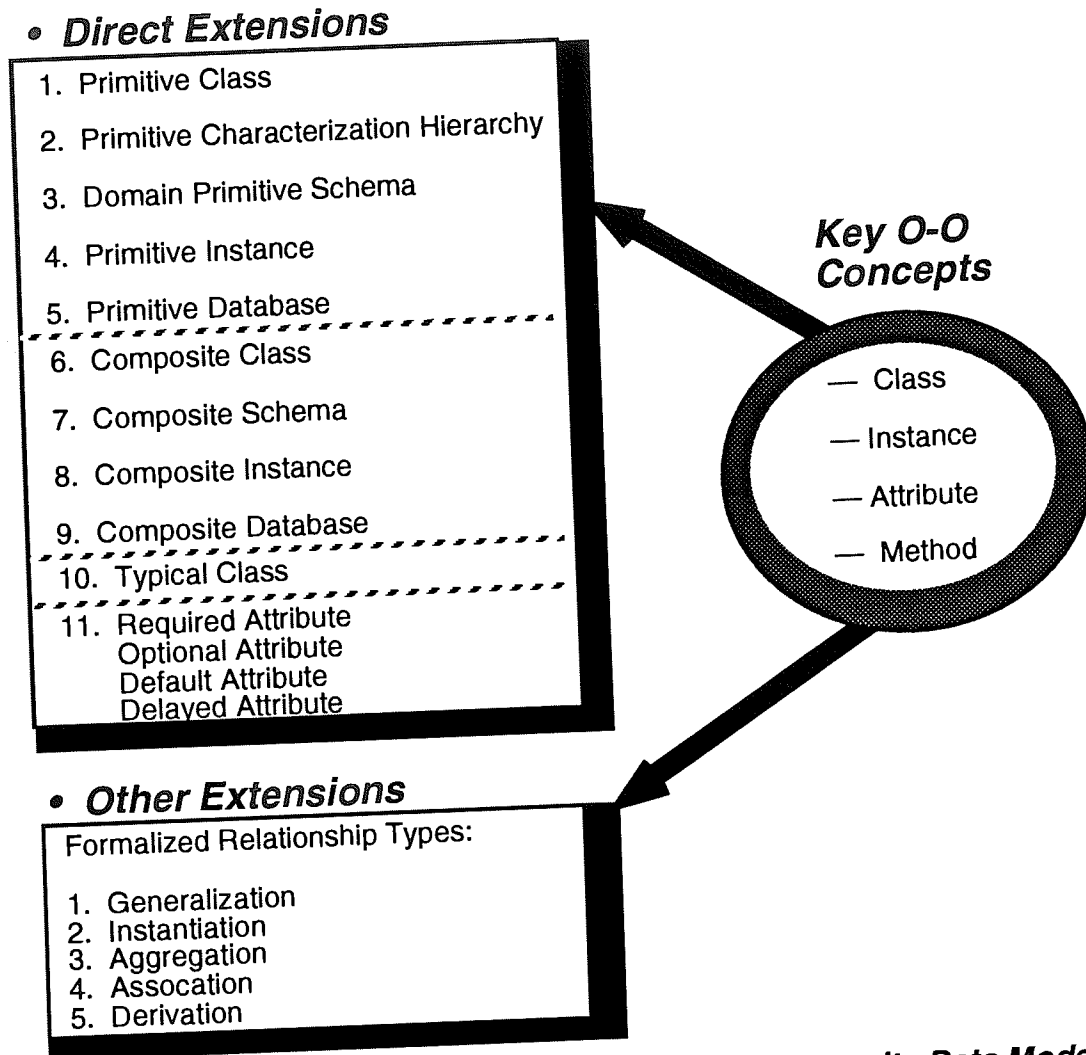
To aid the modeler in designing the domain primitive schema, the P-C Approach provides an object-oriented data model, the Primitive-Composite (P-C) Data Model, and an accompanying method for using this model. The model includes concepts (such as primitive and composite classes and instances) and several relationship types (such as generalization, aggregation, and association). The method provides the steps for the design of a domain primitive schema and a composite schema based on the concepts of the model. In addition, the method provides detailed rules and guidelines for the design of domain primitive schemata or composite schemata. Both the model and method were applied to the domain of transmission towers, the result of which will be presented in the next chapter.

## 6.2 The P-C Data Model

### 6.2.1 Building Blocks of the Model

The key object-oriented concepts (i.e., classes, attributes, methods, and instances) serve as the core of the P-C Data Model. (Before continuing, the reader may wish to review Section 2.1.3.2 of Chapter 2, which explains these concepts.) As Figure 6.1 shows, this model incorporates two sets of extensions to those concepts:

- The first set includes direct extensions to the key object-oriented concepts such as "primitive classes," "primitive characterization hierarchies," "composite classes," "primitive instances," "typical class," etc. This set also includes concepts that support data modeling and data exchange using the P-C Approach: "domain primitive schema," "primitive database," "composite schema," and "composite database."



**FIGURE 6.1: Building Blocks of the Primitive-Composite Data Model.**

- The second set of extensions consists of a number of formalized relationship types (i.e., generalization, instantiation, aggregation, association, and derivation) that are based on the key abstraction methods (or modeling techniques) used in the model.

### 6.2.2 Direct Extensions to Object-Oriented Concepts

The concepts presented below constitute the first set of extensions to the key object-oriented concepts in the P-C Data Model.

**1. Primitive Class**—A primitive class is a module of attributes that is designed to have maximum cohesion and reusability. Each primitive class belongs to a primitive characterization hierarchy of a domain primitive schema.

*Explanation:* Primitive classes share the following design properties: Each primitive entity identified from the preceding domain entities analysis is refined and transformed into a primitive class during the domain schema design. Therefore, cohesion and reusability determine which attributes belong to the primitive class. Cohesion measures how closely the attributes relate to one another, while reusability measures the extent to which the object class as a whole can later be reused. Each primitive class is designed to

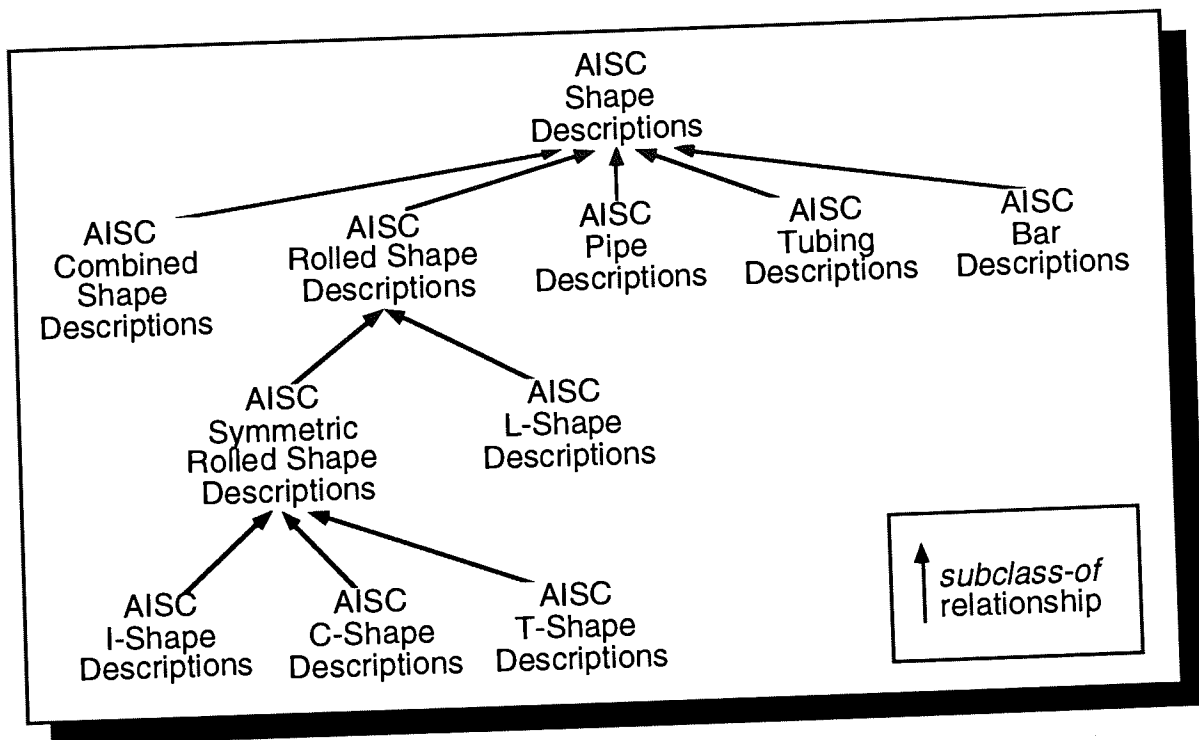
be as cohesive and reusable as possible. In addition, each primitive class is rooted in only one primitive characterization hierarchy in a domain primitive schema. This property ensures the conceptual simplicity of primitive classes.

*Example:* The entity, "AISC L-Shape Descriptions," introduced in the previous chapter is designed as a primitive class. The class definition includes five attributes: "shape size designation," "dimension unit," "dimension d1," "dimension d2," and "thickness t." The last three attributes come directly from the domain entity analysis, whereas the first two attributes are part of the refinement that completes the entity description.

**2. Primitive Characterization Hierarchy**—A primitive characterization hierarchy is a hierarchy of primitive classes that represent increasing degrees of specialization of a single concept pertinent to one conceptual category.

*Explanation:* The modeler can design a primitive class hierarchy that moves from the more general classes to the more specialized ones, as long as the hierarchy depicts one concept. This design property eliminates the problem of non-homogeneous class hierarchies described in Chapter 2 and produces clean, highly modular data representations that can later be assembled to describe complex facility design objects.

*Example:* Figure 6.2 shows a sample primitive characterization hierarchy in the domain primitive schema for transmission towers, which involves the concept of AISC-standard shape descriptions.



**FIGURE 6.2: A Sample Primitive Characterization Hierarchy.**

**3. Domain Primitive Schema**—A domain primitive schema (or “primitive schema”) is a set of primitive characterization hierarchies that define the basic concepts used by experts in a domain. (This schema is the end result of the modeler’s work using the P-C Approach).

*Example:* In Chapter 7, the domain primitive schema for transmission towers includes several primitive characterization hierarchies that define the basic concepts of the domain. These concepts belong to different conceptual categories concerning form, function, and behavior in that domain.

**4. Primitive Instance** —A primitive instance is an instance of a primitive class.

**5. Primitive Database**—A primitive database contains instances of primitive classes from a domain primitive schema. Those primitive classes are necessary to describe a facility design object or a set of design objects.

**6. Composite Class** —A composite class is a subclass, aggregation, or association of several primitive classes (or even of other composite classes) that a user<sup>1</sup> of the domain primitive schema customizes. To customize a composite class, the user first selects primitive classes from different primitive characterization hierarchies and then determines the relationships (of generalization, aggregation, and association types) between those primitive classes and the composite class under construction.

*Explanation:* A composite class provides a construct for formalizing an abstraction of several concepts that a single primitive class cannot represent. Composite classes exhibit the following design properties, which distinguish them from primitive classes: First, users of the domain primitive schema create these classes to represent their own complex abstractions about the facility design objects. As a result, composite classes typically contain a large number of attributes depicting various aspects of the design objects. Composite classes and class hierarchies need not be predefined because users can always assemble their own at any time. Composite classes typically are not access-cohesive, concept-cohesive, time-cohesive or source-cohesive.

*Example:* Figure 6.3 shows the customization of a composite class called “Transmission Tower Legs As Analyzed.” This composite class represents one specific abstraction of the transmission tower legs when they are subjected to structural analysis during the Tower Structural Detailed Design phase.

**7. Composite Schema**—A composite schema includes a subset of the primitive schema and a set of composite classes that represent a user’s view of the underlying facility data and therefore, suits the particular needs of that user.

**8. Composite Instance**—A composite instance is an instance of a composite class.

**9. Composite Database**—A composite database is a database that contains instances of composite and primitive classes from a composite schema.

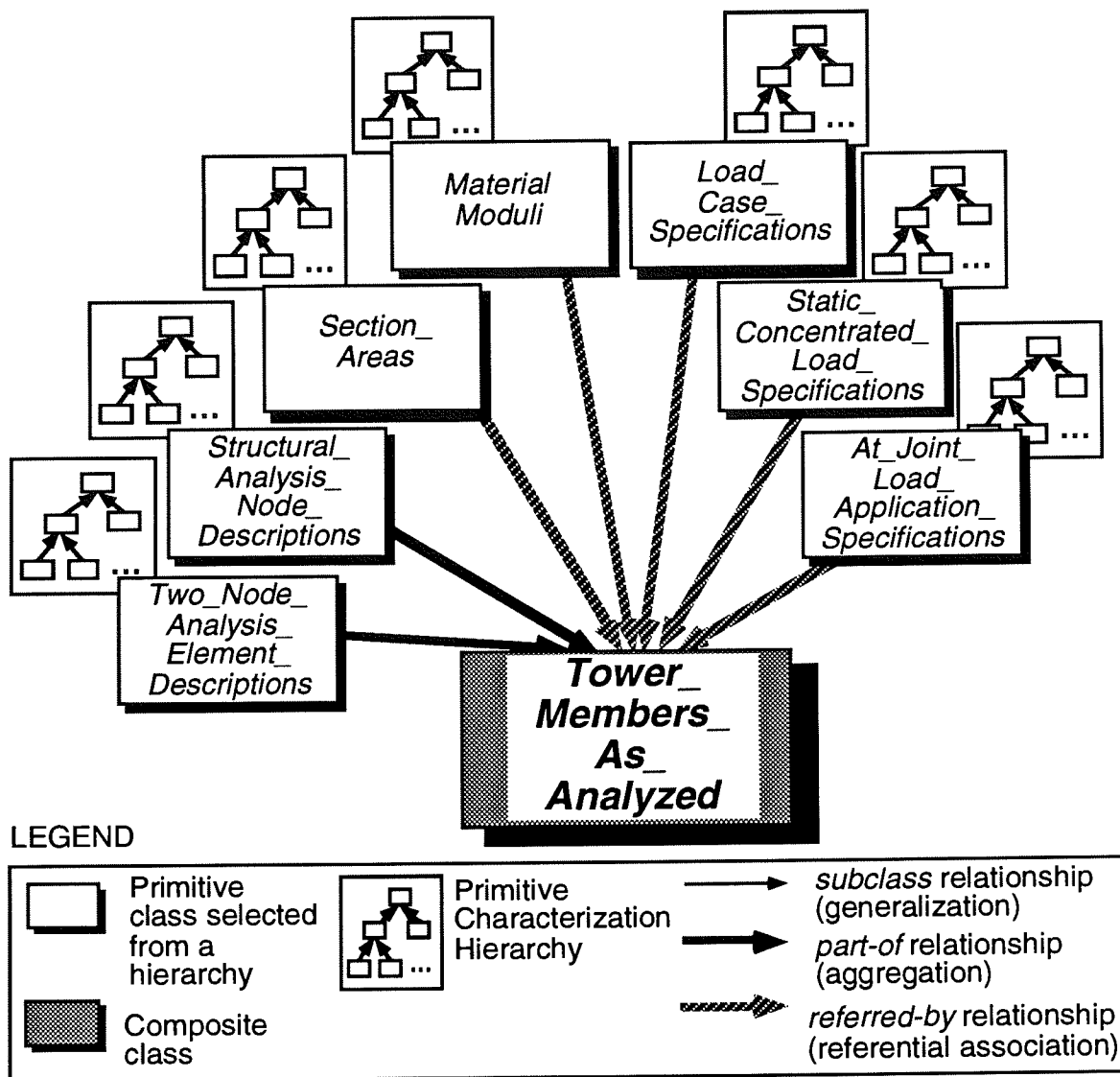
**10. Typical Class**—A class whose instances share some specific common attribute values. A typical class does not add new attributes to those inherited from its superclass; however, it must have at least one default attribute whose value is predefined. Either a primitive class or a composite class can be a typical class.

---

<sup>1</sup> A user here means anyone who uses a domain primitive schema to customize a view, to build a primitive or composite database, to formulate a query on such a database, or to build a new computer application sharing the data representations of the primitive schema. On the other hand, a modeler strictly means the person who designs the domain primitive schema.

*Explanation:* Typical classes provide useful and efficient constructs for representing real-life facility design objects. With typical classes, the modeler can create class templates for prototypical facility design such as the floor layout design described in the following example. (Typical classes correspond to the typical entity definitions in the SSFDM model [Lavakare 89] and to the “specific level” of Product Definition Unit in the GARM model [Gielingh 88].) Default attributes are explained next.

*Example:* In a building floor layout design, the same prototypical design is used for all floor levels. In this case, typical classes can capture the data common to the floor levels (e.g., outer beam sizes, girder lengths, column locations, etc.). Instances of these classes then include the detailed data particular to each floor level.



**FIGURE 6.3: Customization of a Composite Class.**

**11. Required, Optional, Unique, Default, and Delayed Attributes**—A “required attribute” must always have a known value that the user specifies. By contrast, an “optional attribute” can have an unknown value at any time. These two types of attributes are mutually exclusive. A “unique attribute” has a distinct value for each instance. A “default attribute” has a predefined value used for every instance where the user does not specify a value. A “delayed attribute” of a new instance has an unknown value until the end of the delay time specified by the user.

*Explanation:* All of these attribute types must be considered in the design of classes, both primitive and composite. They directly affect the implementation of the classes. Data items of primitive entities from the domain entities analysis are converted into required attributes of primitive classes during the domain schema design. Optional attributes enable the modeler to add desirable but non-essential information to a class definition. Unique attributes apply mainly to user-defined object identifiers. Delayed attributes are primarily used for relationship attributes of the derivation type, which will be explained later. Delayed attributes provide users with the flexibility of delaying the creation of derived instances.

*Example:* The “AISC L-Shape Descriptions” primitive class defined earlier has a required and unique “shape size designation” attribute, three required attributes (i.e., “dimension d1,” “dimension d2,” and “thickness t”), a “dimension unit” attribute defaulted to “inch,” and a delayed “derived section properties” attribute. Here, the user can delay the computation of the section properties until a later time.

*Discussion:* The modeler must declare the attribute types for each attribute and the default value in the case of a default attribute. The user must specify the delay type for each delayed attribute of a new instance. The Structural Data Model [Wiederhold 80] defines three possible delay types: delaying until the execution of a certain transaction, delaying until a specific time, and delaying until the end of a specific interval. Although optional attributes are desirable in some cases, they adversely affect the time-cohesion and use-cohesion of a class. Therefore, rules and guidelines of the modeling method closely monitor and direct the use of this attribute type.

### 6.2.3 Relationships and Relationship Types

Relationships between objects play an important role in the object-oriented paradigm. They augment the description of object states and establish direct links between objects [Abdalla 89]. However, the object-oriented paradigm supports relationships only in terms of logical pointers from one object to others. These pointers do not capture semantics nor do they enforce any integrity constraints. Formalized relationship types are an integral part of the P-C Data Model since they provide the mechanism for explicitly representing the way in which facility design objects are interrelated, for capturing the semantics of these relationships (thereby reducing the complexity of embedding such semantics elsewhere), and for relating primitive classes and instances to composite classes and instances.

#### 6.2.3.1 Definition of Relationships

**Relationship**—A relationship represents an explicit directed link between two or more classes, between two or more instances, or between two or more classes and instances.

*Explanation:* A relationship can also be seen as a logical pointer with built-in semantics. The elements at the origin of the relationship link are called “source.” Those at the other end are called “destination.”

*Example:* "leg23" is *part of* "panel2," and "leg23" is *referred to* "A36 steel material yield strength," where *part of* and *referred to* are relationships between the indicated instances.

**Inverse Relationship**—Relationships are usually defined in pairs: A relationship represents a link in one direction, and its inverse relationship describes a link in the opposite direction.

*Example:* "leg23" is *part of* "panel2," and "panel2" has *subpart* "leg23."

**Relationship Attribute**—Within a class definition, a relationship attribute is a place-holder for a relationship to another class. The specification of the relationship attribute declares the expected destination class.

**Cardinality**—The “cardinality” of a relationship (e.g., one-to-one, many-to-many, many-to-one, etc.) specifies the expected number of elements (classes or instances) at each end of the relationship.

### 6.2.3.2 Types of Relationships

The P-C Data Model supports five relationship types that are based on the key abstraction methods used in the model. They are: “generalization,” “instantiation,” “aggregation,” “referential association” and “derivation.” These relationship types are consistent with those found in the literature on data modeling.

**Generalization**—The abstraction method of “generalization” [Smith 77] is used to relate a subclass to a superclass. Generalization makes use of inheritance. Relationships of this type are called *subclass of* and have inverse relationships called *subclass*. A class can have zero or more superclasses. A class can also have zero or more subclasses.

**Instantiation**—The abstraction method of “instantiation” enables us to define instances from a class or a typical class. Relationships of this type are called *instance* and have inverse relationships called *instance of*. A class can have zero or more instances; however, each instance can be instantiated from only one class.

**Aggregation**—The abstraction method of “aggregation” [Smith 77] is used to construct an instance representing a complex design object from instances representing components of that object. Relationships of this type are called *part of* and have inverse relationships called *subpart*. For convenience, we call the source of a *part of* relationship “component instance” and its destination “aggregate instance.” An aggregate instance can have several component instances. A component instance can be part of one or more aggregate instances.

**Referential Association**—The abstraction method of “referential association” allows us to define a reference from one instance to another. Relationships of this type are called *referred to* and have inverse relationships called *referred by*. For convenience, we call the source of a *referred to* relationship “referring instance” and its destination “referred instance.” A referring instance can have zero or more referred instances. A referred instance can be *referred by* zero or more referring instances.

**Derivation**—The abstraction method of “derivation” enables us to compute or derive the attribute values of an instance from those of another instance. Relationships of this type are called *derived from* and have inverse relationships called *derives*. For convenience, we call the source of a *derived from* relationship “derived instance” and its destination “deriving instance.” An instance can be derived from only one instance of another class. Similarly, a deriving instance can derive only one instance of another class. (However, an instance can derive one or more instances of different classes.)

### **6.2.3.3 Semantics of Relationships**

A relationship of a type has certain implications for the existence and integrity of its source and destination elements. The “semantic rules” for the relationship type formalize these implications. These rules govern the creation, storage, modification, and deletion of the elements linked by the relationships. These rules are important to the definition of the P-C Data Model, as well as to the implementation of a primitive or composite schema and to the semantic integrity maintenance of a primitive or composite database.

#### ***Semantic Rules of the Generalization Relationship Type:***

1. A class cannot be deleted from the schema unless it has no subclasses.
2. A class definition in the schema cannot be modified unless it has no subclasses or its subclasses have no instances.

#### ***Semantic Rules of the Instantiation Relationship Type:***

1. The attribute values of an instance can be modified, but the behavior defined by the class methods cannot.
2. An instance with no relationships to other instances can always be deleted from the database. An instance with relationships can be deleted only if the semantic rules of those relationships allow the deletion.
3. A class can be deleted from the schema only if all its instances are first deleted from the database. (This rule goes together with the first rule of the generalization relationship type.)
4. A class definition in the schema cannot be modified unless it has no instances. (This rule goes together with the second rule of the generalization relationship type.)

#### ***Semantic Rules of the Aggregation Relationship Type:***

1. An aggregate instance can be stored in the database with zero or more component instances, but a component instance cannot be stored unless it is owned by an aggregate instance.
2. The deletion of an aggregate instance from the database at any time triggers the deletion of all its component instances, providing that no other aggregate instance or instances jointly own these component instances. A component instance can be deleted only if it is no longer part of some aggregate instance or instances.
3. The modification of an aggregate instance does not affect its component instances, and vice versa.

#### ***Semantic Rules of the Association Relationship Type:***

1. A referred instance can be stored in the database independently of all other instances. A referring instance can be stored in the database with or without referred instances, depending on the type of its relationship attribute (i.e., required or optional).
2. The deletion of a referring instance from the database at any time does not affect the existence of all its referred instances. However, a referred instance can be deleted only if it is no longer referred by other instances.



3. The modification of a referring instance does not affect its referred instances, and vice versa.

***Semantic Rules of the Derivation Relationship Type:***

1. A deriving instance can be stored in the database with or without its derived instance (or instances of different classes) as long as the delay type is specified. A derived instance cannot be stored without a source instance.
2. The deletion of a deriving instance from the database at any time triggers the deletion of all its derived instances (since derived instances are meaningful only if their deriving instance exist). A derived instance can be deleted only if its deriving instance is first deleted.
3. The modification of a deriving instance requires the modification of the derived instances. This is the only condition under which a derived instance can be modified.

**6.2.3.4 Permissible Relationship Types Among Primitive and Composite Classes and Instances**

The permissible relationship types among primitive and composite classes and instances can be summarized as follows:

- The relationships between primitive classes of any two consecutive levels in a primitive characterization hierarchy are of the generalization type. *The definition of primitive classes can never use multiple inheritance.*
- The relationships between primitive classes from different primitive characterization hierarchies are of the aggregation, association, or derivation type.
- The relationships between primitive classes and composite classes and between two composite classes are of the generalization, aggregation or association type. The definition of composite classes can take advantage of multiple inheritance. (The derivation relationship type is inappropriate in relating a primitive class to a composite class.)
- The relationships between primitive instances and their primitive classes, or composite instances and their composite classes, are of the instantiation type.

---

**6.3 The P-C Data Modeling Method**

---

**6.3.1 Overview of the Method**

**6.3.1.1 Steps for the Design of a Domain Primitive Schema**

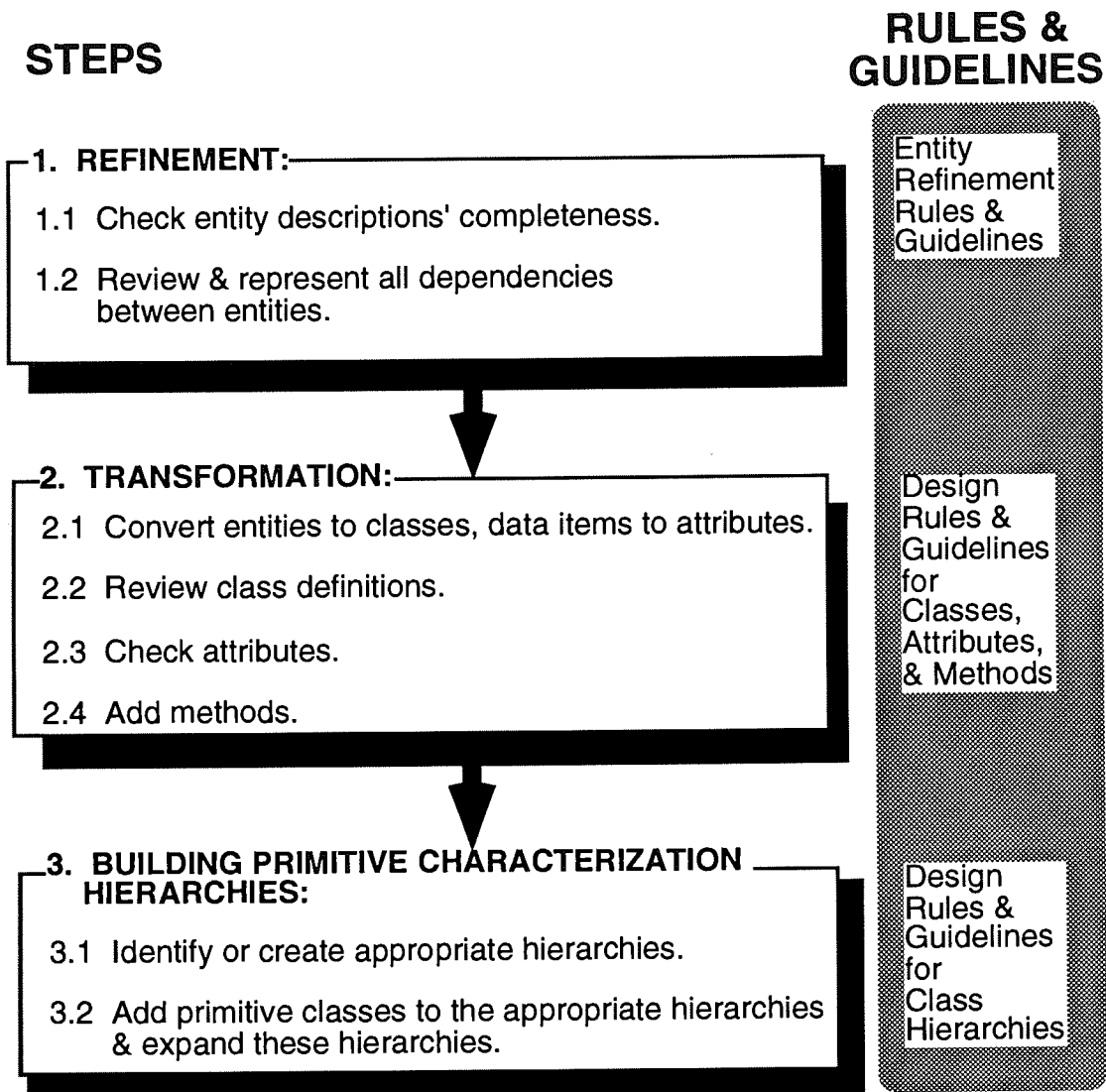
The P-C Data Modeling Method provides the steps for the design of a domain primitive schema (i.e., Phase 4 of the P-C Approach) based on the concepts of the P-C Data Model. Figure 6.4 shows three major steps for doing this:

- *Refining Primitive Entities:* This step involves adding to primitive entities any data items necessary to complete their description and representing the dependencies between these entities. The representation of entity dependencies is an important

part of the design. Section 6.3.2 will explain entity dependencies and discuss the different types.

- *Transforming Refined Primitive Entities into Primitive Classes:* This step involves converting entities into object classes, converting data items into attributes, and adding methods for computing derived attribute values and for setting and retrieving independent attribute values.
- *Building Primitive Characterization Hierarchies with the New Primitive Classes:* This step involves identifying the appropriate characterization hierarchies for the new primitive classes, creating new hierarchies if necessary, adding primitive classes into the appropriate hierarchies, and expanding the hierarchies.

Section 6.3.3 goes over these steps in more detail. Section 6.3.4 then shows the steps for the design of a composite schema, which are included in this modeling method.



**FIGURE 6.4: Overview of the Design of a Domain Primitive Schema.**

### 6.3.1.2 Rules and Guidelines of the Method

Figure 6.5 summarizes the rules and guidelines of the method, which can assist the modeler in designing domain primitive schemata and composite schemata. These rules and guidelines also ensure desirable qualities such as minimality, expressiveness, efficiency, and self-explanation of the resulting schema [Batini 92].

**Usage** Modelers should use these rules and guidelines when they design a domain primitive schema as described in the previous section, or a composite schema. Alternatively, experienced modelers can use these rules and guidelines as a check list to verify the qualities of the resulting schema after they have completed their design. *In either case, modelers should view rules as specifications that they must follow in order to achieve good results and also, save time and effort. They should consider guidelines to be potentially useful suggestions in addition to the rules.*

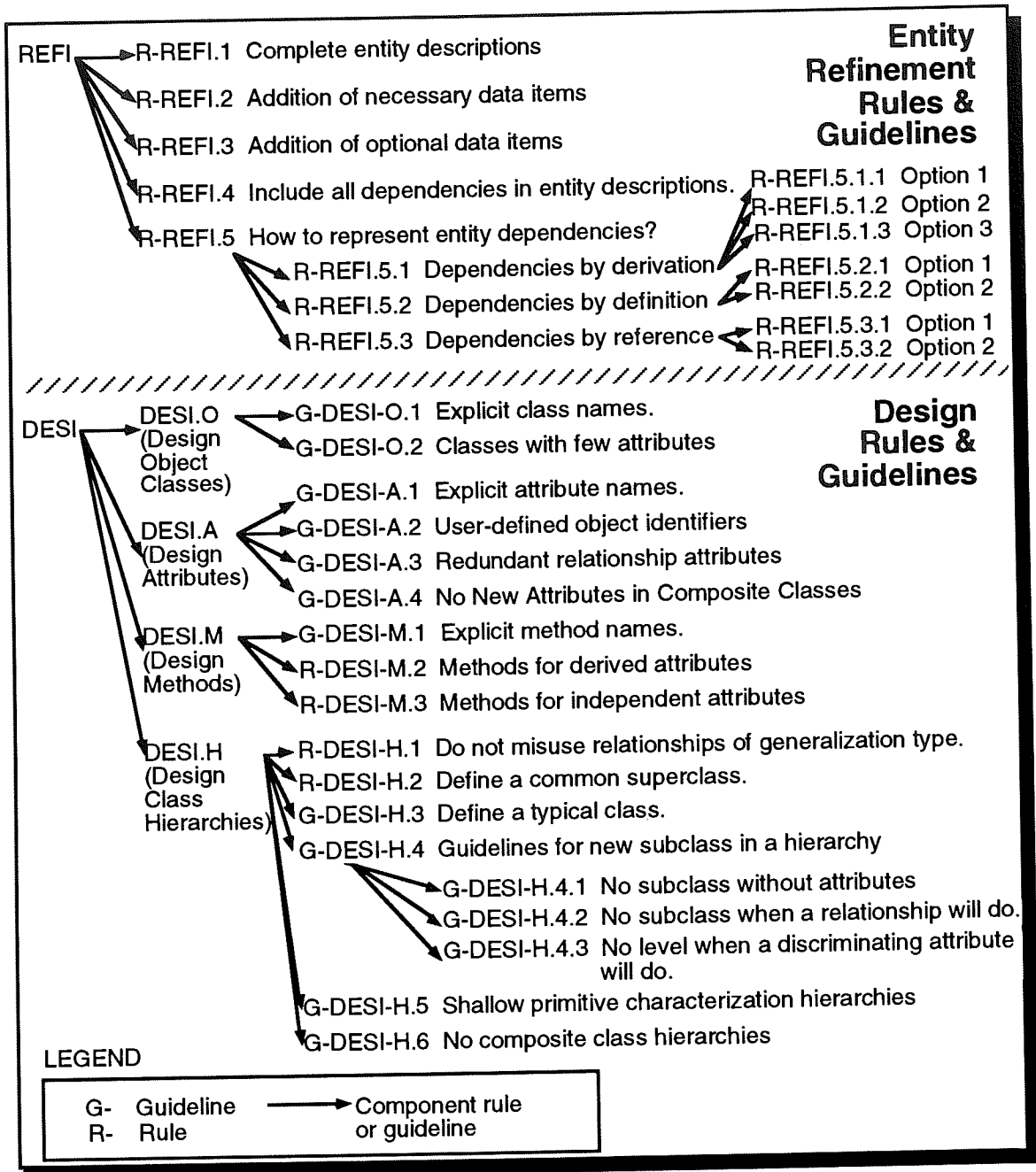
**Organization** The rules and guidelines are divided into two major groups: (1) those for the refinement of entities and (2) those for the design of object classes, attributes, methods, and class hierarchies. Within each group, rules and guidelines can have other component rules and guidelines. They are also listed in the order in which the modeler should consider them while designing a schema.

**Label of Rules and Guidelines** The following conventions are adopted here to label the rules and guidelines for later reference:

- R for rules and G for guidelines,
- REFI for entity refinement (i.e., the first category of rules and guidelines) and DESI for design (i.e., the second category), and
- O for the design of object classes, A for the design of class atttributes, M for the design of class methods, and H for the design of object class hierarchies.

Component rules and guidelines within each group are labeled using subscripts such as .1, .2, .3, etc.

**Full Documentation of Rules and Guidelines** A consistent format is adopted here to document the rules and guidelines fully. Whenever possible, documentation includes the following items: label and name (e.g., “R-REFI.1” and “Complete Entity Descriptions”), statement, explanation, example(s), exception(s), counterexample(s), discussion (i.e., applicability, advantages, tradeoffs, and other comments) of the rule or guideline; and references to other relevant rules and guidelines. Appendix C contains the full documentation of all rules and guidelines of the P-C Data Modeling Method.



**FIGURE 6.5: Summary of Rules and Guidelines of the P-C Data Modeling Method.** The names of the rules and guidelines are shortened here due to space constraints.

### 6.3.2 Entity Dependencies and Dependency Types

The domain entities analysis in Chapter 5 deals with decomposing large sets of data items describing domain entities into more manageable, cohesive, and reusable modules called “conceptual primitive entities.” The domain schema design in this chapter involves building an object-oriented schema from those primitive entities. Ideally, these cohesive and reusable primitive entities should also be self-descriptive and independent modules. In reality, they are not independent of one another. The following are examples of entities that depend on other entities: “Cartesian Vector Direction Cosines” on “Cartesian Vectors,” “Line Segments” on “Points,” “Loading Condition Specifications” on “Load Source Specifications,” and “Design Decision Descriptions” on “Design Alternative Descriptions.” Therefore, “entity dependencies” (or “entity coupling”) are another important dimension that must be considered in the schema design.

**Entity Dependencies** Entity dependencies are primarily concerned with how much must be known about other entities in the schema to describe an entity. An entity, X, is dependent on another entity, Y, if the description of X requires some description of Y. That degree of dependency is determined by the type of entity dependency, which will soon be explained. An entity dependency is not exactly the same as a relationship (defined earlier in Section 6.2.3.1). An entity dependency conceptualizes a way in which one entity is dependent upon another, whereas a relationship can represent that dependency using a logical pointer between two classes or instances. In fact, considering dependencies between entities in the domain schema design precedes transforming these entities into classes and defining relationships between those classes. We need to consider entity dependencies for two reasons:

1. *The notion of entity dependency is consistent with that of entity cohesion:* The entity cohesion’s evaluation in the domain entity analysis complements the entity dependencies’ consideration in the domain schema design in order to produce a modular schema design in the end. While the former measures the properties inside the entities, the latter examines the dependencies outside the entities.
2. *Entities and entity dependencies provide an implementation-independent level of abstraction above the logical level of classes and relationships:* This level of abstraction enables the modeler to think about how one entity is dependent on other entities in the schema. The modeler can then decide how to represent that dependency using relationships or other means at the next logical level. Not every dependency type must be represented using relationships. In fact, the rules and guidelines in Section 6.3.3 will show ways to represent entity dependencies without using relationships.

**Entity Dependency Types** Like cohesion and reusability, entity dependency can be operationalized by distinguishing the following three types:

**Entity Dependencies by Derivation**—An entity, X, is dependent by derivation on another entity, Y, if some data item values of every instance of X are derived from those of some instance of Y.

*Notation:* X |>by-derivation Y.

*Example:*

- “Cartesian Vector Direction Cosines” |>by-derivation “Cartesian Vectors”
- “Path Connectivities” |>by-derivation “Path Descriptions”
- “Load Case Specifications” |>by-derivation “Loading Condition Specifications”

***Entity Dependencies by Definition***—An entity, *X*, is dependent by definition on another entity, *Y*, if *X* is defined, according to our knowledge and experience, in terms of *Y*. In other words, an occurrence of *X* includes occurrences of *Y* as part of its description.

*Notation:*  $X \text{ |>by-definition } Y$ .

*Example:*

- “Line Segments” |>by-definition “Points”
- “Structural Analysis Node Descriptions” |>by-definition “Cartesian Coordinates”
- “Surfaces of Revolution” |>by-definition “Curves”

***Entity Dependencies by Reference***—An entity, *X*, is dependent by reference on another entity, *Y*, if *Y* contains further information pertinent to *X*.

*Notation:*  $X \text{ |>by-reference } Y$ .

*Example:*

- “Loading Condition Specifications” |>by-reference “Load Source Specifications,”
- “Design Decision Descriptions” |>by-reference “Design Alternative Descriptions”
- “Design Artifact Descriptions” |>by-reference “General Requirements”

### 6.3.3 Designing A Domain Primitive Schema

This section describes in detail the steps involved in the design of a domain primitive schema (i.e., Phase 4 of the P-C Approach). Due to space constraints, this section describes only briefly all the rules and guidelines used in the design. Each description includes a label (e.g., “R-REFI.1”), name (e.g., “Complete Entity Descriptions”), statement (shown in italics), explanation and example of the rule or guideline. Appendix C contains the full documentation of the rules and guidelines.

The design of a domain primitive schema involves:

**1. Refining Primitive Entities** For each primitive entity identified from the domain entities analysis:

*1.1 Check the entity description’s completeness in accordance with Rule R-REFI.1 and add new data items using Rules R-REFI.2 and R-REFI.3 if the description is incomplete.*

**R-REFI.1 (Complete entity descriptions):** *An entity description must include all the data items that the entity needs.* This rule ensures that all entity descriptions are complete and self-sufficient in terms of their data items. For example, an “AISC L-Shape Descriptions” entity description (which includes “dimension d1,” “dimension d2,” and “thickness t” data items) needs another data item, “dimension unit,” to complement the data items already included.

R-REFI.2 (Safe addition of necessary data items to a primitive entity description): *An incomplete primitive entity description can include necessary new data items only if they do not affect its cohesion.* Otherwise, the modeler must re-evaluate the entity's cohesion (and possibly decompose the entity as in Phase 3). For example, the modeler can "safely" add the dimension unit data item to "AISC L-Shape Descriptions" in the previous rule. Descriptive data items are items such as units, names, verbose descriptions, notes, references, user-defined identifiers, etc. that further enhance an entity description. "Dimension unit" is such an item. Adding such data items will not affect the cohesion of primitive entities.

R-REFI.3 (Addition of optional data items to an entity description): *An entity description can include optional data items, which will be converted into optional attributes in the equivalent class definition. However, these data items must be limited to few (or even one) per primitive entity.* This rule provides the modeler with the flexibility of adding information to an entity description. However, optional data items affect the time-cohesion of primitive entities since the optional data item values need not be assigned at any time. (By definition, a primitive entity is time-cohesive if all its data item values must be specified at the same time.) They also affect the entities' use-cohesion since they may not always be used with the existing data items of the entity. (By definition, an entity is use-cohesive if all or none its data items are used in the "primary data uses" of the domain.) Therefore, optional data items must be limited to few (or even one) per primitive entity. For example, the description of a primitive entity, "Behavior Response Forces," includes one optional data item, "optional name," whose values can be designations such as "F<sub>A</sub>," "M<sub>B</sub>," "T<sub>C</sub>," etc.

1.2 *Review all the dependencies of the primitive entity on other entities in the schema using Rule R-REFI.4 and represent each of these dependencies in the entity description using the rule set R-REFI.5.*

R-REFI.4 (Include all dependencies in the entity description.): *An entity description must capture all the dependencies of the entity on other entities in the schema.* The next rule and its components specify different representations for various types of entity dependencies. For example, a "Design History Descriptions" entity has two dependencies: a dependency by reference on "Design Artifact Descriptions" and another by definition on "Design Operation Descriptions."

R-REFI-5 (Representing entity dependencies): This rule includes the following component rules.

R-REFI-5.1 (Options for representing entity dependencies by derivation): *If a primitive entity, X, is dependent by derivation on another primitive entity, Y, the modeler can choose one of the following options to represent the dependency: She can (1) incorporate a "derives" relationship of derivation type into the description of Y and include a user-defined object identifier in the descriptions of both X and Y, (2) incorporate a "derives" relationship of derivation type into the description of Y and a "derived-from" inverse relationship into the description of X, or (3) eliminate X by incorporating its data items as "derived data items" into the description of Y.* No single rule applies to all cases of entity dependencies by derivation. Instead, three different options are available. Rules R-REFI-5.1.1 to 5.1.3 in Appendix C explain the representations, advantages, and tradeoffs of these options. They also show examples for each of these options.

R-REFI-5.2 (Options for representing entity dependencies by definition): *If a primitive entity, X, is dependent by definition on another primitive entity, Y, the modeler can choose between the following options to represent the dependency: (1) representing Y as an abstract data type and using this data type in the description of X, or (2) incorporating a "subpart" relationship of aggregation type into the description of X.* Rules R-REFI-

5.2.1 to 5.2.2 in Appendix C explain the representations, advantages, and tradeoffs of these options. They also show examples for each of these options.

**R-REFI-5.3** (Options for representing entity dependencies by reference): *If a primitive entity, X, is dependent by reference on another primitive entity, Y, the modeler can choose between the following options to represent the dependency: (1) incorporating a “referred-to” relationship of referential association type into the description of X, or (2) including user-defined object identifiers in the descriptions of both X and Y to associate their instances.* Rules R-REFI-5.3.1 to 5.3.2 in Appendix C explain the representations, advantages, and tradeoffs of these options. They also show examples for each of these options.

**2. Transforming Refined Primitive Entities into Primitive Classes** For each primitive entity refined in the previous step:

2.1 *Convert the entity description into a class definition, each reference data item in the description into a relationship attribute, each derived data item into a derived attribute, and each of the remaining data items into an independent attribute.*

2.2 *Review the class definition using Guidelines G-DESI-O.1 to O.2.*

**G-DESI-O.1** (Use explicit class names.): *The name of a class should explicitly articulate what the class represents.* This guideline is designed to enhance the self-explanation and expressiveness of the resulting schema. For example, “Square Shape Descriptions,” “Solid Rectangle Shape Descriptions,” “Triangle Shape Descriptions,” and “Solid Circle Shape Descriptions” classes represent four common geometric shapes. In cases where the class name is long and inconvenient to the implementation (e.g., “Cross-Section Modulus of Elasticity Properties”), the modeler can shorten it while making it as explicit as possible (e.g., “Section Elastic Moduli”).

**G-DESI-O.2** (Reconsider classes with few attributes.): *A class with one or a few attributes should be considered for possible elimination using Guidelines G-DESI-H.2 to H.4.* This guideline is intended to enhance the efficiency of the resulting schema and to make sure that a class is only introduced when it is needed (and thus meaningful to the resulting schema). The complementary guidelines G-DESI-H.2 to H.4 show when to include a class in a class hierarchy and discuss the special case of abstract superclasses with no attributes. For example, a “Structural Analysis Element Descriptions” class that has only one attribute, “element identifier,” was reconsidered but not eliminated because it provides a place-holder for adding subclasses representing more specific types of analysis elements.

2.3 *Review each attribute in the class definition using Guidelines G-DESI-A.1 to A.3.*

**G-DESI-A.1** (Use explicit attribute names.): *The name of an attribute should explicitly articulate the object property that the attribute represents.* This guideline is designed to enhance the expressiveness and self-explanation of the resulting schema. For example, the name of a relationship attribute can include a prefix such as “derived,” “own,” or “referred,” which clearly denotes the relationship type. Examples are “derived load cases,” “own load parameters,” and “referred load sources.” In cases where the attribute name (e.g., “standard derived cross section properties”) becomes long and inconvenient to the implementation, the modeler can shorten it (e.g., “derived section properties”) while making it as explicit as possible.



G-DESI-A.2 (When to include a user-defined object identifier): *A class does not need a user-defined object identifier unless the user uses such an identifier. In that case, such an identifier must be a required and unique attribute.* In the object-oriented paradigm, the object identity of each instance is globally unique and is created and maintained by the system. However, if the user chooses to define an explicit attribute as her own identifier of object instances, then that attribute must be given a unique value in each instance. For example, “shape size designation,” “material designation” and “requirement identifier” are user-defined object identifiers for “AISC Shape Descriptions,” “Material Properties,” and “General Requirements” classes respectively.

G-DESI-A.3 (Remove redundant relationship attributes.): *Redundant relationship attributes should be removed.* This rule ensures minimality of the resulting schema. Redundant relationship attributes occur when different paths that follow the relationship links go from one source class to the same destination class and produce the same effect. This rule applies to relationships of any type. In particular, cycles of relationships must be completely eliminated. For example, a “Load Condition Specifications” class has relationships with two other classes, “Load Source Specifications” and “Load Case Specifications.” “Load Case Specifications,” in turn, has a relationship with “Load Source Specifications.” In this case, the last relationship creates a cycle and can be removed.

2.4 *Add to the class definition a “compute” method for each derived attribute using Rule R-DESI-M.2 and a pair of “get” and “set” methods for each of the remaining attributes using Rule R-DESI-M.3. Name each method using Guideline G-DESI-M.1.*

G-DESI-M.1 (Use explicit method names.): *The name of a class method should explicitly articulate what the method performs.* This guideline is designed to enhance the expressiveness and self-explanation of the resulting schema. For instance, the name of a method for setting an attribute value includes a prefix such as “set” to denote the operation of the method clearly. Similarly, the name of a method for retrieving an attribute value includes a prefix such as “get.” As an example, the “distance d1” attribute of the “AISC Angle Shape Description” class corresponds to two methods, “set distance d1” and “get distance d1.”

R-DESI-M.2 (Define methods for computing derived attribute values.): *A class method must be defined for each derived attribute in order to compute its value on demand.* A derived attribute is dependent on other attributes; a method is defined to compute that attribute, which is derived on demand. (By contrast, an independent attribute is stored in the database.) For example, the three derived attributes of a “Cartesian Vectors” class, which represent x, y, and z directional cosines, yield three corresponding methods: “compute cosine x,” “compute cosine y,” and “compute cosine z.”

R-DESI-M.3 (Define methods to update and retrieve independent attribute values.): *A pair of class methods must be defined for each independent attribute to set and retrieve its value. “Set” methods must have at least one argument, while “get” methods must return at least one response value.* Attributes are typically private to their object class. This rule ensures that a class has access to its attributes via methods. In fact, these methods set the object state or provide the means to inquire about the object state. “Set” methods may not return any response, while “get” methods may not have any argument. For example, the two independent attributes of the “Cartesian Vectors” class in the previous rule, which represent the magnitude and direction, yield four methods: “set magnitude,” “get magnitude,” “set direction,” and “get direction.”

### **3. Building Primitive Characterization Hierarchies with the New Primitive Classes** For each primitive class resulting from the preceding step:

- 3.1 *Identify the primitive characterization hierarchy to which the primitive class belongs. If no such hierarchy exists, create a new hierarchy whose root class is the most general class representing the concept pertinent to the primitive class.*
- 3.2 *Add the primitive class to the identified or newly created hierarchy and expand that hierarchy using Rules R-DESI-H.1 to H.2 and Guidelines G-DESI-H3 to H.5.*

R-DESI-H.1 (Do not misuse relationships of generalization type in class hierarchies.): *Two classes belong to the same class hierarchy only if one represents a more specialized definition of the concept represented by the other. For example, a “Structural Members” class should not be a subclass of “Buildings.”*

R-DESI-H.2 (Define a common superclass.): *If two or more classes have one or more attributes in common, these classes must have a new common superclass in the class hierarchy. This rule is designed to remind the modeler to take advantage of generalization. However, the modeler must weigh the tradeoffs between introducing a superclass and implementing the code of the common attributes in all classes sharing those attributes. For example, “AISC I-Shape Descriptions,” “AISC T-Shape Descriptions,” and “AISC C-Shape Descriptions” classes can share a common superclass, “AISC Symmetric Rolled Shape Descriptions,” which includes four attributes common to these classes.*

G-DESI-H.3 (Define a typical class.): *If several instances share two or more common attribute values, they should be instances of a new typical class in the class hierarchy. This guideline is designed to remind the modeler to take advantage of typical classes, which are very beneficial in representing data of facility design objects. The new typical class has default attributes whose values are the commonly used values. For example, a “Tower Leg Analysis Element Descriptions” typical class is a subclass of a “Two-Node Analysis Element Descriptions” class. The typical class attributes “element type,” “start support type,” “end support type,” and “length unit” have the default values of “truss element,” “hinged,” “hinged,” and “feet” respectively.*

G-DESI-H.4 (When to introduce a new subclass in a class hierarchy): This guideline includes the following component guidelines.

G-DESI-H.4.1 (Do not invent a subclass with no attributes.): *A new subclass in a class hierarchy should add at least one new attribute to the set inherited from its superclass. This guideline ensures that a new subclass is added only when the subclass has a distinct identity in its hierarchy. For example, three new subclasses, “AISC I-Shape Descriptions,” “AISC T-Shape Descriptions,” and “AISC C-Shape Descriptions,” are added to the class hierarchy representing AISC standard shape descriptions. Each subclass has at least one new attribute. By contrast, superclasses with few (or even one) attribute are useful where they provide a place-holder for: (1) adding more specialized subclasses later, or (2) defining relationship attributes of other classes, which can be linked to any subclasses of the superclass. These superclasses are called “abstract superclasses.” The example given in Guideline G-DESI-O.2 demonstrates the first case. An illustration of the second case is the abstract superclass, “AISC Rolled Shape Descriptions.” This superclass provides a convenient place-holder for an instance of “AISC Combination Shape Descriptions” that can later be linked to instances of subclasses of the superclass (e.g., “AISC L-Shape Descriptions,” “AISC I-Shape Descriptions,” “AISC C-Shape Descriptions”).*

G-DESI-H.4.2 (Do not invent a subclass when a relationship will do.): *A new subclass should not be introduced if its only additional feature can be represented by a relationship in its superclass.* For example, “Gravity Load Resisting Frames” class should not be introduced as a new subclass of “Frames” since its only new feature can be represented by a relationship to the appropriate load types supported by those frames.

G-DESI-H.4.3 (Do not add a level when a discriminating attribute will do.): *Another level in the class hierarchy should not be introduced if a single attribute in the superclass can distinguish the different subclasses.* This guideline is designed to enhance the minimality and efficiency of the resulting schema. For example, a “AISC Combination Shape Descriptions” class includes an attribute to distinguish different types of shape combinations (i.e., “combination shape type”), thereby eliminating the need for another unnecessary level of subclasses such as “S-C Combination Shape Descriptions,” “C-L Combination Shape Descriptions,” “C-C Combination Shape Descriptions,” etc.

G-DESI-H.5 (Shallow Primitive Characterization Hierarchies): *Primitive characterization hierarchies should be kept shallow.* Several shallow primitive class hierarchies are preferred to a few deep hierarchies. As a rule of thumb, a primitive characterization hierarchy must contain at most three levels. Otherwise, the modeler should first review the hierarchy using Rule R-DESI-H.1 and the preceding set of guidelines. She should consider eliminating classes at the intermediate levels if possible. In addition, the modeler should review the overall concept that was used to build the class hierarchy. A deep primitive characterization hierarchy is indicative of a concept that is not sufficiently distinctive. In that case, the concept should be refined into several subconcepts, each of which will lead to a more shallow primitive class hierarchy. For example, a deep primitive characterization hierarchy on shape descriptions was broken up into two separate hierarchies: one on geometric shape descriptions and the other on AISC standard shape descriptions.

### **6.3.4 Designing A Composite Schema**

As explained earlier, a composite schema includes a subset of a domain primitive schema and a set of composite classes that represent a user’s view of the underlying facility data. Therefore, a composite schema can be defined to formulate a complex query of facility design objects on a primitive database or build a new application that will operate on this database. The steps in defining a composite schema are:

1. *Determine all the data items needed in that composite schema.*
2. *Divide the data items into three sets: those that can be represented using primitive classes from the given primitive schema, those that can be represented using combinations of primitive classes, and those that cannot be represented using the primitive schema.*
3. *For the first set of data items, select the appropriate primitive classes from the primitive schema and include them in the composite schema.*
4. *For the second set of data items, define composite classes as necessary combinations of primitive classes selected from the schema. Include these primitive classes in the composite schema and determine the relationships (of the generalization, aggregation, or association type) between them and the new composite classes. If necessary, define new composite classes using those that were just created.* Guideline G-DESI-H.6 suggests that composite class hierarchies are not needed.

**G-DESI-H.6 (No Composite Class Hierarchies):** *Composite class hierarchies should not be necessary.* Although by definition, a composite class can be a subclass of another composite class, composite class hierarchies are not recommended here for the following reason: The flexibility of the P-C Approach comes from allowing the user to define any composite class when needed from a given primitive schema. At any time, the user can instantly customize a composite class by selecting primitive classes from primitive characterization of this schema. Therefore, both predefined composite classes and composite class hierarchies should not be necessary. An exception to this guideline is when a typical class (e.g., “Tower Cage Members As Designed”) that is a subclass of a composite class (e.g., “Tower Members As Designed”) is needed to capture the common values shared by instances of the composite class (e.g., length, member size, material designation).

5. *For the third set of data items, define new classes of the composite schema to represent those data items.* Guideline G-DESI-A.4 recommends that only as the last resort should a composite class add attributes other than those of the primitive or composite classes from which it is defined.

**G-DESI-A.4 (No new attributes in composite classes):** *The introduction of new attributes in composite classes should be minimized.* A composite class should not introduce any attributes other than those of the primitive or composite classes from which it is defined. This guideline implies that the domain primitive schema should if possible, provide all the primitive classes necessary to support different user views in the domain. For example, the definition of the composite class, “Tower Members As Analyzed,” uses the following primitive classes: “Two-Node Analysis Element Descriptions,” “Section Areas,” “Material Moduli,” “Load Application Specifications,” and “Cartesian Coordinate Systems.” It does not add any new attributes.

## **6.4 Chapter Summary**

---

This chapter describes the design of a domain primitive schema using an object-oriented data model, the Primitive-Composite (P-C) Data Model, and an accompanying method, the P-C Data Modeling Method. The development of this method (in addition to the model’s development) was an attempt to fulfill the need for object-oriented modeling methods that was pointed out in Chapter 2.

With the primitive entities identified in the domain entity analysis, a modeler designs a domain primitive schema (or “primitive schema”) in Phase 4 of the P-C Approach. This schema includes hierarchies of primitive classes that define the basic concepts used by experts in a domain. The schema design involves three steps: refining those primitive entities, transforming them into primitive classes, and building primitive characterization hierarchies with these classes. To aid the modeler with this design, the P-C Approach provides an object-oriented data model, the P-C Data Model, and an accompanying method, the P-C Data Modeling Method, for using this model. The model includes key object-oriented concepts such as classes, attributes, methods, and instances. It incorporates two other sets of extensions: (1) extensions to the key object-oriented concepts such as “primitive classes,” “primitive characterization hierarchies,” “composite classes,” “primitive instances,” “typical class,” etc., and (2) a number of formalized relationship types such as generalization, instantiation, aggregation, association, and derivation. The method provides the steps for the design of a domain primitive schema based on the concepts of the model. In addition, the method provides rules and guidelines for the design of the primitive schema. With the resulting primitive schema, composite schemata can be defined to formulate complex queries of facility design objects on the primitive database or build new applications that will operate on this database. Composite

schemata include subsets of the primitive schema, as well as their own composite classes representing specific views of the underlying facility data. The method also provides the steps and guidelines for defining a composite schema.

The next chapter presents the domain primitive schema that we developed for transmission towers. It explains the form, function, and behavior representations included as primitive classes in the tower primitive schema. It also shows different ways to describe a tower facility's hierarchical decomposition using this schema.

# Chapter 7

## Form, Function, and Behavior Representations in the P-C Approach

### Chapter Abstract:

The previous chapters described the methodology and modeling tools a modeler can use to develop a domain primitive schema following the P-C Approach. This chapter focuses on the content of such a schema, namely the representation elements that the schema should include. The examples shown in the chapter come from the schema for the transmission tower domain. First, form, function, and behavior are represented in separate primitive class hierarchies of the schema. Geometric curves, topological elements, material properties, fabrication features, functions, requirements, strength behavior (i.e., response forces and stresses), serviceability behavior (i.e., displacements and strains), design description primitives, etc., are represented in these primitive class hierarchies. In addition, the schema includes design description primitive classes such as design artifacts, features, parameters, versions, and alternatives. These primitive classes represent important elements used in describing a design. With the schema, a facility can be decomposed hierarchically in two ways: by the functions that it provides or by the design artifacts created by the designers. The schema includes primitive classes needed to support both decomposition types, as well as the mappings between functions and artifacts.

### Organization:

#### 7.1 Overview

#### 7.2 Form, Function, and Behavior Representations

##### 7.2.1 Form Representation

##### 7.2.2 Function and Behavior in addition to Form

##### 7.2.3 Function Representation

##### 7.2.4 Behavior Representation

#### 7.3 Hierarchical Facility Decomposition by Functions or by Artifacts

##### 7.3.1 Decomposition by Functions

##### 7.3.2 Decomposition by Design Artifacts

##### 7.3.3 Mappings Between Functions and Artifacts

#### 7.4 Chapter Summary

## 7.1 Overview

The P-C Approach advocates the representation of the function and behavior of facility design objects in a domain primitive schema, in addition to their form. The inclusion of these representations can help in: (1) answering useful queries about the purpose of facility design objects and the states for which these objects were analyzed and designed; (2) improving communication about the design among the participants in collaborative facility engineering work; (3) assisting design-related tasks such as what-if analysis, facility retrofitting/rehabilitation, design reuse across projects, and design tutoring; and (4) preserving corporate knowledge about constructed facilities' design and operations. In the P-C Approach, form, function, and behavior are represented explicitly as classes in several separate primitive characterization hierarchies. Examples of these hierarchies are geometric curves, topological elements, material properties, fabrication features, functions, requirements, strength behavior, serviceability behavior, etc. The separation of these hierarchies results in clean and modular facility data representations. In fact, each hierarchy uses a single criterion to define the primitive classes and thus provides a homogeneous view about one specific aspect of the facility design objects. In addition, the domain primitive schema includes design description primitive classes such as design artifacts, features, parameters, constraints, versions, alternatives, etc. These primitive classes represent important elements used in describing a design. Together, form, function, behavior and design primitive classes of the schema support three principal dimensions of a facility design: (1) design characterization (i.e., form, function, behavior), (2) design decomposition and recomposition, and (3) design versioning and alternative selection. Finally, with this schema, a facility can be decomposed hierarchically in two ways: by the functions that it provides or by the design artifacts created by the designers. The schema includes primitive classes needed to support the two decomposition types, as well as the mappings between functions and artifacts.

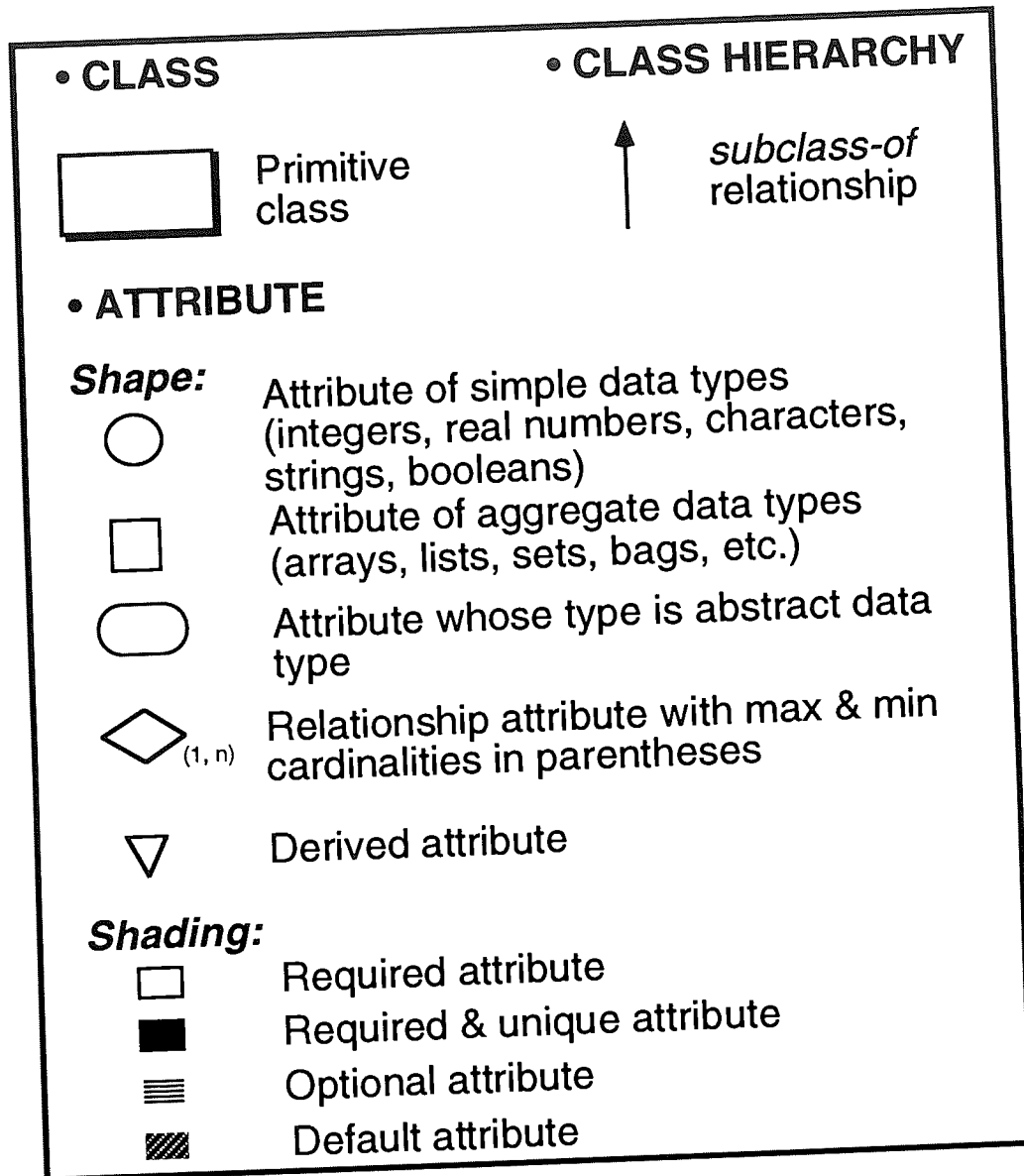
We applied the P-C Approach to the transmission tower domain. Figure 1.1 in Chapter 1 illustrates a sample transmission tower. The resulting domain primitive schema includes 216 primitive classes. These classes are necessary to describe the facility design objects in the tower domain. Although there is a large number of primitive classes in this case, these classes are organized into 30 primitive characterization hierarchies. Each primitive characterization hierarchy describes only one concept about either form, function, or behavior. In addition, we implemented this schema using a commercial object-oriented database management system (ONTOS) and built a database of a selected tower to demonstrate the research concept.

All classes of the tower domain primitive schema are fully documented in Appendix D. Only sample hierarchies of these classes are illustrated in this chapter. Graphical representations that are used to illustrate the hierarchies are based on those presented in [Batini 92] for conceptual data modeling. Due to space constraints, Figure 7.1 provides a legend for the graphical representations that apply to all the primitive characterization hierarchies shown in this chapter.

## 7.2 Form, Function, and Behavior Representations

### 7.2.1 Form Representation

A domain primitive schema should include primitive classes that represent the physical characteristics of the facility design objects. These classes can be divided into the following categories:



**FIGURE 7.1: Legend for the Graphical Representations of Sample Primitive Characterization Hierarchies.**

**Spatial Reference Form**—Primitive classes in this category are used to describe where and how a physical object is located and oriented in three-dimensional space. They can also represent the spatial enclosure of the object and the enclosure's location and orientation with respect to a global coordinate system or relative to other objects in the same environment.

*Example:* "Position Forms," "Orientation Forms," "General Coordinate Systems," "Spatial Enclosure Shapes," and "Spatial Enclosure References" are spatial form primitive classes.

**Geometry Form**—Primitive classes in this category represent fundamental geometric elements such as points, curves, and surfaces, and many specialized types of these elements.



*Example:* “Cartesians Points,” “Cylindrical Points,” “Line Segments,” “Conic Curves,” “Arcs,” “Planes,” “Conical Surfaces,” and “Surfaces of Linear Extrusion” are primitive classes representing specialized types of geometric elements.

**Topology Form**—Primitive classes in this category are used to describe the connectivity of a physical object in its constructed environment. These classes correspond to the topological entities that are defined in the PDES/STEP IPIM [Wilson 88] and the GARM [Gielingh 88].

*Example:* “Vertex Descriptions,” “Edge Descriptions,” “Face Descriptions,” “Path Descriptions,” “Wire Shell Descriptions,” and “Region Descriptions” are topological form primitive classes.

**Shape Representation Form**—Primitive classes in this category are used to describe the shape of a physical object, including its dimensions and section properties. Physical objects are three-dimensional, and their shapes are commonly represented using parametric descriptions or solid modeling elements [Mortensen 85].

*Example:* “Geometric Shape Parametric Descriptions,” “General Shape Section Properties,” “AISC Standard Shape Descriptions,” “Shape Parametric Projections,” “Right Solid Cylinders,” and “Solids of Revolution” are shape representation primitive classes.

**Material Form**—Primitive classes in this category represent properties of the material from which the physical object is built. The materials used in civil engineering include steel, reinforced concrete, asphalt, mortar, timber, etc. In general, materials can be classified into types such as homogeneous, isotropic, orthotropic, anisotropic, and composite [Wilson 88]. They can have structural, thermal, and thermal expansion properties.

*Example:* The primitive classes, “Mass Densities,” “Material Moduli,” “Material Strength Properties,” and “Thermal Expansion Coefficients” represent different material properties.

**Part Detailing/Fabrication Form**—Primitive classes in this category represent features of a fabrication part that a designer specifies for fabrication purpose. A number of standard fabrication features are defined in the PDES Integrated Product Information Model [Wilson 88] and the NIDDESC Ship Structural Model [Gerardi 88].

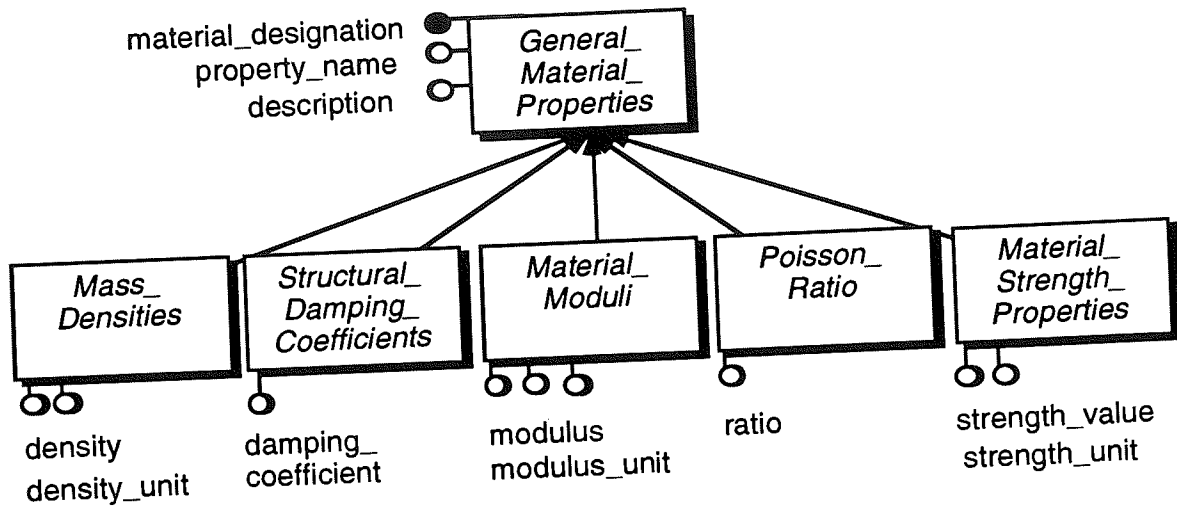
*Example:* “Fabrication Dimensions,” “NC Mark Features,” “Bolt Hole Features,” “Edge Clipping Features,” and “Bend Features” belong to a large set of primitive classes representing fabrication features.

Figure 7.2 shows sample primitive classes representing material properties.

### 7.2.2 Function and Behavior in addition to Form

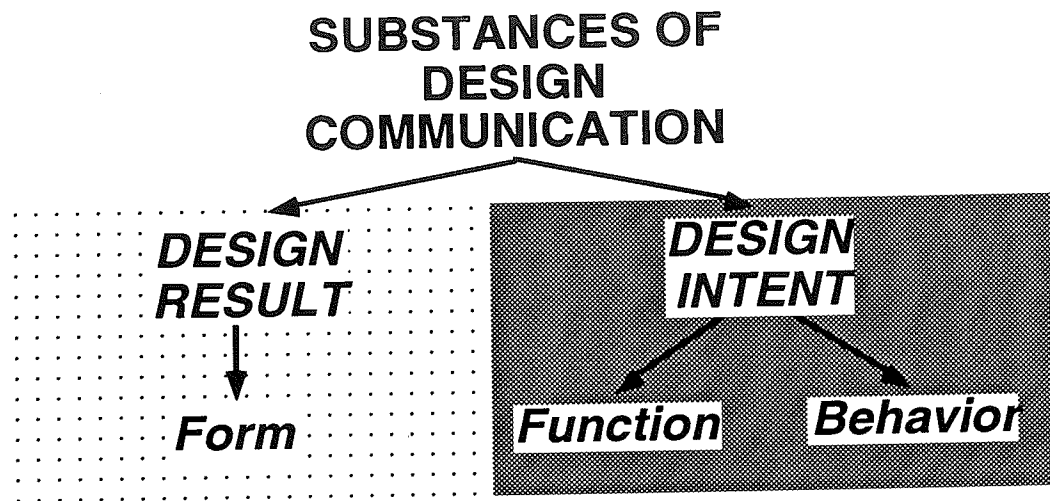
The P-C Approach advocates the representation and storage of function and behavior data in addition to form data. This can provide benefits in the following four areas:

- *Explanation*—Answering queries about the purpose (function) of the facility or a system or component, and the state (behavior) for which the object of interest was analyzed and designed: The study about information requested by engineers in [Kuffner 91] and [Ullman 91a] emphasizes the need for representing and storing function and behavior data for later use. Form cannot explain why the facility was designed and operates a certain way. But form, function and behavior can produce a well-founded explanation of the facility’s design and operations.



**FIGURE 7.2: Sample Primitive Classes Representing Material Properties.**

- Design communication—Improving design communication in collaborative facility engineering work:* Since storing function and behavior data enhances the explanation of the design, it also improves the communication about the design among different participants of a facility engineering project. In the past, engineering drawings have been the most common way to communicate information about a facility design product. Generally speaking, these drawings specify how the parts must be fabricated and how the facility must be constructed. They mainly show the physical description or “form” of the facility that results from the design, but do not capture the designer’s intent. Such drawings can be ineffective in design communication. The catastrophic structural failure at the Hyatt Regency Hotel in Kansas City showed how miscommunication of the designer’s intention can lead to loss of human life. The participants must be able to convey to one another the purpose of the design as well as the behavioral responses for which the facility was analyzed and designed. Design communication of this nature can help the participants resolve and even prevent errors, delays, cost overruns, low quality, and other problems [Howard 91a]. As Figure 7.3 emphasizes, *in the communication of information about a design product, the “design intent,” which includes function and behavior, is as important as the design result, which includes form.*
- Design—Improving facility engineering design:* The available function and behavior information can assist designers in performing the following tasks: re-specification [Shema 90], [Dixon 86], what-if analysis [Boy 91], retrofit and rehabilitation [Rafiq 90], design reuse across projects [Franke 91], and design tutoring [Gruber 91].
- Design knowledge capture—Helping companies to preserve knowledge about their facility design work for both immediate and long-term use:* Design knowledge capture is the process of “eliciting, recording and organizing design knowledge” [Gruber 90b].



**FIGURE 7.3: What Should Be Communicated About A Facility Design?**

Design knowledge encompasses the physical description of the design product (form), the assumptions about the context in which the product should be used (function), and the understanding of the state for which the product was analyzed and designed (behavior) [Gruber 90a]. Design knowledge capture involves an active area of research [Baudin 89], [Fischer 89], [Schema 90], [Rafiq 90], [Boy 91], [Conklin 91], [Gruber 91], [Garcia 92].

### 7.2.3 Function Representation

In the P-C Approach, the complete function representation of a facility design object includes three components: (1) the description of the object's purpose (or purposes), (2) the description of the requirements for achieving that purpose, and (3) the description of the artifact designed for that purpose and of other important elements such as features, parameters, constraints, versions, alternatives, etc., of the design. The following sections present the primitive classes that represent these descriptions.

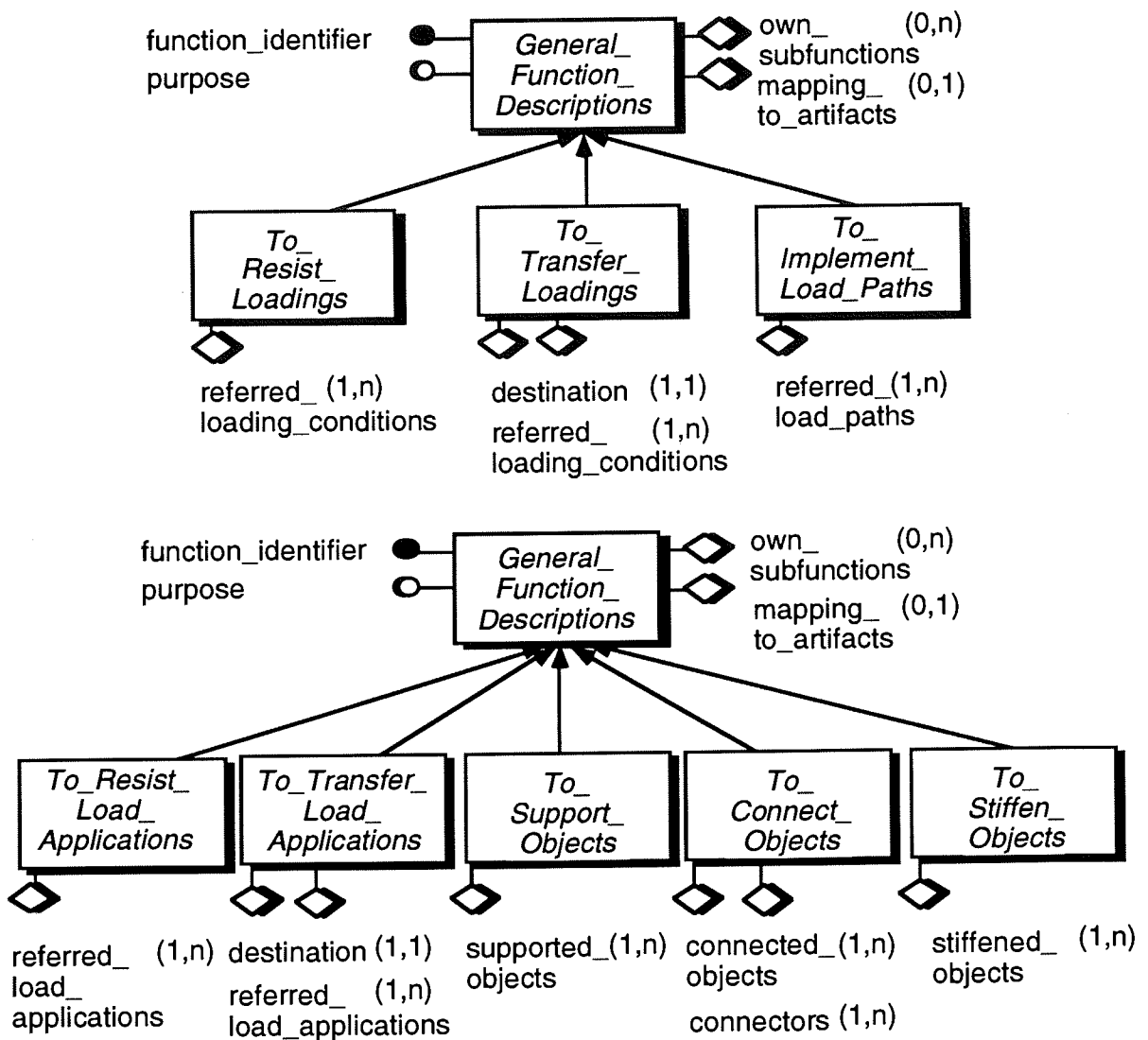
#### 7.2.3.1 Description of Functions

A function corresponds to a purpose or intended role of a design object in its constructed environment. A design object can perform several functions. For example, a wall resists loading (structural function) and provides a partition (architectural function). Even within the structural engineering view, the same wall may perform several functions such as resisting applied loads (gravity or lateral), supporting another object such as a floor slab, and transferring the applied loads to a destination such as a foundation. The representation of a function includes the function name, the perceived need, the purpose that the function describes, the requirements for achieving that purpose, the design solution (i.e., design artifacts or artifacts) that satisfies the function, and the component functions into which the function is decomposed.

The following functions are represented as function primitive classes:

**Functions of facility structures and structural systems**—One of the most important goals in designing a facility structure is to prevent failures. To accomplish this goal, the facility structure must provide two key functions: (1) *resisting loadings from the*

environment, and (2) transferring loadings to the ground. For example, a transmission tower must resist loading that comes from wind, ice, and gravity. Structural engineers often designs systems that carry out these functions. These systems perform another important function: *implementing gravity and lateral load paths*. For example, tower arm systems provide gravity load paths from the attached electrical equipment to the main body of the tower. The first subclasses of the “General Function Descriptions” class shown in Figure 7.4 represent these functions of facility structures and structural systems. In addition, the facility structure may perform functions involving other requirements such as owner, architectural, mechanical, physical, cost, and constructibility requirements. These requirements influence the facility’s structural design and are often reviewed by more than one project participant. The next section discusses the representation of requirements.



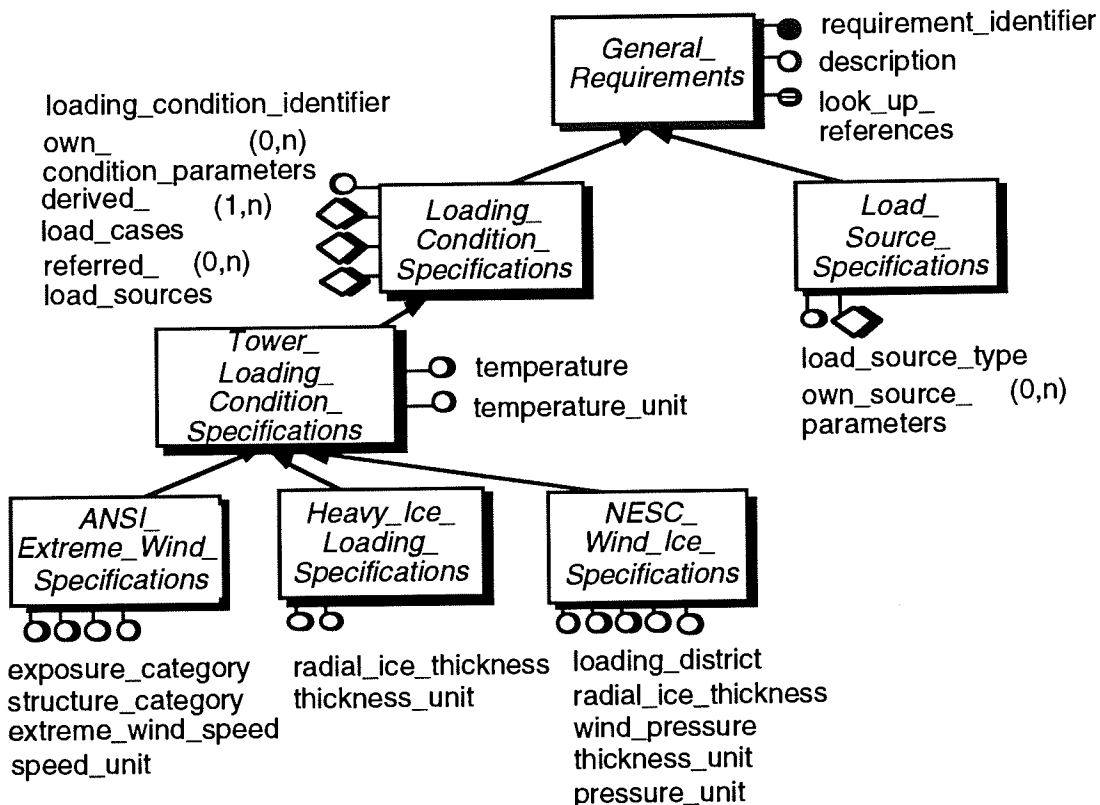
**FIGURE 7.4: Sample Primitive Classes Representing Structural Engineering Function Descriptions.**

**Functions of structural components (i.e., members and connections)**—Structural components are the next level of detail in a facility design. As Figure 7.4 shows, the functions that these components perform include: (1) *resisting a load application*, (2) *transferring a load application to another component*, (3) *supporting other components*, (4) *connecting other components*, and (5) *adding stiffness to other components*. For example, a tower’s leg member resists external wind loads applied to it and transfers them to the leg member below it. A tower’s redundant members add stiffness to leg and lacing members. A tower’s gusset plate connects a leg member, a lacing member, and a horizontal girt member using nuts and bolts.

### 7.2.3.2 Description of Requirements

The performance of a function can be measured by verifying the function’s requirements. The following are represented as requirement primitive classes:

**Principal loading conditions for which the facility structure, including systems and components, is analyzed and designed**—This information is very important to the analysis and design of a facility structure. It includes the parameters that are necessary to calculate the external loads that the structure must resist. The concept of a “loading condition” represents a scenario in which a facility structure is subjected to loadings from the environment. For example, transmission towers’ loading conditions include extreme hurricane winds and heavy ice, as well as normal wind and ice conditions. Each loading condition can involve one or more load sources. A load source can be gravity, wind, snow, high or low temperatures, an earthquake, an impact, etc. Figure 7.5 illustrates primitive classes representing these requirements.

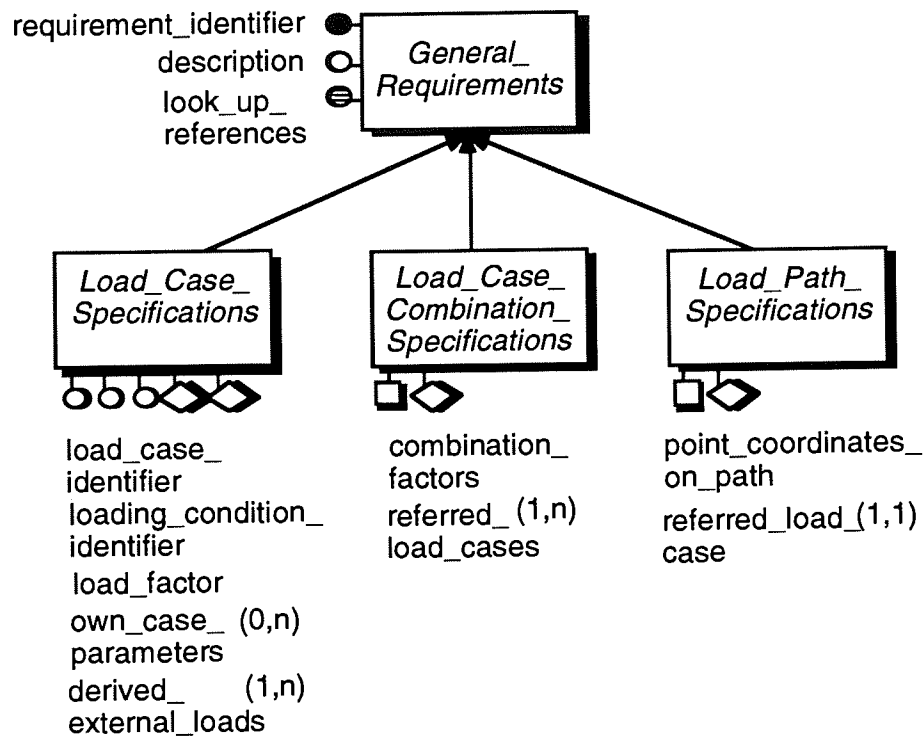


**FIGURE 7.5: Sample Primitive Classes Representing Specifications of Load Conditions and Load Sources.**

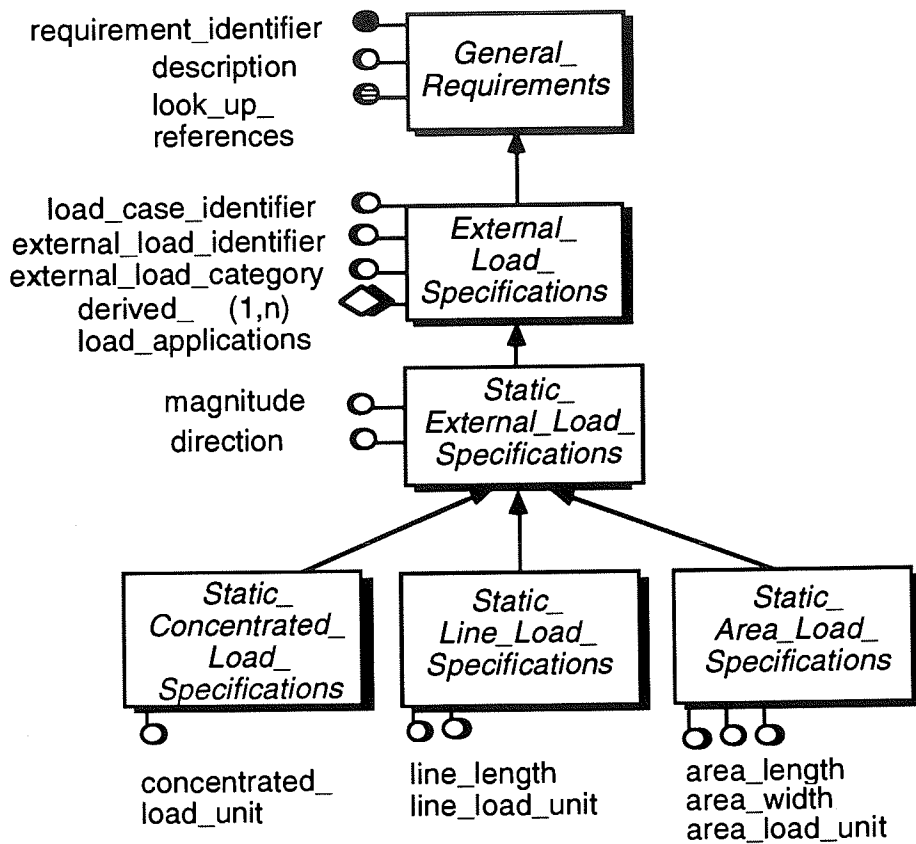
**Principal load cases for which the facility structure, including systems and components, is analyzed and designed**—A “load case” is a particular way in which a loading condition may occur. For example, an extreme hurricane wind loading condition can produce several load cases, each corresponding to a different wind direction: perpendicular to the transverse face of the tower, perpendicular to the conductor wires, at 45 degrees to the conductor wires, etc. Figure 7.6 illustrates primitive classes representing these requirements.

**Primary gravity and lateral load paths that structural systems implement**—There are two ways to represent a load path. As Luth [91] suggested, the first uses an instance of the “To Implement Load Paths.” This instance then refers to an instance of “Load Path Specifications” that directly records the successive points along the load path. The second uses an instance of the “To Support Objects” class for each member that belongs to the system. It defines the chain of load-resisting members along which loads are transferred. This second method applies to cases where a geometric configuration of the structure is already assumed when the load path is conceived. Figure 7.6 illustrates primitive classes representing these requirements.

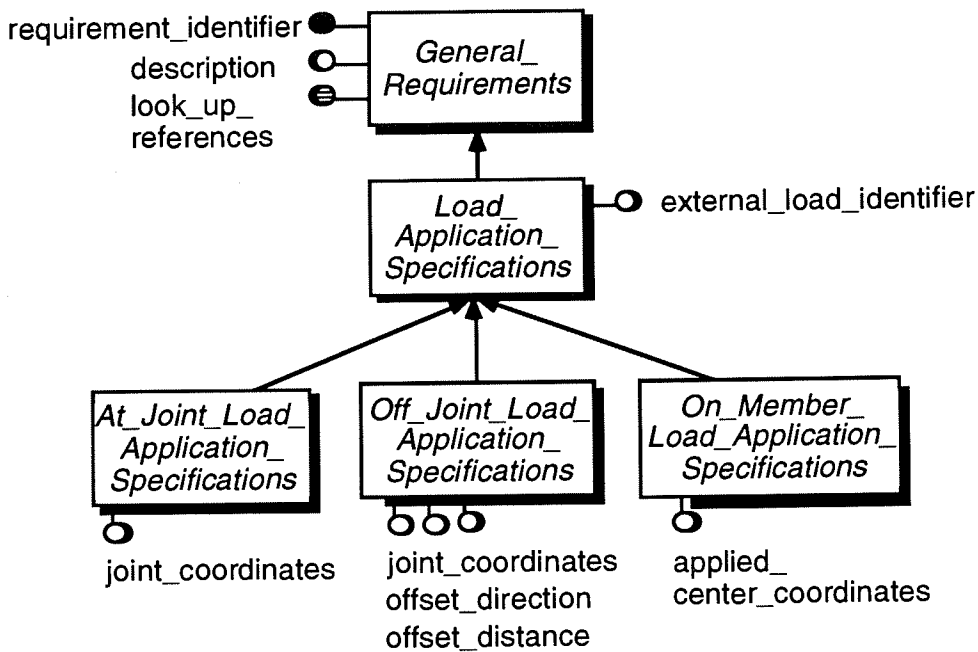
**External loads applied on structural members or at joints in the aforementioned load cases**—The concept of a “load application” represents a particular way in which an external load from the environment is applied to a member or joint of the structure and causes structural responses. Loads from “controlling load cases” should be stored. These load cases produce critical structural responses that govern the design of the members and connections. Figures 7.7 and 7.8 illustrate primitive classes representing these requirements.



**FIGURE 7.6: Sample Primitive Classes Representing Specifications of Load Cases and Load Paths.**



**FIGURE 7.7: Sample Primitive Classes Representing External Load Specifications.**

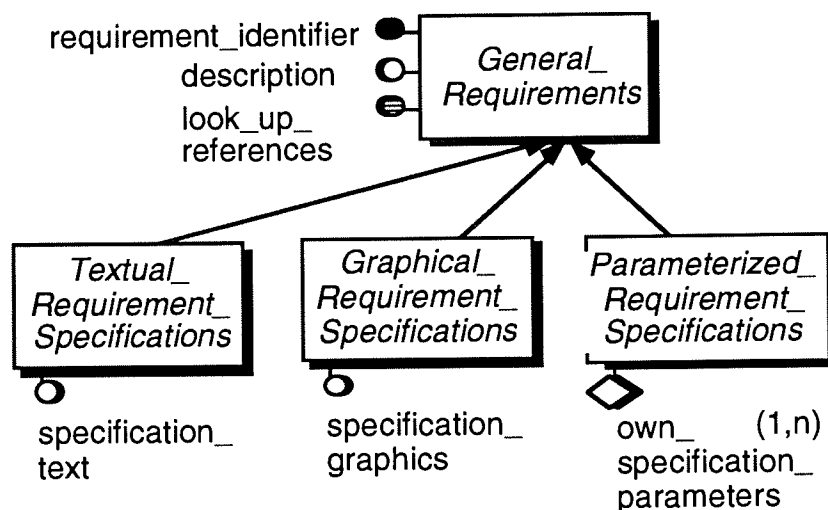


**FIGURE 7.8: Sample Primitive Classes Representing External Load Application Specifications.**

**Other requirements that govern the design of a facility structure**— A transmission tower design must meet strength, serviceability, physical, and cost requirements. Specification of these requirements involves recording the predefined values of hard constraints such as minimum safety factors, maximum deflection ratios, tower heights, electrical clearances, and a cost threshold. In general, requirements can be specified in building codes, company design manuals, engineering drawings, organization-sponsored design standards, etc. Requirement specifications can be textual, graphical, or parameterized. The first type includes written specifications such as fabrication notes, erection notes, etc. The second type includes drawing specifications for the fabrication of parts, connection of members, erection of the structure, etc. The third type of specifications are defined in terms of specific design parameters that need to be determined and satisfied during the design. Figure 7.9 shows primitive classes representing the three types of requirement specifications.

### 7.2.3.3 Description of Design

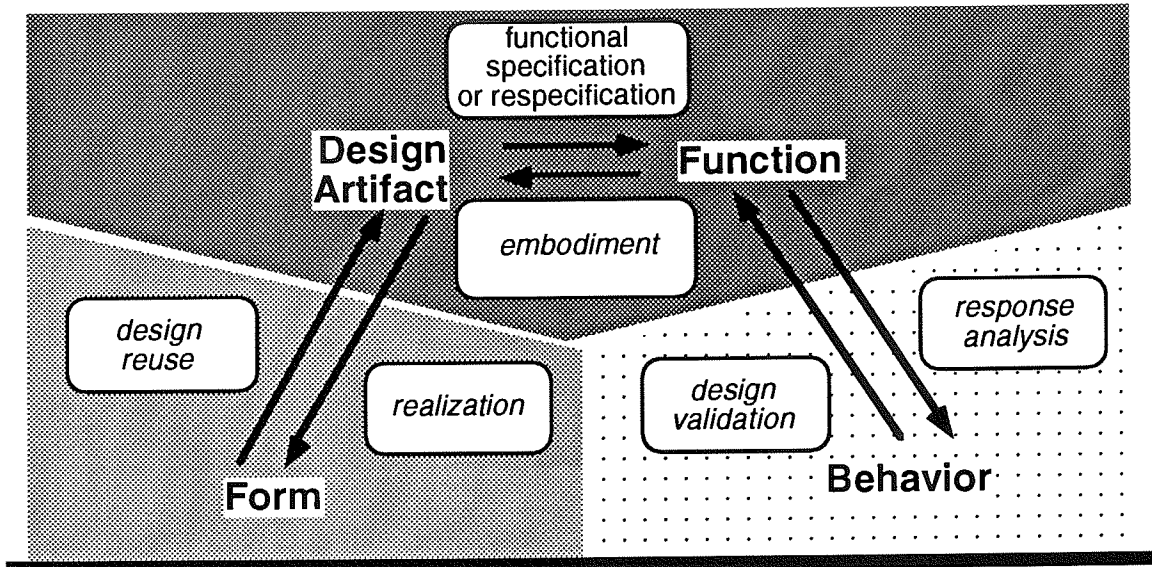
Design is important to the representation of facility design objects. For instance, the description of functions discussed earlier would not be complete without that of design artifacts. Therefore, a domain primitive schema should include primitive classes used to describe a facility design. “*Design description primitive classes*” represent important elements such as design artifacts, features, parameters, constraints, versions, and alternatives that are usually considered during a design. The definitions of these classes come from a review of the following work: [Mittal 86], [Dixon 86], [Dixon 88], [Meunier 88], [Geiling 88], [Zweben 89], [Sause 89], [Gruber 90b], [Luth 91], [Jain 91], etc. All the primitive classes, including the design description primitive classes and those describing form, function, and behavior, support three principal dimensions of a facility design as illustrated in Figure 7.10: “design characterization,” “design decomposition/recomposition,” and “design versioning and alternative selection.” The General A/E/C Reference Model (GARM) advocates three analogous principles of abstraction: “characterization,” “aggregation/decomposition,” and “product life cycle.” However, GARM does not include behavior, decisions, and alternatives.



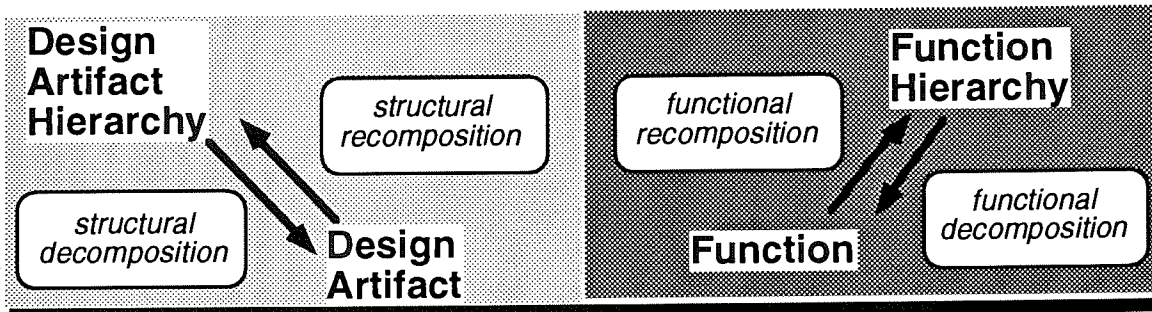
**FIGURE 7.9: Sample Primitive Classes Representing Requirement Specifications.**



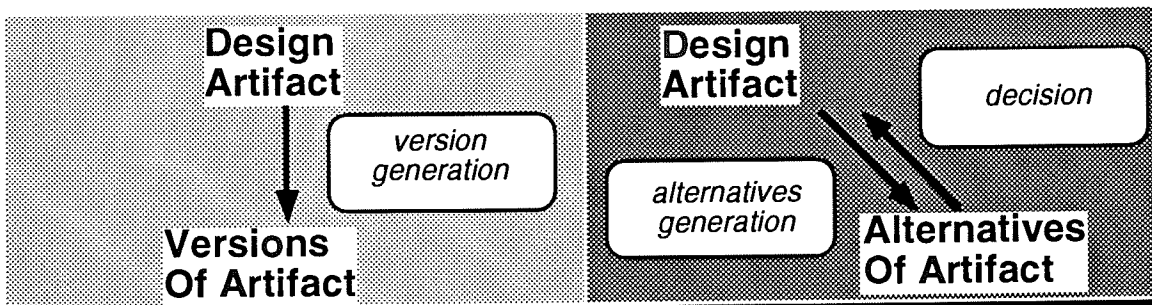
### 1. CHARACTERIZATION



### 2. DECOMPOSITION and RECOMPOSITION



### 3. VERSIONING and ALTERNATIVE SELECTION



LEGEND

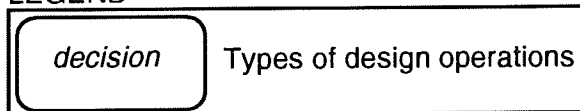


FIGURE 7.10: Three Principal Dimensions of a Facility Design Supported by the P-C Approach.

**Design Description Primitives** As Figure 7.11 shows, design description primitive classes represent descriptions of design artifacts, features, parameters, constraints, versions, alternatives, objective functions, design notes, and design graphics.

**Design Artifact**—A physical or abstract object created by a designer that can be realized from a formal specifications to provide an intended functionality or service. Examples of design artifacts are a transmission tower, a tower arm system, a tower leg member, a tower failure-safe mechanism, etc. A description of an artifact here involves stating what the designer is creating, namely its:

- classification (i.e., structure, system, assembly, member, or connection),
- design characteristic (i.e., standard, custom, or configuration) [Dixon 86],
- design components (i.e., other design artifact instances that constitute the design artifact's hierarchy), and
- mapping to the function or functions that the artifact performs.

**Design Feature**—An abstract or physical aspect of a design artifact that captures an important result from the process of creating the artifact. A feature can be defined using several design parameters. For example, leg members of a tower have an “interval step bolts” design feature that enables a construction worker to climb the tower. A design feature can be mapped into a form feature (or features) that is chosen to be the design solution.

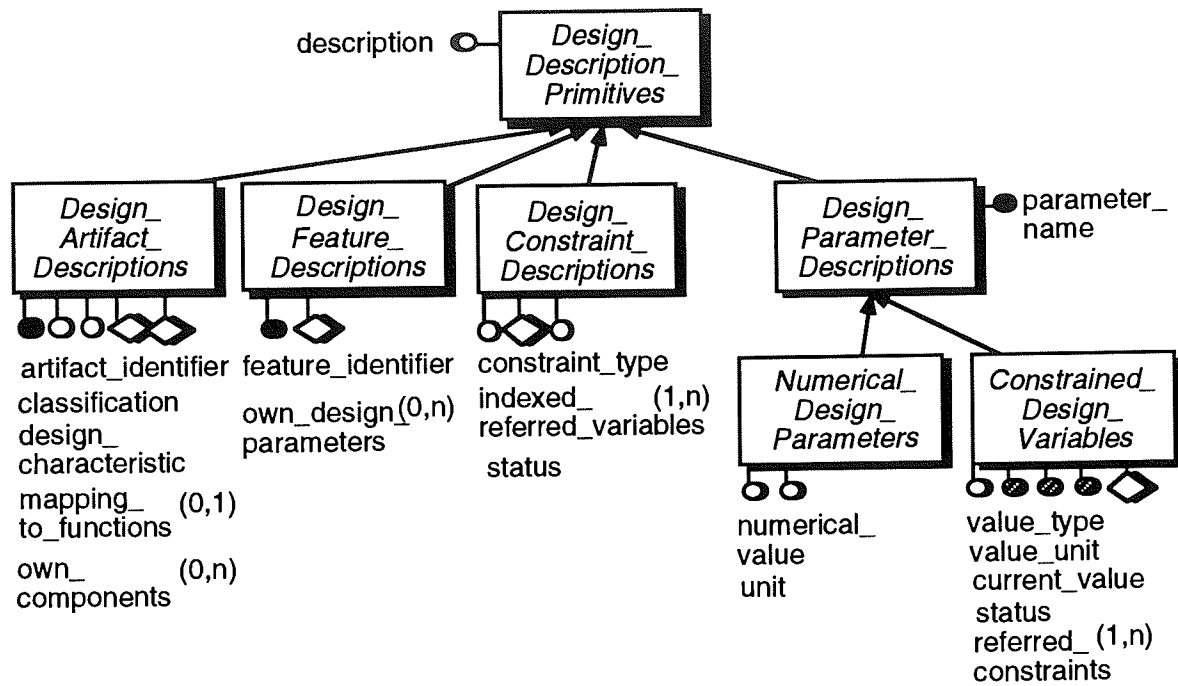
**Design Parameter**—A variable considered during the design of an artifact. Its value can be defined a priori or set during the design according to some hard or soft constraints. For example, a design of tower leg members involves setting the “Unbraced Lengths” design parameter. A design parameter can be a data item whose value can keep changing until the design is finalized.

**Design Constraint**—A condition that must hold for a set of design variables for a given artifact. A design variable can have several constraints that are considered during the design. The recording of constraints helps in the documentation of the design. This class definition is based on the work described in [Mittal 86] and [Zweben 89]. For example, the “Unbraced Lengths” design parameter of tower leg members must result in slenderness ratios of less than 200 to prevent column buckling.

**Design Alternative**—A proposed description of a design artifact that competes with other candidate descriptions for the same design goal and under the same design criteria. The designer may accept or reject a design alternative. The definition of the primitive class “Design Alternatives” is based on those given in [Abdalla 89] and [Gruber 90b]. For example, “butt-spliced members” and “lap-spliced members” are two alternatives for designing leg members of a tower.

**Design Version**—An accepted description of a design artifact that is either a modification, improvement, or elaboration of an earlier description. The designer may keep the current version or roll back an older version. The definition of the primitive class “Design Versions” is based on those given in [Abdalla 89] and [Gruber 90b].

**Objective Function**—“A function that evaluates the utility of alternatives with respect to some set of criteria” [Gruber 90b]. Indeed, objective functions can be used to measure the goodness of a design solution and to evaluate design alternatives that compete under a given set of criteria. Examples of objective functions for a tower design are the tower weight and cost functions.



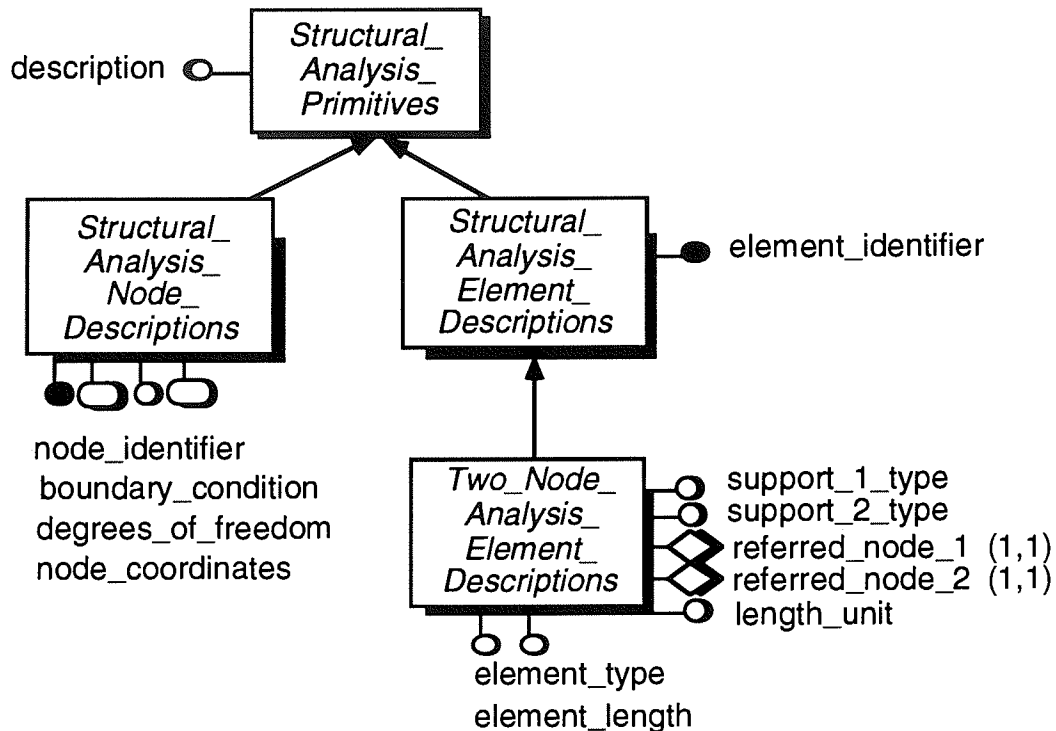
**FIGURE 7.11: Sample Design Description Primitive Classes.**

**Design Note and Design Graphic**—A design note is a textual annotation of a design. On the other hand, a design graphic is a sketch or a formal illustration of a design or some aspect of it. Both design notes and design graphics enhance the documentation of the design and thus the ability to explain the design later. A design note or graphic can be attached to any of the design description elements mentioned earlier. In addition, the user can nest a design note or graphic within another note or graphic and, as a result, can create an elaborate documentation, both textual and graphical, of a design.

### 7.2.4 Behavior Representation

The behavior of a design object is the object's response to environmental stimuli. Since a design object may perform several functions, it may exhibit different behaviors, each corresponding to a particular function. For example, in resisting gravity loads, a wall object may develop internal axial stresses; in resisting lateral loads, it exhibits shear and bending stresses. In structural engineering, internal forces, stresses, displacements, strains, vibrations, etc., characterize the behavior of a system or component under the influence of external loading.

Structural engineering design often involves performance criteria that impose certain limits on the behavior of the design objects. These criteria specify how effectively the structure can prevent different failure modes. They also ensure that the design objects perform according to professional standards. These criteria include: strength (i.e., stresses and internal forces), serviceability (e.g., deflection, cracking, vibration, etc.), reliability, stability, etc. In particular, strength ensures that the structure can sustain loading without failure. Serviceability ensures that the deformed structure under applied loading is both functional and acceptable. Since data describing behavioral responses is used to measure a design against such criteria, the representation of this data can be organized according to those criteria.



**FIGURE 7.12: Sample Primitive Classes Representing Descriptions of Structural Analysis Elements and Nodes.**

The following descriptions are represented as behavior primitive classes:

**Descriptions of the structural analysis elements and nodes, including their boundary conditions (i.e., input to the structural analysis)**—These descriptions are assumptions made in the structural analysis about the way structural components would behave. The information involved is usually encoded as input to a finite-element analysis program. Structural engineers review these assumptions when they designs the members and connections. In particular, they examine these assumptions closely when structural failures occur. Figure 7.12 shows primitive classes describing structural analysis elements and nodes.

**Descriptions of the structure’s behavioral responses to the applied loads (i.e., output from the structural analysis)**—According to [Hsieh 82] and [Wang 83], the types of behavior responses of a structure subject to loads include: (1) internal element forces, (2) support reactions (i.e., external “reactive” forces at the boundaries of the elements [Meriam 86]), (3) stresses, (4) linear or rotational displacements (for short, translations and rotations), (5) deformations (i.e., elongations, shortenings, elastic bending curves, and twistings), and (6) strains. The first three types constitute the structure’s “force responses” and deal with the strength performance criterion. The rest comprise the structure’s “displacement responses” and pertain to the serviceability criterion. Figures 7.13 and 7.14 show strength and serviceability behavior primitive classes respectively.

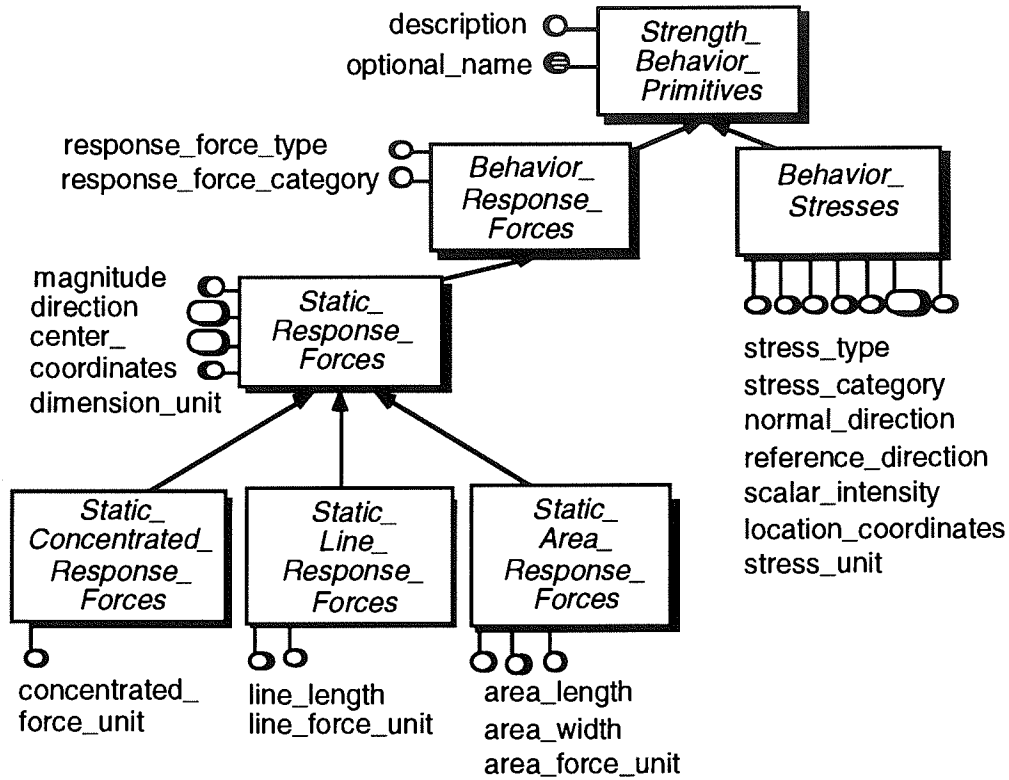


FIGURE 7.13: Sample Strength Behavior Primitive Classes.

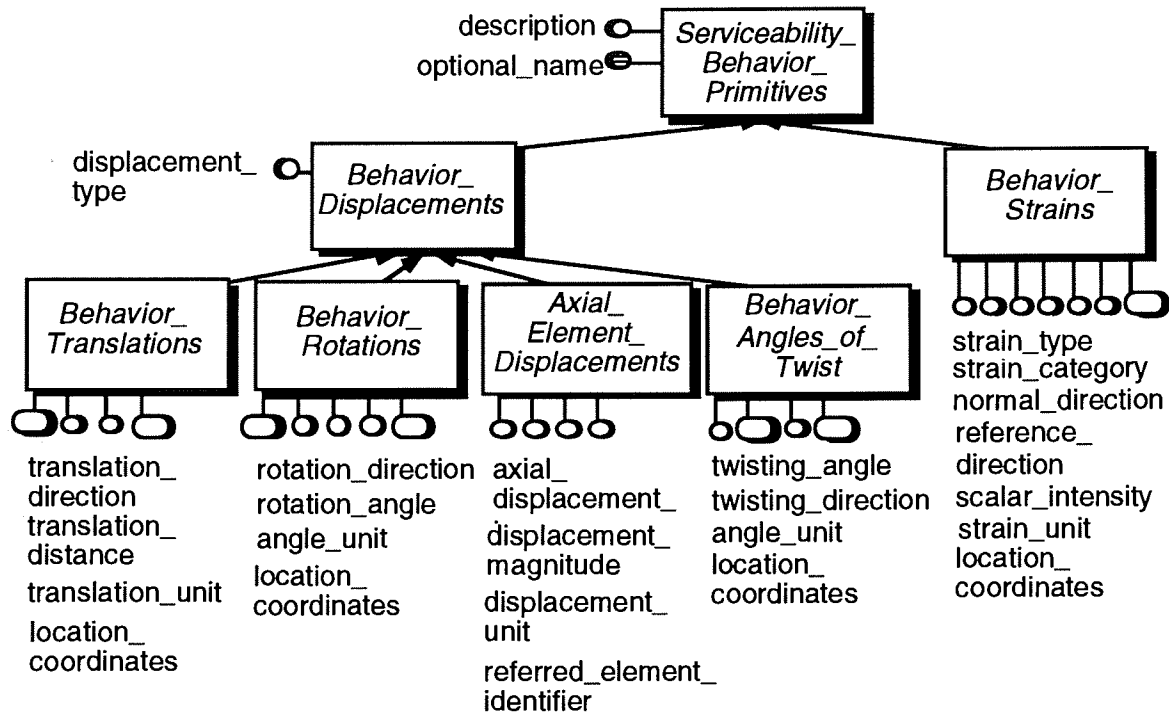


FIGURE 7.14: Sample Serviceability Behavior Primitive Classes.

### **7.3 Hierarchical Facility Decomposition by Functions or by Artifacts**

Studies on decomposition of complex products have been reported in literature on general design theory [Newell 72], [Goel 89a], VLSI design [Steinberg 87], mechanical engineering design [Dixon 86], [Mittal 86], [Meunier 88], and facility engineering design [Gielingh 88], [Sriram 89], [Sause 90]. In fact, scholars have proposed the following approaches for decomposing an engineering device: a single “functional hierarchy” [LaRota 90], a network of “functional units” and “technical solutions” [Geilingh 88], a “structure-substructure” hierarchy and a “function-behavior” hierarchy [Goel 89b], a “requirements model” and an “artifact model” [Mark 90], a functional hierarchy connected to a network of “behavioral views” [Umeda 90], and a “physical organization” and “functional organization” [Davis 85]. Other approaches focus on the decomposition of a facility: a single “abstraction hierarchy” with multiple levels of aggregation (e.g., building, systems, subsystems, members, elements, etc.), each containing a number of predefined entities [Maher 85], [Sriram 84], [Bjork 88], [Lavakare 90]; a “component hierarchy” of basic components and connections [Powell 88]; a “spatial hierarchy” and a “functional hierarchy” (including frames, bays, aisles, joints) [Eastman 78]; and three linked hierarchies such as “topological hierarchy,” “structural hierarchy,” and “architectural hierarchy” [Law 86].

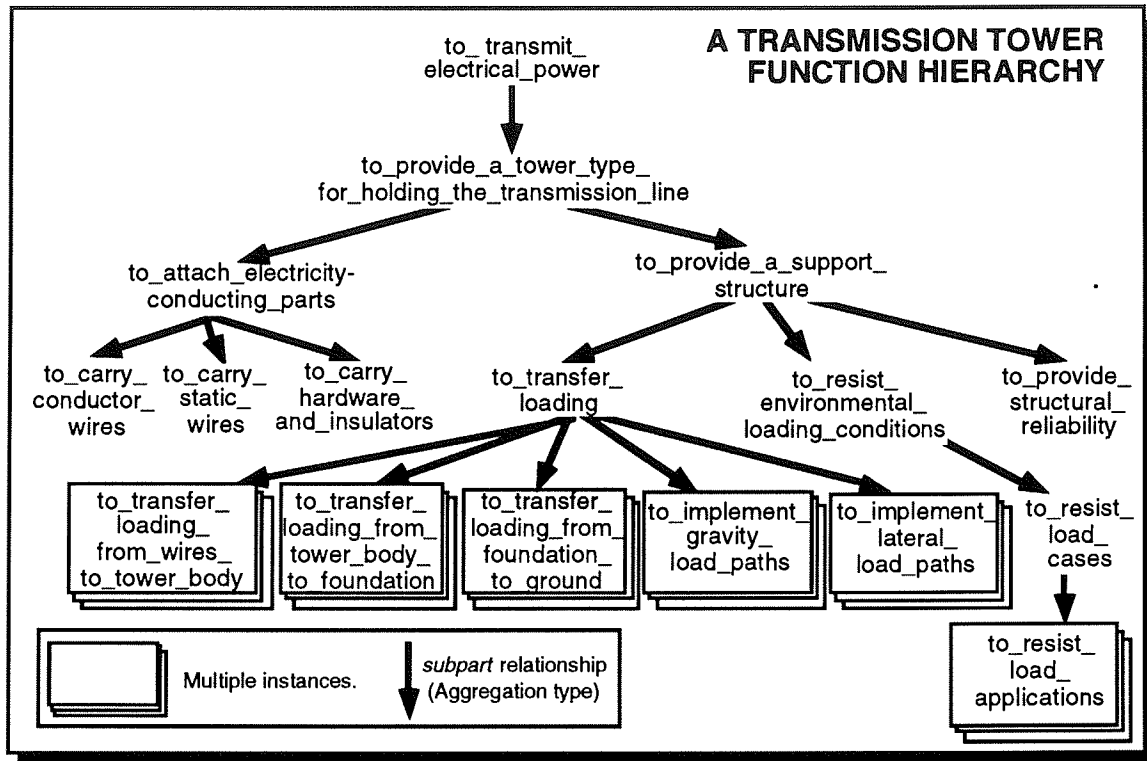
*The P-C Approach advocates that users of a domain primitive schema should have complete control over the hierarchical decomposition of a given facility. This decomposition should also closely reflect the opportunistic way in which facility designers come up with their design.* In fact, the approach does not impose any predefined aggregation levels by which the user must abide (e.g., the levels of building, systems, subsystems, members, elements, parts, connections, and connection methods described in the Structural Steel Framing Data Model [Lavakare 89]). The user can decompose a facility in two ways: by the functions that it performs, or by the design artifacts created by the engineers. The domain primitive schema provides primitive classes needed to support both types of decomposition. The following sections explain the two decomposition types using the example of transmission towers.

#### **7.3.1 Decomposition by Functions**

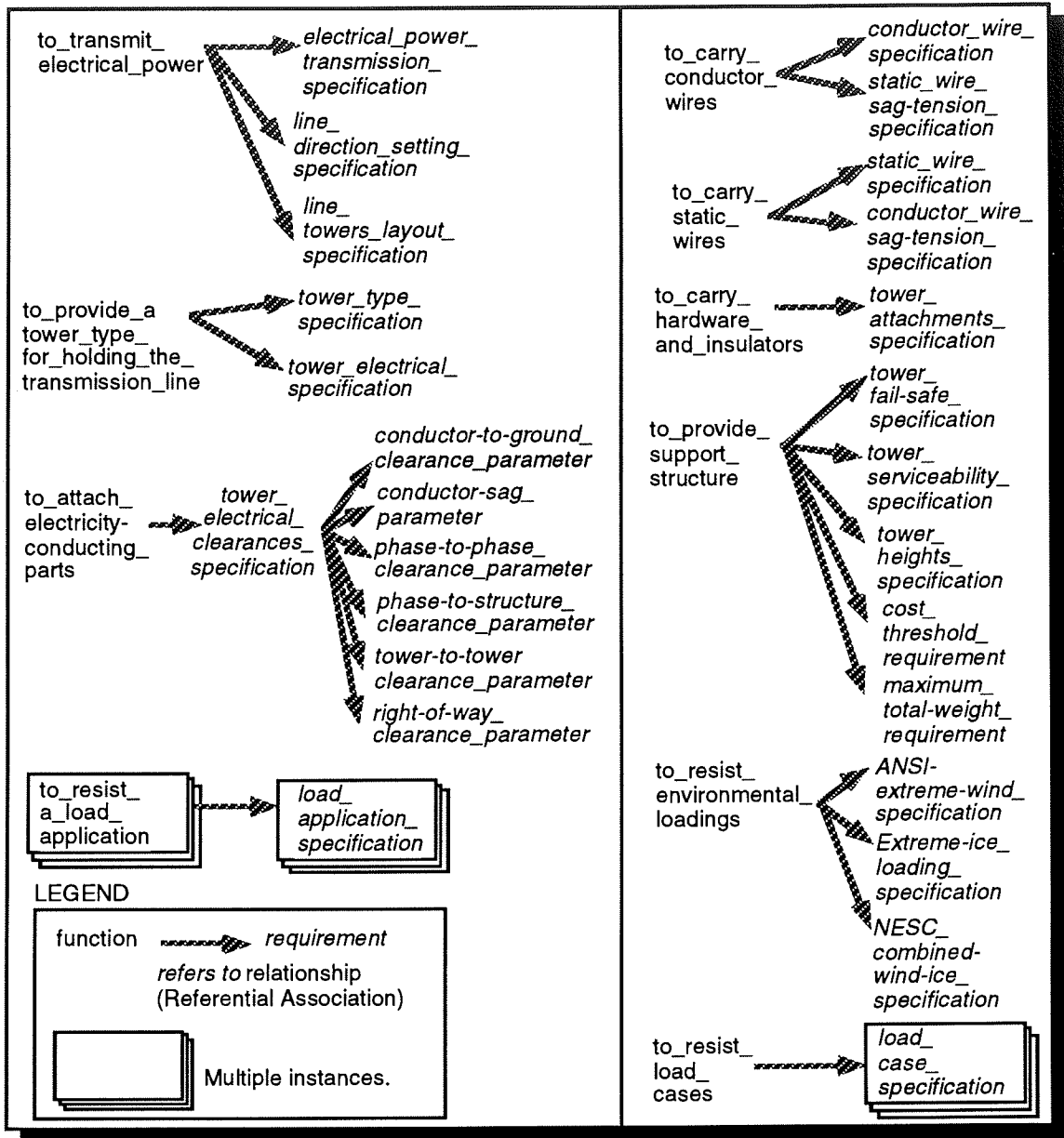
A facility can be seen as a hierarchy of the functions that it performs. Each function can include other supporting functions. Figure 7.15 shows the “function hierarchy” of the sample transmission tower illustrated in Figure 1.1 of Chapter 1.

This decomposition type provides a hierarchical view that is important to the design of a facility. In fact, *designing a facility is about defining, planning, implementing and validating its functions.* In other words, the designer reasons about the functions the facility must perform and how the functions of its systems and components can be aggregated to achieve the overall functions of the facility.

In addition, a function hierarchy provides a framework for defining and organizing the requirements for designing and constructing the facility. The modeler can associate each function with a set of requirements for achieving it. Figures 7.16 show the requirements associated with the sample tower’s functions in Figure 7.15.



**FIGURE 7.15: A Sample Transmission Tower's Function Hierarchy.**



**FIGURE 7.16: A Sample Transmission Tower's Functions and Requirements.**

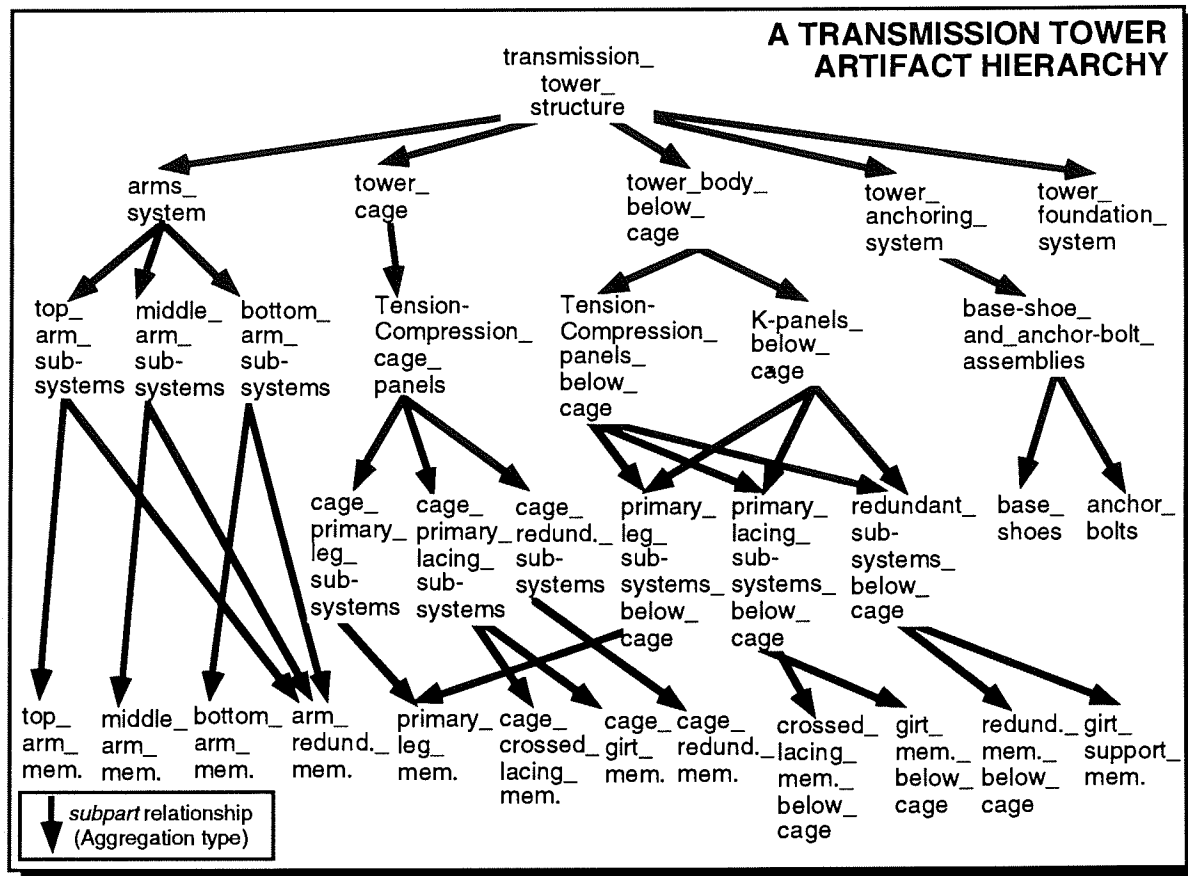


### 7.3.2 Decomposition by Design Artifacts

This decomposition type yields a hierarchical view of a facility in terms of the objects created by the designer, the “artifacts.” Figure 7.17 shows a sample “artifact hierarchy” of the transmission tower seen in the previous section. An artifact hierarchy is similar to a solution hierarchy that parallels a function hierarchy.

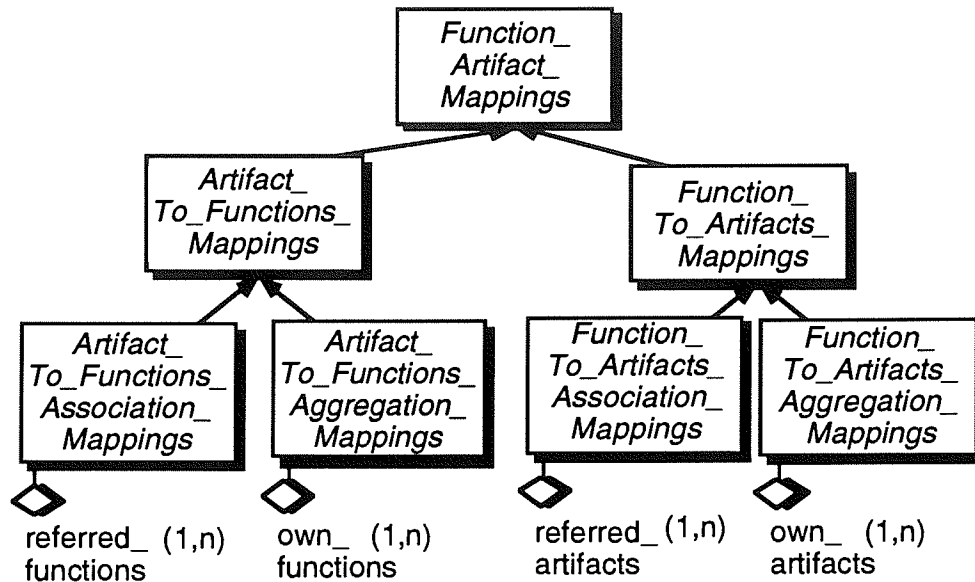
The reader must make the following distinctions:

- *Design artifacts vs. physical objects:* As mentioned earlier, design artifacts can be abstract. They are not all physical objects with well-defined boundaries or made from certain material. A facility decomposition by design artifacts can yield a result different from that of a decomposition based on bounded physical objects of the facility.



**FIGURE 7.17: A Transmission Tower’s Artifact Hierarchy.**

The elements shown in this hierarchy have more than one instance, except those at the two highest levels. Each element has its unique design characteristics, functions, and requirements. The arrows indicate “subpart” relationships of the Aggregation type. For the purpose of simplicity, this figure does not show tower extensions, the tower foundation branch, connection assemblies, connections, and parts.



**FIGURE 7.18: Sample Function-Artifact Mapping Primitive Classes.**

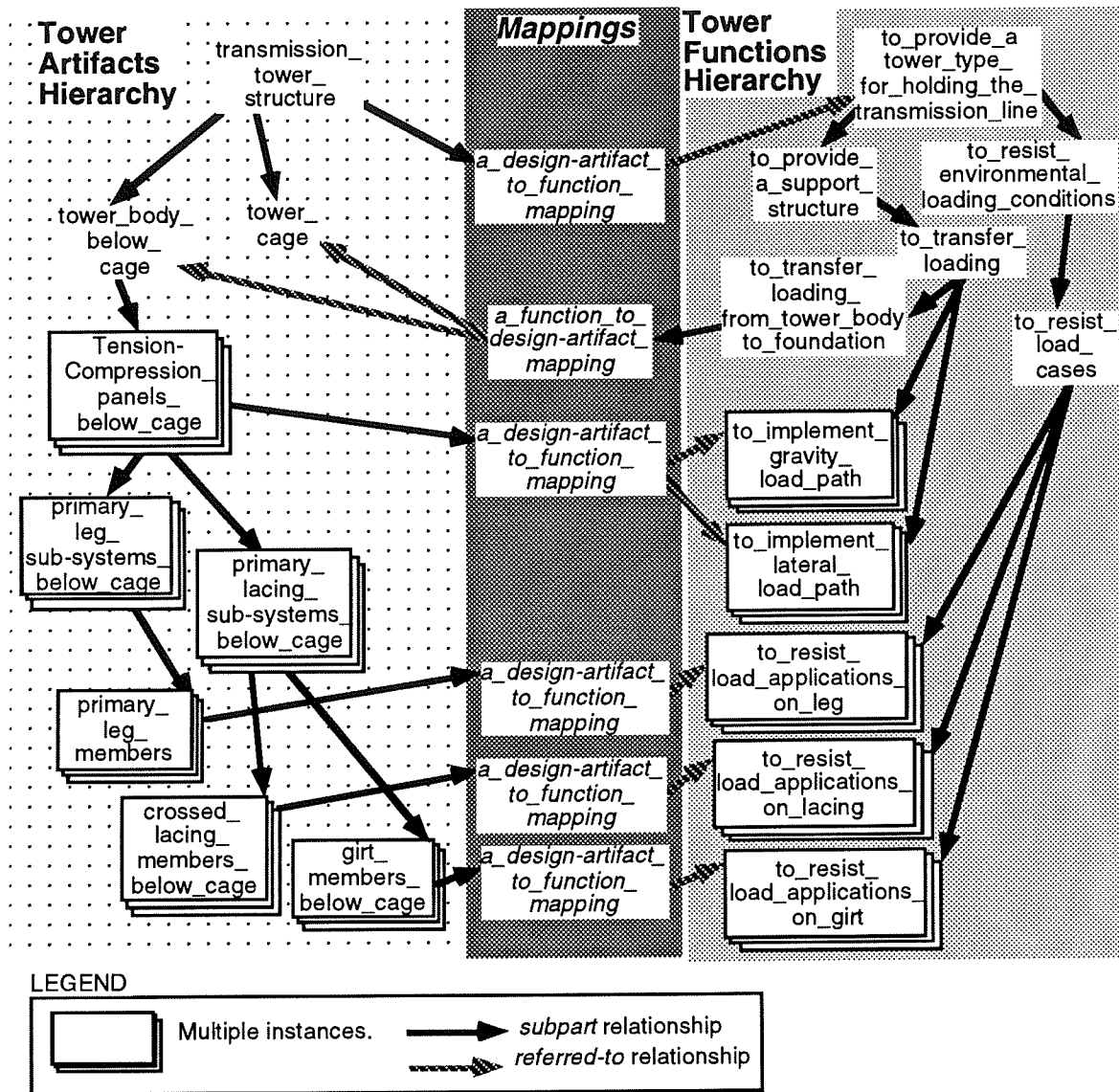
- *Decomposition by artifacts vs. decomposition by functions:* Figures 7.15 and 7.17 show that the application of the two decomposition types to the same transmission tower can yield completely different results. A *design artifact obviously is not a function*. However, an artifact can be seen as a design solution to functions in a function hierarchy.

### 7.3.3 Mappings Between Functions and Artifacts

A domain primitive schema should provide primitive classes that support mappings between a function hierarchy and an artifact hierarchy. The user of the schema can use these classes to define her own mapping instances for a given facility to represent closely the way in which designers come up with the design of that facility. Figure 7.18 shows the “function-artifact mapping” primitive classes. These classes present the user with three alternatives for creating facility hierarchies: creating dual artifact and function hierarchies, creating an aggregated artifact-function hierarchy, and creating mixed artifact-function hierarchies.

**First Alternative—Dual Artifact and Function Hierarchies** Using the “Artifact-to-Function Association Mappings” and “Function-to-Artifact Association Mappings” primitive classes in Figure 7.18, the user can create dual hierarchies with the following characteristics: The user can decompose a facility by its functions independent of its physical characteristics and design artifacts. Alternatively, she can decompose it by its design artifacts independent of its functional aspects. The resulting function hierarchy and artifact hierarchy do not have to be identical or correspond to each other in a one-to-one fashion. The mappings between the two hierarchies can go in either direction (i.e., from the artifact hierarchy to the function hierarchy or vice versa) to closely represent the way in which the design has occurred. Since these mappings use relationships of the Referential Association type, changes made to one hierarchy (i.e., additions, modifications, and deletions of elements in the hierarchy) do not affect the other hierarchy. However, when a function or artifact is deleted, it is the user’s responsibility to

clean up the hanging references that are associated with the deleted instance. Figure 7.19 shows an example of such hierarchies for a transmission tower.



**FIGURE 7.19: Dual Artifact and Function Hierarchies of A Transmission Tower.**

**Second Alternative—Aggregated Artifact-Function Hierarchy** Using the “Artifact-to-Function Aggregation Mappings” and “Function-to-Artifact Aggregation Mappings” primitive classes in Figure 7.18, the user can create a single hierarchy that contains both artifacts and functions. This aggregated hierarchy has the following characteristics: First, the hierarchy shows how functions of the facility break down and, at the same time, describes what artifacts designers create. In this alternative, functions and design artifacts are “cohorts” of the same hierarchy. More importantly, the hierarchy captures a different semantic of the relationship between functions and design artifacts. Artifacts and functions of the hierarchies are linked together using relationships of aggregation type. Consequently, deletion of an object at one level of the hierarchy will automatically trigger the deletion of all other component objects of the lower levels, whether these objects are functions or artifacts. The advantage of this approach over the first alternative is that when a function or artifact is deleted, the user does not have to clean up any “hanging” references associated with the deleted instance (i.e., references that point to nothing as a result of the deletion). Another advantage is that the user can introduce new functions and design artifacts at any level of the hierarchy. This approach is useful when a complete decomposition of the facility by either artifacts or functions is not feasible or too complex to define, especially at the detail levels of the hierarchy. The tradeoff is that the modeler gives up the clean separation between the functions and the artifacts of the facility.

**Third Alternative: Mixed Artifact-Function Network** The third alternative is a hybrid of the first two alternatives. It allows both aggregation and association mappings between design artifacts and functions at any level of the hierarchy. Therefore, it provides the modeler with a lot of flexibility in decomposing a facility. The trade-off is that since both aggregation and association relationship types are used to create a highly complex network, the modeler must be aware of what relationships exist between function and design artifact instances and what effects these relationships entail when instances are deleted.

## **7.4 Chapter Summary**

---

This chapter focuses on the content of a domain primitive schema following the P-C Approach, namely the representation elements that the schema should include. First, form, function and behavior have been discussed in literature on knowledge representation and design knowledge capture. In this research, form, function, and behavior of design objects for transmission towers are all represented in a detailed schema. The inclusion of form, function, and behavior representations in this schema was an attempt to overcome the deficiencies of existing schemata in facility engineering that were pointed out in Chapters 1 and 2. This schema includes 216 primitive classes that are organized into 30 primitive characterization hierarchies. The sample hierarchies (e.g., material properties, functions, requirements, strength behavior, serviceability behavior) shown in this chapter come from the schema for the transmission tower domain. The separation of these hierarchies results in clean and modular facility data representations. In fact, each hierarchy uses a single criterion to define the primitive classes and thus provides a homogeneous view about one specific aspect of the facility design objects. In addition, the schema provides primitive classes representing important elements of a design such as design artifacts, features, parameters, constraints, versions, and alternatives. Together, form, function, behavior and design primitive classes of the schema support three principal dimensions of a facility design: (1) design characterization (i.e., form, function, behavior), (2) design decomposition and recomposition, and (3)

design versioning and alternative selection. With this schema, a facility can be decomposed hierarchically in two ways: by the functions that it provides and by the design artifacts created by the designers. The schema also includes primitive classes needed to support the two decomposition types, as well as the mappings between functions and artifacts. Appendix D contains the documentation of all the classes of the schema.

The last part of this dissertation describes the testing of the P-C Approach, the contributions of this research, and the final conclusions. In particular, the next chapter explains the testing of this approach in the transmission tower domain and presents the findings from that testing. The next chapter also shows how the tower primitive schema described here was used to represent different user views in a variety of tasks spanning from the conceptual design to the facility management phase in the tower life-cycle.

# **PART III: TESTING, CONTRIBUTIONS, AND CONCLUSIONS**

## **Chapter 8**

---

---

### **Testing of the P-C Approach**

#### **Chapter Abstract:**

This chapter explains the testing of the P-C Approach in the transmission tower domain and presents the findings from that testing. The chapter first introduces the main purpose and a complete definition of the testing. Next, it presents the research question and hypothesis and describes the test variables, measurements, cases, and procedure. Finally, it presents and discusses the test results.

#### **Organization:**

##### *8.1 Introduction*

##### *8.2 Testing of the P-C Approach*

###### *8.2.1 Research Question*

###### *8.2.2 Research Hypothesis*

###### *8.2.3 Independent and Intermediate Variables and Their Measurements*

###### *8.2.4 Key Dependent Variable and Its Measurements*

###### *8.2.5 Test Cases*

###### *8.2.6 Test Procedure*

###### *8.2.7 Test Results*

##### *8.3 Chapter Summary*

## **8.1 Introduction**

---

The main purpose of the testing described here was to verify the hypothesis that the application of the P-C Approach to a given facility engineering domain results in a schema that can be shared within that domain. The testing was conducted in the transmission tower domain using the schema already developed for this domain. The testing was based on social science research methodology. This methodology provided the basis to describe the research work, formalize the testing, and analyze the outcome of the testing [Lave 75], [Eisenhardt 89], [Babbie 92]. Using this methodology, a complete definition of the testing included:

- A general question that guided the research;
- A research hypothesis that the testing would verify or disprove;
- All test variables, including independent, dependent and intermediate variables;
- The test measurements, namely the values with which the test variables are measured;
- The specific test cases that were conducted: A test case corresponds to a unique combination of input variable values that is chosen prior to the test. All of the possible combinations constitute the total “test space.” Each test case is a point within that space.
- The test procedure, namely the steps for obtaining test results that are the measured values of the dependent variables.

The testing also led to the evaluation of the scope of applicability, strengths, and limitations of the P-C Approach. This evaluation was presented in Chapter 3.

## **8.2 Testing of the P-C Approach**

---

### **8.2.1 Research Question**

The general research question is:

*What is needed to model data in a given facility engineering domain and as a result, to build a schema that can be shared by multiple users during the life-cycle phases in that domain (and thereby, can support data integration)?*

### **8.2.2 Research Hypothesis**

The hypothesis that was tested here is:

*A methodology for modeling data in a given facility engineering domain can produce a schema that can be shared by multiple users across the life-cycle phases in that domain.*

This thesis already described such a methodology, the P-C Approach, and its application to a tower domain that resulted in a schema. The testing described here focuses on measuring how well this schema can be used to support data uses by multiple users across the life-cycle phases and, thus, how well it can be shared within the domain.

### 8.2.3 Independent and Intermediate Variables and Their Measurements

The above hypothesis suggests three obvious variables:

- “Facility Engineering Domain:” *A facility engineering domain corresponds to a product from the A/E/C industry.* This independent variable has many possible values: electrical utility transmission structures, steel-framed warehouses, reinforced-concrete buildings, etc. These values are called “nominal” measurements because they tend to be exhaustive and mutually exclusive [Babbie 92].
- “Data Modeling Methodology:” *A data modeling methodology is a collection of techniques and tools (including concepts, rules, guidelines, procedures, and operations) that can be used to model data and that leads to the development of a schema.* A schema is a description of the way in which the data is structured. This independent variable has many nominal values: the P-C Approach, the Nijssen’s Information Analysis Method (NIAM) [Verheijen 82], [Williams 89], and the Integrated Computer Aided Manufacturing Definition (IDEF) methodologies [Bravoco 85a], [Bravoco 85b], [Mayer 92] to name a few.
- “Domain Schema:” *The use of a methodology to model data in a domain yields a schema for that domain or a “domain schema.”* As Figure 8.1 shows, this intermediate variable depends on both the domain and methodology under consideration. Therefore, its value varies according to the input domain and modeling approach.

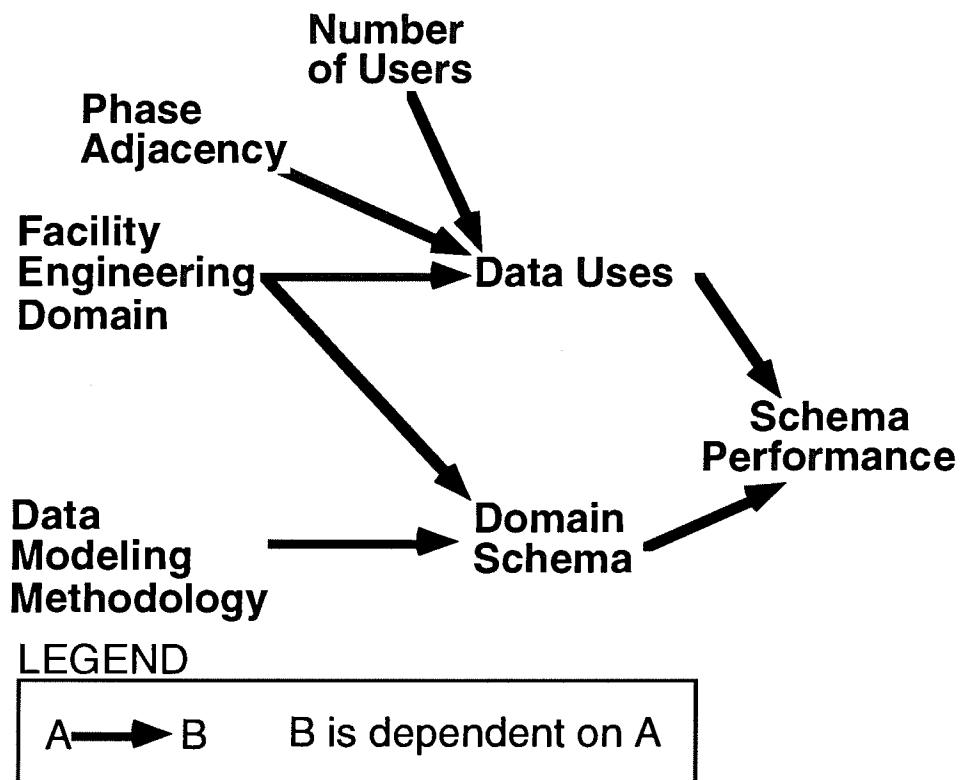


FIGURE 8.1: All Test Variables.

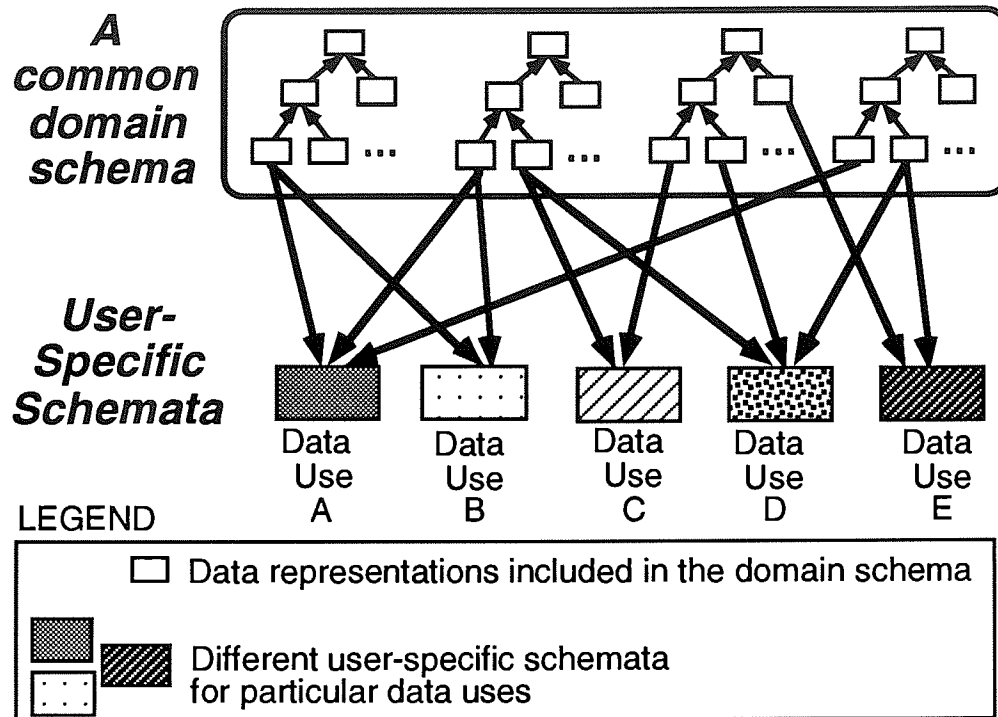


Figure 8.1 shows three other variables:

- “Data Uses:” *A data use refers to a specific occurrence in which a fixed set of data items are used by a user to carry out an activity. A user is defined here as a person or computer application who carries out an activity and needs to use data. This variable has the following nominal values in the transmission tower domain: electrical design, conceptual structural design, structural analysis, detailed structural design, structure detailing, fabrication, construction planning, construction execution, facility management, and other. These nominal values are defined from studying the different categories of activities in the tower facility life cycle (described in Appendix A). The next two variables influence the selection of data uses considered in the testing.*
- “Number of Users:” *This variable indicates whether a single user or multiple distinct users are considered in a single test case. This variable has two values: “single user” and “multiple users.” These are “ordinal” measurements, which are ordered using some logical ranking.*
- “Phase Adjacency:” *This variable represents the closeness of the phases in which the data uses being considered occur. This variable has three ordinal values: same phase, consecutive phases, and non-consecutive phases.*

#### **8.2.4 Key Dependent Variable and Its Measurements**

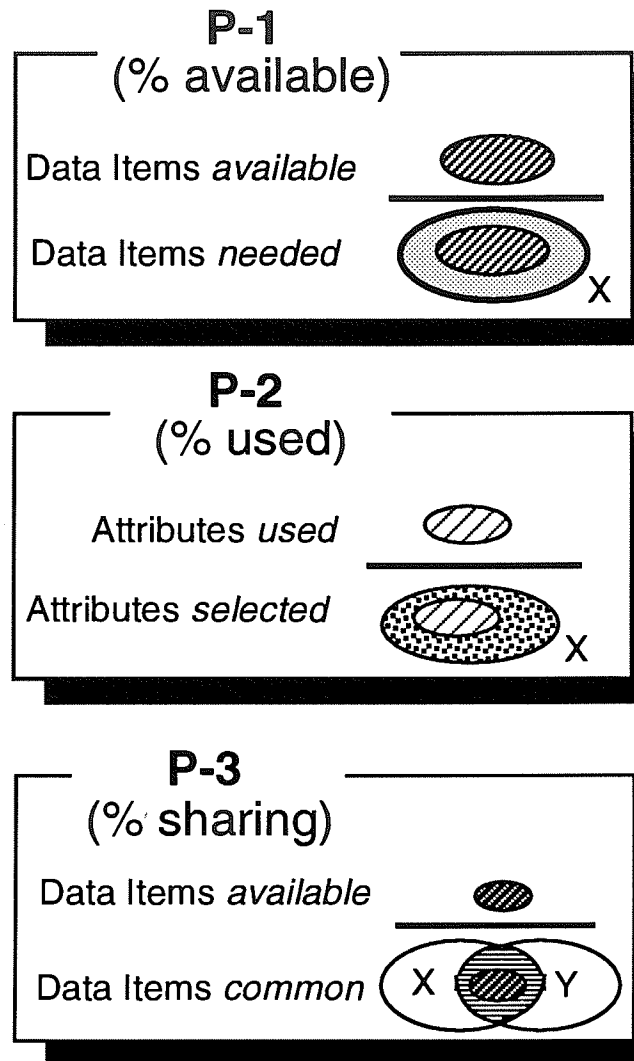
As pointed out in Chapter 2, little work has been done in the area of defining requirements, criteria, measures to test and evaluate data models, database schemata, and data standards. To our knowledge, no rigorous measures have been defined for testing how effectively a data standard or schema developed to support integration can be shared within a domain and across domains and thus, for accepting that standard or schema. This research takes the first step toward defining such measures by introducing a variable, “Schema Performance,” and its three components: P-1 (or schema completeness), P-2 (or schema efficiency), and P-3 (or schema sharing). Schema Performance, the key dependent variable, depends on the individual domain schema and the data uses considered in the testing. *Schema Performance measures how well a domain schema can be used and in this case, can be shared by multiple users and across life-cycle phases in that domain. Specifically, Schema Performance measures the ability to generate specific schemata from the given domain as they are needed by the individual users in various life-cycle phases.* This is illustrated in Figure 8.2.



**FIGURE 8.2: An Illustration of Generation of User-Specific Schemata from a Common Domain Schema.**

The Schema Performance includes three components, as Figure 8.3 illustrates:

- P-1 (or Schema Completeness): The percentage of data items available from the schema to generate user-specific schemata for different data uses. To measure P-1, let X be a data use considered in the test. First, determine the number of data items needed (or  $I_{\text{needed}}$ ) for each data use X. Then, determine the number of data items available from the schema (or  $I_{\text{available}}$ ). Calculate P-1 as the average ratio of  $I_{\text{available}}$  over  $I_{\text{needed}}$  for all the data uses considered in the test. For example, one activity is considered in the test. That activity needs ten data items, and five are available from the schema. Thus, P-1, the percentage of data available, is 50%. P-1 indicates how complete the schema is.*
- P-2 (or Schema Efficiency): The percentage of attributes from the schema that are actually used to support the data uses. To measure P-2, first determine the number of attributes of primitive classes selected from the schema ( $A_{\text{selected}}$ ) for each data use X. Then, determine the number of attributes that are actually used to represent X ( $A_{\text{used}}$ ). Calculate P-2 as the average ratio of  $A_{\text{used}}$  over  $A_{\text{selected}}$  for all the data uses considered in the test. For example, a primitive class with seven attributes is selected, and five are needed for the composite object. In this case, P-2, the percentage of attributes used, is 71%. P-2 indicates how efficient the schema is (how efficiently attributes in the schema are grouped together to support the data uses).*



**FIGURE 8.3: Three Components of the Schema Performance Variable.**

- P-3 (or Schema Sharing): P-3 captures the percentage of data shared by the data uses that are available from the schema. To measure P-3, let X and Y be a pair of distinct data uses considered in the test. First, determine the number of data items common to X and Y (or  $I_{\text{common}}$ ) for each pair. Then, determine the number of data items available from the schema to represent those common data items (or  $I_{\text{available}}$ ). Calculate P-3 as the average ratio of  $I_{\text{available}}$  over  $I_{\text{common}}$  for all the pairs of distinct data uses considered in the test. For example, X and Y share eight data items, four of which are available from the schema. In this case, P-3, the percentage of shared data available, is 50%. P-3 indicates the extent to which the schema promotes sharing of data among data uses.*

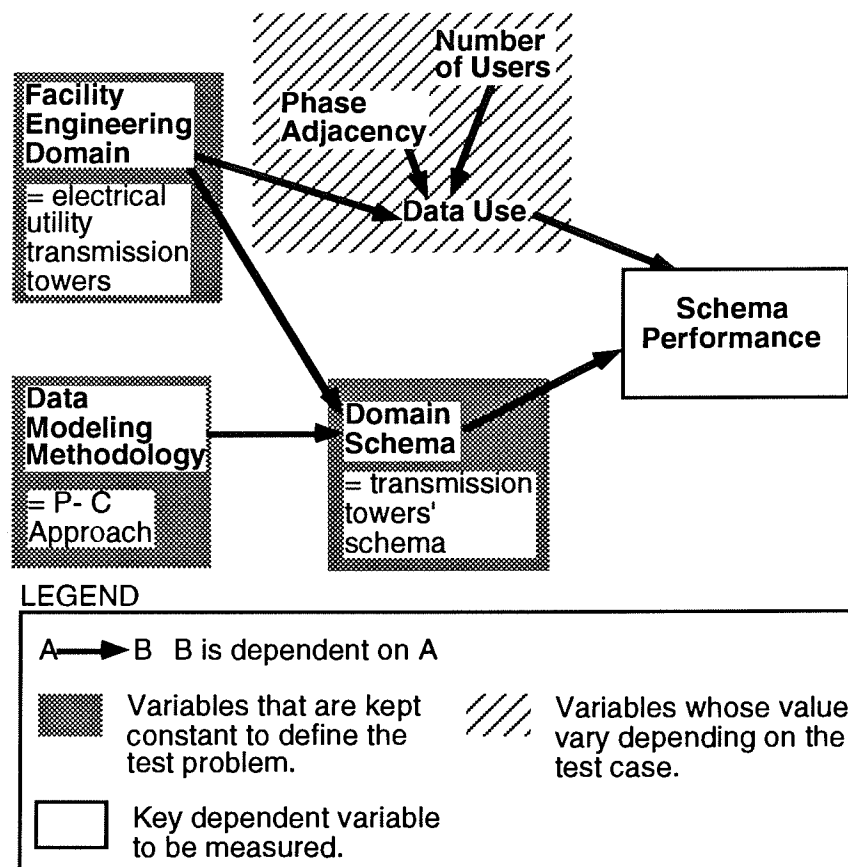
This testing suggests the combination of the above three components as a way to measure how well a schema can be used to support data uses by different users and across life-cycle phases and, thus, how well it can be shared within the domain. Meanwhile, other performance aspects of a schema such as flexibility, extensibility, accuracy, etc. were considered. However, operational measurements for those aspects were found to be very difficult to define. Additional research work is needed to further define all the essential component measurements of schema performance.

## 8.2.5 Test Cases

### 8.2.5.1 Description of the Main Test Problem and Cases

A test problem was defined through the selection of a facility engineering domain and a data modeling methodology. Figure 8.4 illustrates the main test problem that is considered here: The facility engineering domain is electrical utility transmission towers, and the data modeling methodology is the P-C Approach. This approach was applied to the tower domain to build a domain schema. In this test problem, test cases were selected by varying the values of the data uses, number of users, and phase adjacency. Conducting the selected test cases resulted in measurements of the Schema Performance variable.

Six test cases covered the test space defined by the two variables, Number of Users and Phase Adjacency. As Table 8.1 shows, each of these cases corresponded to a unique combination of values of those two variables. The resulting set of six cases tested the schema performance from a single user to multiple users and from the same phase to non-consecutive phases. This set considered the test subject of transmission tower members. (A test subject is a topic to which the data uses specified in the test case relate.) In these test cases, data uses were selected to cover a variety of activity types and, to as large a degree as possible, the available test space. In fact, these data uses came from specific activities of the tower engineering process that are described in Appendix A. The next section describes the selected data uses in detail.



**FIGURE 8.4:** The Main Test Problem.

**TABLE 8.1: The Selected Data Uses Selected in the Six Test Cases.**

<b>Number Of Users vs. Phase Adjacency</b>	<b>Single User</b>	<b>Multiple Users</b>
<b>Same Phase</b>	<ul style="list-style-type: none"> <li>• Data Use 1-A: <i>Tower Members as Analyzed</i> (structural engineer, Phase III)</li> <li>• Data Use 1-B: <i>Tower Members as Designed</i> (structural engineer, Phase III)</li> <li>• Data Use 1-C: <i>Tower Members as Bolted</i> (structural engineer, Phase III)</li> </ul>	<ul style="list-style-type: none"> <li>• Data Use 4: <i>Tower Members as Anchored</i> (foundation engineer, Phase III)</li> </ul>
<b>Consecutive Phases</b>	<ul style="list-style-type: none"> <li>• Data Use 2: <i>Tower Members as Conceptualized</i> (structural engineer, Phase II)</li> </ul>	<ul style="list-style-type: none"> <li>• Data Use 5-A: <i>Tower Members as Detailed</i> (structural detailer, Phase IV)</li> <li>• Data Use 5-B: <i>Tower Members as Delivered</i> (fabricator, Phase V)</li> <li>• Data Use 5-C: <i>Tower Members as Assembled</i> (construction crews, Phase V)</li> </ul>
<b>Non-consecutive Phases</b>	<ul style="list-style-type: none"> <li>• Data Use 3: <i>Tower Members as Redesigned</i> (structural engineer, Phase VI)</li> </ul>	<ul style="list-style-type: none"> <li>• Data Use 6: <i>Tower Members as Checked</i> (electrical engineer, Phase IV)</li> </ul>

### 8.2.5.2 Selection of Data Uses

**Considering Single User, Same Phase: Selecting the First Three Data Uses, 1-A to 1-C** First, three uses of data describing the tower members were selected. These data uses involve a single user, the structural engineer, in the same Phase III, Tower Structural Detailed Design. However, they occur in three separate activities:

- preparing the input data to the structural analysis program (see Activity III.S1.3 in the graphical functional schemata in Appendix A),
- designing the members (see Activity III.S2.1) and
- designing the members' end connectors (i.e., by determining the number, pattern and diameter of bolt holes) (see Activity III.S2.3).

These activities use the member data differently and provide three ways of viewing the tower members: as analyzed, as designed, and as bolted.

**Considering Single User, Consecutive Phases: Selecting Data Use 2** This data use by the same user, the structural engineer, occurs in Phase II, Tower Structural Conceptual Design. Here, the structural engineer assumes a preliminary geometry for the tower by arranging tower systems and configuring their members (see Activity II.S1.2). Only the members' position, orientation and topology are determined at this point. The new data use yields another view of the tower members: as conceptualized.

**Considering Single User, Non-Consecutive Phases: Selecting Data Use 3** This data use by the same user occurs in Phase VI, Tower Facility Management. Here, the structural engineer redesigns the tower members to add new electrical equipment to the tower structure. This data use yields another view of the tower members: as redesigned.

**Considering Multiple Users, Same Phase: Selecting Data Use 4** This data use in the same Phase III involves a different user, the foundation engineer. Here, the foundation engineer uses the data of selected members to design the tower anchoring devices (see Activity III.G4.2). (These anchoring devices are not part of the members.) The new data use yields another view of the tower members (i.e., as anchored) by a different user.

**Considering Multiple Users, Consecutive Phases: Selecting Data Uses 5-A, 5-B and 5-C** More data uses from other users were chosen. To cover the test space as much as possible, three data uses (i.e., by the structure detailer, fabricator, and construction crews) were selected. This time, the corresponding activities occur in consecutive phases, Tower Construction Planning (Phase IV) and Tower Construction Execution (Phase V). The two new activities are:

- the detailer detailing of the fabrication parts (see Activity IV.S2.1),
- the fabricator bundling and shipping of parts to the construction site (see Activity V.F1.5), and
- the construction crew assembling the tower members on site (see Activity V.C3.1).

These provide views of the tower members by two other users: as detailed, as fabricated, and as assembled.

**Considering Multiple Users, Non-Consecutive Phases: Selecting Data Use 6** This data use by another user, the electrical engineer, occurs in Phase VI, Tower Facility Management. Here, the electrical engineer checks the tower members for electrical clearances (Phase VI). This provides another view of the tower members: as checked.

### **8.2.6 Test Procedure**

We defined the test procedure and carried out the entire test using the tower domain primitive schema already developed. The procedure was as follows:

1. We determined the data items needed in each of the selected data uses by reviewing the relevant collected documentation. (The latter included project design folders, design manuals, engineering drawings, bills of materials, program input and output files, engineering drawings, etc.)

2. For each data use, we divided the data items into two sets: those that are available from the given primitive schema and those that could not be represented using the schema.
3. For the first set of data items of each data use, we selected the appropriate primitive classes from the schema and defined a composite schema for the data use following the standard steps provided by the P-C Data Modeling Method. (Those steps are described in Section 6.3.4 of Chapter 6). In the primitive class definitions, we eliminated any attributes that were not needed.
4. Finally, assuming that all data items are equally important, we measured the three components of the Schema Performance variable that were defined in Section 8.2.4.

### 8.2.7 Test Results

Tables 8.2 and 8.3 summarize the measurements of the tower primitive schema's performance: 98% data available (P-1), 84% attributes used (P-2), and 83% schema sharing (P-3). The composite classes that were defined for the tested data uses are documented in Appendix E. Those composite classes represent the different user views of the tower members such as "Tower Members As Analyzed," "Tower Members As Designed," "Tower Members As Bolted," etc. Table 8.4 shows the primitive classes that were selected from the tower schema to customize those composite classes.

**TABLE 8.2: Measurements for P-1 and P-2 of the Tower Primitive Schema.**

<b>Data Uses</b>	<b>Views of Tower Members</b>	<b>I<sub>needed</sub></b>	<b>I<sub>available</sub></b>	<b>A<sub>selected</sub></b>	<b>A<sub>deleted</sub></b>
<b>1-A</b>	As Analyzed	39	38	48	10
<b>1-B</b>	As Designed	54	53	65	12
<b>1-C</b>	As Bolted	47	46	52	6
<b>2</b>	As Conceptualized	52	51	58	7
<b>3</b>	As Redesigned	103	102	115	13
<b>4</b>	As Anchored	50	49	59	10
<b>5-A</b>	As Detailed	52	51	54	3
<b>5-B</b>	As Delivered	18	16	21	5
<b>5-C</b>	As Assembled	41	40	51	11
<b>6</b>	As Checked	48	47	50	3

**$P-1$  (or % data available) = Average (I<sub>available</sub> / I<sub>needed</sub>) = 97%**

**$P-2$  (or % attributes used) = Average (A<sub>used</sub> / A<sub>selected</sub>) = 85%**

**TABLE 8.3: Measurements for P-3 of the Tower Primitive Schema.**

Data	Data Items Available (or Iavailable)			Common Data Items (or Icommon)						
	1-A	1-B	1-C	2	3	4	5-A	5-B	5-C	6
1-A										
1-B	13 14					This	upper	triangle	is	the
1-C	23 24	32 33				transpose	of	the		
2	7 8	13 14	9 10			lower	half	triangle		
3	31 32	53 54	46 47	32 33						
4	22 23	35 36	45 46	9 10	49 50					
5-A	0 1	9 10	9 10	6 8	25 26	9 10				
5-B	0 1	8 9	8 9	4 5	12 13	8 9	16 17			
5-C	0 1	8 9	8 9	16 18	21 22	8 9	27 28	16 17		
6	0 1	5 6	5 6	15 16	15 16	5 6	9 10	4 5	4 5	

**P-3 (or % schema sharing) = Average (Iavailable / Icommon) = 83%**



**Table 8.4: Customization of Composite Classes for Data Uses Considered in the Test.**

<b>Data Use</b>	<b>1-A</b>	<b>1-B</b>	<b>1-C</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5-A</b>	<b>5-B</b>	<b>5-C</b>	<b>6</b>
Reference Activity (or Phase) No. in Appendix A	III.S1.3	III.S2.1	III.S2.3	II.S1.2	Phase VI	III.G4.2	IV.S2.1	V.F1.5	V.C3.1	Phase VI
Composite Class Defined → Primitive Classes Used ↓	Tower Members as Analyzed	Tower Members as Designed	Tower Members as Bolted	Tower Members as Concept.	Tower Members as Redesign.	Tower Members as Anchored	Tower Members as Detailed	Tower Members as Delivered	Tower Members as Assembl.	Tower Members as Checked
2-Node Analysis Elem. Descript.	•		•		•	•				
Structural Analysis Node Desc.	•		•		•	•				
Section Areas	•	•	•	•	•	•				
Material Moduli	•	•			•					
Load Case Specifications	•	•	•		•	•				
Static Concentrated Load Spec.	•									
At-Joint Load Application Spec.	•			•	•					
AISC L-Shape Descriptions		•	•	•	•	•	•	•	•	•
Static Concent. Response Forces		•	•		•	•				
Numerical Design Parameters		•	•		•	•	•	•	•	
Material Strength Properties		•	•		•	•				
Design Artifact Descriptions		•		•	•					
Behavior Stresses		•			•					

**Table 8.4 (cont.): Customization of Composite Classes for Data Uses Considered in the Test.**

<b>Data Use</b>	<b>1-A</b>	<b>1-B</b>	<b>1-C</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5-A</b>	<b>5-B</b>	<b>5-C</b>	<b>6</b>
Reference Activity (or Phase) No. in Appendix A	III.S1.3	III.S2.1	III.S2.3	II.S1.2	Phase VI	III.G4.2	IV.S2.1	V.F1.5	V.C3.1	Phase VI
Composite Class Defined → Primitive Classes Used →	Tower Members as Analyzed	Tower Members as Designed	Tower Members as Bolted	Tower Members as Concept.	Tower Members as Redesign.	Tower Members as Anchored	Tower Members as Detailed	Tower Members as Delivered	Tower Members as Assembl.	Tower Members as Checked
Section Inertias		•			•	•				
To Resist Load Applications				•	•					
Edge Descriptions				•					•	
Line Segments				•					•	
Vertex Descriptions				•					•	
Vertex Connectivities				•					•	
Cartesian Points				•			•		•	
Cartesian Coordinate Systems				•	•					•
Cartesian Transformations				•	•					•
Fabrication Dimensions					•		•	•	•	
Bolt Hole Patterns					•		•		•	
Fabrication Locations					•		•		•	
Fabrication Quantities							•	•	•	

**Table 8.4 (cont.): Customization of Composite Classes for Data Uses Considered in the Test cases.**

<b>Data Use</b>	<b>1-A</b>	<b>1-B</b>	<b>1-C</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5-A</b>	<b>5-B</b>	<b>5-C</b>	<b>6</b>
Reference Activity (or Phase) No. in Appendix A	III.S1.3	III.S2.1	III.S2.3	II.S1.2	Phase VI	III.G4.2	IV.S2.1	V.F1.5	V.C3.1	Phase VI
Composite Class Defined → Primitive Classes Used ↓	Tower Members as Analyzed	Tower Members as Designed	Tower Members as Bolted	Tower Members as Concept.	Tower Members as Redesign.	Tower Members as Anchored	Tower Members as Detailed	Tower Members as Delivered	Tower Members as Assembl.	Tower Members as Checked
NC Mark Features							•			
NC Mark Texts							•	•	•	
Design Notes							•			
Design Graphics							•			
Member Layout Features							•			•
Member Clearance Features							•			
Cartesian Positions										•
Cartesian Orientations										•
Spatial Enclosures										•
Spatial References										•

For the purpose of comparison, we ran the test again but used a composite schema. To be conservative, we selected a composite schema that includes many primitive classes, namely the one defined for the data use 1-B (Tower Members As Designed). Further, we assumed that this schema has not been designed following the P-C Approach. Therefore, it consists of a single object class that includes a large number of attributes. These attributes represent all the data items needed in that data use. The results from this run are: 46% data available (P-1), 43% attributes used (P-2), and 58% schema sharing (P-3). The comparison between the two sets of results shows that a domain primitive schema has improved performance over a schema that is not designed following the P-C Approach.

In short, the testing described here shows that the tower domain primitive schema can effectively support data uses from multiple users for a variety of activities spanning from the tower conceptual design phase to the the tower facility management phase. *Consequently, this testing confirms that the schema can be shared within the tower domain and helps in verifying the hypothesis stated earlier.*

### ***8.3 Chapter Summary***

---

This chapter described the testing of the P-C Approach in the domain of transmission towers. Three schema performance components and their measurements were introduced: P-1 or schema completeness, P-2 or schema efficiency, and P-3 or schema sharing. In the testing, measurements of these components were obtained for the tower domain primitive schema. The test results showed that this schema supports a variety of data uses by different users across the life-cycle phases. Therefore, this testing verified that the application of the P-C Approach to a facility engineering domain produces a schema sharable within that domain, and consequently, verified the research hypothesis.



# Chapter 9

---

---

## Contributions, Conclusions, and Future Research

### **Chapter Abstract:**

The research described here focuses on the modeling of facility data to support data integration. Its goal is to better understand what is needed to carry out this task and to develop modeling tools that aid modelers. The immediate contribution of the research is a methodology, the Primitive-Composite Approach, for conducting this task in a given domain. Specifically, the research shows that the application of this approach to a real-life tower domain leads to the development of a schema shared by different users throughout the life-cycle phases in that domain. This schema itself shows that form, function, and behavior of facility design objects are all represented in a coherent fashion. This research also involves the testing of the P-C Approach in the tower domain. As a result, the strengths and limitations of this approach are clearly stated. This chapter presents the contributions, conclusions, and impacts of this research on data modeling and data exchange, as well as future research directions.

### **Organization:**

#### *9.1 Contributions*

9.1.1 Definition of the Modeling Requirements and Criteria

9.1.2 Development of the P-C Approach

9.1.3 Development of the Modeling Tools

9.1.4 Development and Implementation of the Test Domain Schema

#### *9.2 Conclusions*

*9.3 Impacts of the Research on Data Modeling and Data Exchange*

*9.4 Directions for Future Research*

*9.5 Final Remarks*

## **9.1 Contributions**

---

*In this research, we developed, applied, and tested a methodology for modeling facility data to support data integration.* The immediate contribution of this work is the methodology, the Primitive-Composite (P-C) Approach. This contribution can be explained in four parts:

- (1) Definition of the requirements and criteria for modeling facility data to support data integration,
- (2) Development, application, and evaluation of a methodology that can be used to analyze a given facility engineering domain and design a schema meeting those requirements,
- (3) Development of modeling tools that aid modelers in using this methodology, and
- (4) Development, testing, and demonstration of a detailed schema, as a result of applying this methodology to a test domain, that represents form, function, and behavior of design objects in that domain.

The following sections elaborate on these parts.

### **9.1.1 Definition of the Modeling Requirements and Criteria**

As pointed out in Chapters 1 and 2, the majority of existing schemata in facility engineering lack the flexibility to support multiple user views. Once defined, they cannot be extended to accommodate evolving life-cycle phases. *In this research, we incorporated sharability as requirements for the design of a primitive schema in a given domain.* In fact, we included two sharability requirements: user view support and schema extensibility. *To meet these requirements, we used cohesion and reusability to measure the qualities of primitive classes.* Although cohesion and reusability are rooted in software engineering, we showed that they can be applied effectively to data modeling. Further, we identified five principal dimensions of cohesion through studying data used by experts in a real-life facility engineering domain. We also defined five levels of reusability of primitive classes. *More importantly, the realization of the cohesion dimensions and reusability levels has changed the definition of primitive classes, which was initially based on intuition [Howard 92], into an operational definition with fewer and more measurable parameters than before.* Indeed, we defined a primitive class as a module that is “designed” to have maximum cohesion and reusability.

### **9.1.2 Development of the P-C Approach**

The original research intention was to develop a data model, the P-C Data Model, that supports data integration in facility engineering. However, the sheer complexity of facility engineering domains raised many issues. These issues mandated an understanding of the engineering process and the data describing the complex objects designed by experts in the domain. That understanding is indispensable to any data modeling task. Without it, any proposed data model has limited usefulness. These issues also pointed out the need for developing modeling tools to acquire the requisite understanding. Consequently, we developed the P-C Approach to resolve these issues. *This approach offers capabilities both to analyze a given domain and to design a schema that can be shared by multiple users and across the life-cycle phases in that domain.* We defined the necessary steps and developed modeling tools that aid modelers in using this approach.

These modeling tools also include the originally intended P-C Data model. In short, the development of the P-C Approach stemmed from a growing realization of what is needed to model real-life facility data. The fact that we developed a methodology rather than simply a data model makes this research a much more significant development work than was initially planned. In addition, we applied this approach to a tower engineering domain and evaluated its scope of applicability, strengths, and limitations.

### **9.1.3 Development of the Modeling Tools**

**Partitioned eEngineering DAta flow model (or PANDA)** The Data Flow model has been a popular choice for functional analysis of a variety of processes. However, *additional requirements were needed to analyze complex facility engineering processes, and no model had been developed specifically for doing this. We developed PANDA as a solution.* PANDA provides concepts, graphical representations, rules, schema transformation operations, and a customized method and guidelines for using the model. All these elements are necessary for the potential development of CASE tools that can assist modelers in doing functional analysis using PANDA. We introduced a unique partitioned architecture in PANDA, which helps modelers to organize their thinking about complex engineering processes and which enhances both the conceptual readability and graphical readability of the process' functional schema. We also used PANDA to model transmission tower engineering and produced a detailed set of functional schemata.

**Domain Entities AnaLysis method (or DEAL)** The Entity-Relationship model [Chen 76] is the accepted model for conceptual data modeling. It advocates using entities and relationships, but is deficient in providing criteria for entity definitions. This deficiency becomes more critical when the modeler deals with large complex engineering domains. As Wimmer and Wimmer [92] point out, *few methods had been developed for conceptual data modeling in engineering domains. We developed DEAL, a method that uses cohesion and reusability as direct criteria for analyzing data used by experts in a facility engineering domain and for doing conceptual modeling of that domain.* DEAL provides the terms and concepts, graphical representation, procedures, operations, and rules for carrying out the analysis.

**Primitive-Composite (or P-C) Data Model And Method** Object-oriented models and design methods are available today. However, *we developed a unique combination of an object-oriented data model and accompanying method.* First, this model and method work with the other modeling tools provided by the P-C Approach, namely PANDA and DEAL. Second, the model combines the notions of primitives and composites with object-oriented data modeling concepts, resulting in a stronger paradigm for modeling facility data to support data integration than those concepts alone. Third, the method provides a set of detailed rules and guidelines for designing a domain primitive schema or composite schemata based on the concepts of the model.

### **9.1.4 Development of the Test Domain Schema**

Existing schemata in facility engineering include mainly form descriptions and do not distinguish form, function, and behavior. In this research, *we showed that form, function, and behavior of design objects can all be represented in a detailed schema for a facility engineering domain.* This schema also includes primitive classes representing important elements such as design artifacts, features, parameters, constraints, versions, alternatives, etc., of a design. Previous work in facility data modeling did not take into



consideration this representation. Moreover, the modeling of facility data in the past has been limited to trivial academic exercises or, at most, to small and simple real-life domains. For the transmission tower domain, we defined approximately thirty primitive class hierarchies and more than two hundred primitive classes. To our knowledge, no such schema has previously been shown in facility engineering.

Previous facility data models impose hierarchical levels of aggregation such as building, systems, subsystems, components, parts, and connections for the decomposition of a facility. These levels are rigid and may not apply to a given facility. In this research, we suggested two flexible ways to decompose a facility: by the functions that the facility and components perform and by the design artifacts of the facility created by its designers. As explained in Chapter 7, we defined primitive classes for both types of decomposition. *In addition, we showed three alternatives for creating facility hierarchies using these classes: creating dual artifact and function hierarchies, creating a single aggregated artifact-function hierarchy, and creating mixed artifact-function hierarchies.*

As pointed out in Chapter 2, little work has been done in the area of defining requirements, criteria, and measures to test and evaluate data models, database schemata, and data standards. *In this research, we tested the tower primitive schema against the sharability requirements.* First, with this schema, we defined composite classes for a variety of data uses in activities across the tower life-cycle. In addition, we proposed three test measures: P-1 or schema completeness, P-2 or schema efficiency, and P-3 or schema sharing. In the testing, we obtained the P-1, P-2, and P-3 measurements for the tower primitive schema. As a result, we showed that this primitive schema has improved performance over a schema that was not designed following the P-C Approach. This testing is an important step toward using operational measures for the validation and acceptance of data exchange standards or schemata developed for the purpose of data integration. Finally, we used the primitive schema to create a database for a selected prototype tower and implemented the database using a commercial object-oriented database system (ONTOS).

---

## 9.2 Conclusions

---

The following paragraphs present our conclusions on the development of sharable object-oriented data representation for facility engineering integration.

**Identification of and Testing against Design Requirements** In building data representations to support integration, the modeler must treat these representations as a design artifact and must identify the design requirements at the beginning of the project. She must also define operational measures for testing the result against those requirements. These measures must include test variables with measurable values, and test cases and procedures. In particular, the potential for sharing data representations in a given domain depends on two critical factors: (1) the ability to support multiple customized user views from the common representations, and (2) the ability to extend these representations as the facility life-cycle evolves without abandoning previously defined representations. These yield two important requirements for the development of sharable data representations.

**Possibility of Defining “Primitives” as Sharable Data Representations** The notions of primitives and composites combined with object-oriented concepts provide a stronger paradigm for modeling facility data to support integration than the object-oriented concepts alone. Primitive classes represent the data shared by multiple users across the life-cycle phases. Composite classes customized by individual users represent complex views about the facility design objects. This research shows the possibility of

defining primitive classes in a given domain. This is accomplished by providing an operational definition of what they are and a methodology for designing them. In fact, the P-C Approach anchors its operational definition of a primitive class in the concepts of cohesion and reusability. Primitive classes need not be at the granularity of individual attributes, but rather are designed and optimized using the cohesion and reusability criteria. The P-C Approach essentially provides a structured methodology, including phases and modeling tools, for analyzing a given domain and designing a schema that includes primitive classes representing the domain.

***Incorporation of Design Criteria into Tools for the Development of Sharable Data Representations*** The modeler must define additional design criteria at the level of individual conceptual entities and, in the case of object-oriented data modeling, of logical object classes. He must incorporate these criteria directly into the tools used in the design of those conceptual entities or logical classes. The enforcement of these design criteria may help the modeler meet the requirements mentioned in the beginning. The P-C Approach advocates using cohesion and reusability to optimize the design of primitive classes, thereby increasing the sharability of the overall domain primitive schema.

***Functional Analysis prior to the Modeling of Data*** Modeling data without prior functional analysis of the process in which the data is used may be both arbitrary (i.e., difficult to justify) and problematic. It runs the risk of producing data representations that are hard to reuse, maintain, and share. Functional analysis provides that understanding and must be a prerequisite to the schema design. In addition, it helps the modeler improve the design requirements and design of the schema, and verify and test the schema.

In developing PANDA for doing functional analysis of facility engineering processes, we learned the following lessons: A proper model with graphical representations is critical to the successful modeling of the process. By “proper,” we mean that the model must be capable of representing multiple participants, non-linear subprocesses, design synthesis loops, decisions, alternatives, and interferences, and complicated data, material and product flow networks. These are essential characteristics of facility engineering processes. In addition, this model must have built-in features that will automatically produce highly readable graphical descriptions of the process. Without these properties, the model may not be used. A method and guidelines for using a suggested model also improves the likelihood that the model will be used effectively. Finally, CASE tools that assist the modeler in doing functional analysis using a suggested model are necessary and will make possible the successful use of the model.

***Representation of Form, Function, and Behavior in Facility Engineering*** Facility databases usually store data about physical properties of the facility design objects. Conventional engineering drawings that are commonly used to communicate the facility design show only the physical descriptions of the facility and its components. These descriptions alone are inadequate to communicate the facility design. An object-oriented schema supporting data integration in a domain must include object classes representing form, function, and behavior of facility design objects. Their instances for a given facility must be stored in a facility engineering project database or databases. The explicit representation of form, function, and behavior in this schema enhances both the description of the design objects and the explanation of their design. Moreover, form, function, and behavior classes can be organized into separate class hierarchies in the schema. This separation results in clean and modular representations that can easily be reused and shared. In addition, the schema can include object classes that represent important elements such as design artifacts, features, parameters, constraints, versions,

and alternatives of a design. Finally, the schema can provide object classes that support mappings between classes that represent form, function, behavior and design artifacts.

### ***9.3 Impacts of the Research on Data Modeling and Data Exchange***

---

The following paragraphs discuss the impacts of this research on data modeling and data exchange.

***The working methodology that results from this research will aid in the future modeling of large complex facility engineering domains.*** The P-C Approach offers a methodology, including phases and modeling tools, for modeling facility data to support integration in a given domain. This approach was applied the transmission tower domain. The resulting primitive schema can provide a unified base to integrate data within this domain: Many composite classes representing complex user views can be customized from the same primitive schema. Those views correspond to a variety of activities across the tower life-cycle. Further, applications could be built using the underlying primitive schema, which provides a common language for the later data exchange between these applications. In the future, this approach will aid the future modeling of other facility engineering domains that are larger and more complex than transmission towers. The tower domain schema can be used as a convenient starting point for developing the schemata of other domains.

***A methodology such as the P-C Approach will help modelers accomplish their data modeling projects.*** From a project management point of view, conducting a modeling project for a facility engineering domain requires planning and coordination of different analysis and design activities, and management of the time and resources involved. Experience in this research project shows that a well-defined methodology is essential to conducting such a project. With the P-C Approach, modelers will have clear guidance about which modeling activities need to be carried out, what deliverables are expected by the end of each activity, and how those deliverables can be used to aid subsequent activities. This approach will also help in the overall planning and coordination of the project and in the estimation and allocation of the required time and resources.

***An integrated set of CASE tools automating the P-C Approach will assist modelers in modeling processes and data and will improve work productivity and design quality.*** Using the modeling tools of the P-C Approach, CASE computer tools can be built to automate the approach. An integrated set of CASE tools will assist modelers in designing, inspecting and verifying schemata that represent processes and data in the domain being modeled. Design decisions can also be recorded and documented for later review. With the assistance of CASE tools, modelers will have more time to focus on critical design issues and to explore more design alternatives. This is likely to expedite the modeling effort, improve the modelers' productivity, and enhance the quality of the final design. In addition, these CASE tools will make the communication between modelers and domain experts more effective. For instance, modelers will use CASE tools to quickly build prototypical representations of the domain's processes and data, and domain experts will instantly review these representations for correctness and accuracy. This communication is vital to the modelers' development of a domain primitive schema that will be used by the domain experts. CASE tools can also be design tools for the domain experts, depending on the

level of sophistication built into them. In that case, domain experts will assemble their own data representations with the aid of the CASE tools, and will negotiate among themselves regarding the sharability of those representations.

***The P-C Approach provides a methodology for building information systems for collaborative facility engineering work environments in the future.*** First, the CASE tools automating the P-C Approach can be linked to a database management system. These tools can be used to model processes and product data in a domain of interest. Databases of facility engineering processes and products can then be created and maintained. Computer-aided design applications in the environment can also be integrated with these databases. The automated modeling tools, process and product databases, and design applications constitute an information system in that domain. This system enables the owner, architects, engineers, and other participants to retrieve necessary information and make decisions in a given project. In this scenario, these users inspect all planning, design and construction activities of the process, review roles of various participants involved, and study the use and exchange of data throughout the life cycle. They query the physical data as well as the function and behavior data of the facility and its components, and ask why certain artifacts operate the way they do. They inspect numerical data, textual specifications, as well as graphical displays depicting facility building elements.

***The P-C Approach suggests a new paradigm for data exchange.*** Under this paradigm, application developers will no longer have to build special-purpose translators. Each application has its own composite classes that are assembled from the primitive classes of the underlying domain schema. Thus, each application essentially carries the descriptive knowledge needed to support the exchange of common data with any other application. To exchange data between two applications, the data from the first application is transferred into a primitive database that contains only instances of primitive classes and is then composed into the composite database of the second application. In any given domain, a model for data exchange will only be the set of primitive classes of that domain, rather than a complex product model that anticipates every possible combination of data in use. Applications need not share the same composite classes or the entire product model in order to exchange their common data. They share primitive classes and exchange only data that they really need. The exchanged data is the primitive instances needed to define composite instances in the application.

## ***9.4 Directions for Future Research***

---

To bring about the impacts described previously, the following paragraphs discuss the areas on which future research should focus.

***Enhancement of the P-C Approach through Application in other Engineering Domains*** In this research, the P-C Approach was developed and applied to a domain of transmission towers. A future extension of this work is the application of the approach to other facility engineering domains such as reinforced-concrete buildings, bridges, etc. The approach's applicability to domains other than structural engineering could also be examined. The experience gained from further validation will help enhance the approach in the following respects:

- Identification of additional requirements for the design of a domain primitive schema to support integration, and definitions of operational measures for testing the schema against these new requirements,
- Verification of the completeness and generality of the concepts included in PANDA (for functional analysis of facility engineering processes) and, if needed, inclusion of additional concepts,
- Study of entities used by experts in other facility engineering domains and, as a result, enhancement of the five cohesion dimensions included in DEAL,
- Verification and improvement of the rules and guidelines of the P-C modeling method for designing primitive classes,
- Identification and compilation of primitive class definitions that have high levels of reusability and thus, can be shared across many facility engineering domains,
- Accumulation of more experience in representing form, function, and behavior in facility engineering, leading to the recommendation of form, function, and behavior primitive classes to be included in a domain primitive schema.

### ***Management of Distributed Data in Collaborative Facility Engineering Work***

***Environments*** This research focused on data modeling rather than data management. Unless a way can be found to model facility data, management of this data is just an abstract question. As a future extension of this research, a study of facility data management will deal with issues such as data ownership, integrity, consistency, and concurrency. All of these issues are important and challenging for the management of distributed data, both primitive and composite, in the work environment of real-life facility engineering projects. This study will lead to recommendations of specific solutions for dealing with these issues. A current project on engineering data integration and concurrent design sponsored by the National Science Foundation [Ullman 91b] is studying some of these issues.

### ***Implementation of Case Tools Automating the Approach and Supporting***

***Data Exchange Using the Approach*** This research concentrated on the development, application, and testing of the P-C Approach, rather than on large software system implementation. In this research, a simple object-oriented database for a prototype tower using a commercial system (ONTOS) was built to demonstrate the research concept. In the future, CASE tools can be built for the following purposes:

- *Assisting the modeler in applying the P-C Approach to a given domain:* CASE tools can automate the modeling tools of the approach in an integrated fashion and can help expedite the modeling process using the approach. The previous section already explained the impacts of having these tools. These CASE tools can also be linked with commercial database management systems to implement a domain primitive schema and to create project databases.
- *Generating product entity definitions:* To support the current PDES/STEP area of product data exchange standards, CASE tools can help in the generation of entity definitions directly from the domain primitive schema using standard specification languages such as the PDES/STEP Express language.
- *Supporting data exchange using the P-C Approach:* Further, tools can be explored to support data exchange under the paradigm (via the domain primitive schema)

based on this approach, which was discussed in the previous section. To do so, additional work is needed to specify the data exchange mechanisms that need to be implemented.

***Further Development of Measures for the Validation of Data Standards or Schemata supporting Data Integration*** This research took the first step toward testing a schema designed for the purpose of data integration. It suggested three variables for measuring the schema performance (i.e., P-1 or schema completeness, P-2 or schema efficiency, and P-3 or schema sharing). Statistical methods can be used to improve the measurements of these variables. In addition, more variables are needed to evaluate other performance aspects of a schema: how accurately the schema represents the data used in the domain, how effectively the schema enables a user to navigate through the database (or how reachable a data item is from the way it is represented in the schema), how relevant the data presented to the user in that navigation is to the context in which the user is interested, etc.

## ***9.5 Final Remarks***

---

This dissertation reported on research in modeling facility data to support integration. The contribution of the research described here is a methodology, the P-C Approach, that can be used to analyze a given facility engineering domain and to design a common object-oriented schema for that domain. With this methodology, the research provided the groundwork for the modeling of large complex facility engineering domains and the development of CASE tools automating the modeling effort. These CASE tools can assist modelers in analyzing processes and data, designing database schemata, and building information systems to support collaborative facility engineering work. Further, this approach suggests a new data exchange paradigm that will be worth exploring. Looking into the future, this research leads to a wide open field of interesting and promising research ventures: enhancement of this approach through application in other engineering domains, study of distributed data management issues in collaborative facility engineering work environment, implementation of CASE tools automating the approach and supporting data exchange using the approach, and further development of measures for the validation of data standards or schemata supporting data integration.



# References

---

- [AISC 89] American Institute of Steel Construction (AISC), Inc., *Manual of Steel Construction — Allowable Stress Design*, Ninth edition, Chicago, IL, 1989.
- [ANSI 82] American National Standards Institute (ANSI), Inc., *Minimum Design Loads for Buildings and Other Structures*, ANSI, New York, NY, 1982.
- [ASCE 71] American Society of Civil Engineers (ASCE), Task Committee on Tower Design of the Committee on Analysis and Design of Structures of the Structural Division, *Guide for Design of Steel Transmission Towers*, Report No. 52, New York, NY, 1971.
- [Abdalla 89] Abdalla, G. A., *Object-Oriented Principles and Techniques for Computer Integrated Design*, Ph.D. Dissertation, Department of Civil Engineering, University of California at Berkeley, Berkeley, CA, 1989.
- [Abdalla 92] Abdalla, G. A., and Yoon, J. C., "Object-Oriented Finite Element and Graphics Data-Translation Facility," *Journal of Computing in Civil Engineering*, Vol. 6, No. 3, pp. 302-322, July, 1992.
- [Abrial 74] Abrial, J. R., "Data semantics," *Data Base Management*, edited by J. W. Klimbie and Koffeman, K. L., pp. 1-59, North-Holland, Amsterdam, 1974.
- [Abudayyeh 91] Abudayyeh, O., and Rasdorff, W. J., "The Design of Construction Industry Information Management Systems," *Journal of Construction Engineering and Management*, American Society of Civil Engineers (ASCE), Vol. 117, No. 4, December, 1991.
- [Ahmed 90] Ahmed, S., Sriram, D., and Logcher, R., *A Comparison of Object-Oriented Database Management Systems for Engineering Applications*, Technical Report No. IESL-90-03, Intelligent Engineering Systems Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139.
- [Albano 83] Albano, A., Cardelli, L., and Orsini, R., *Galileo: A Strongly Typed, Interactive Conceptual Language*, Report 83-11271-2, Bell Laboratories, Murray Hill, N. J., July 1983. (Also appears in [Borgida 85])
- [Andrews 88] Andrews, J., "Electronics, Too, Gets A Design Interchange Standard," *Computer Graphics Review*, pp. 34-44, November/December, 1988.
- [Andrews 90] Andrews, T., "The VBASE Object Database Environment," *Research Foundations in Object-Oriented and Semantic Database Systems*, pp. 221-240, edited by A. F. Cardenas and D. McLeod, Prentice Hall, 1990.
- [Astrahan 76] Astrahan, M. M., et al., "System R: Relational Approach to Database Management," *ACM Transactions on Database Systems*, pp. 97-137, 1976.
- [Atkinson 89] Atkinson, M., et al., "The Object-Oriented Database System Manifesto," *The First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, December, 1989.
- [Babbie 92] Babbie, E., *The Practice of Social Research*, 6th edition, Wadsworth Publication Co., Belmont, CA, 1992.
- [Barsalou 90] Barsalou, T., *View Objects for Relational Databases*, Ph. D. Dissertation, Department of Computer Science, Stanford University, Stanford, CA, 1990.



- [Batini 92] Batini, C., Ceri, S., and Navathe, S., *Conceptual Database Design — An Entity-Relationship Approach*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1992.
- [Batory 76] Batory, D. S., and Buchmann, A., "Molecular Objects, Abstract Data Types and Data Models: A Framework," *Proceedings of Very Large Data Bases*, pp. 9-36, Vol. 1, No. 1, March, 1976.
- [Baudin 89] Baudin, C., Sivard, C., and Zweben, M., "A model-based approach to design rationale conservation." *IJCAI-89 Workshop on Model-based Reasoning*, 1989.
- [Billmers 92] Billmers, M., and Adler, M., "Micromodeling As A Tool for Enterprise Integration," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Bisseret 88] Bisseret, A., Figeac-Letang, C. & Falzon, P., *Modeling opportunistic reasoning: the cognitive activity of traffic signal setting technicians*, Research Report No. 898, INRIA, Roquencourt, France.
- [Bjork 88] Bjork, BC., et al., "A Prototype Building Product Model Using a Relational Data Base," *Technical Research Centre of Finland (VTT)*, Laboratory of Urban Planning and Building Design, December, 1988.
- [Bjork 89a] Bjork, BC., "Product Models of Buildings and Their Relevance to Building Simulation," *Building Simulation '89 Conference*, International Building Performance Simulation Association, Vancouver, Canada, June, 1989.
- [Bjork 89b] Bjork, BC., "Issues in the Development of a Building Product Model Standard," *International Workshop on Computer Building Representation*, LESO-PB, Lausanne, Switzerland, October, 1989.
- [Booch 86] Booch, G., "Object-Oriented Development," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, February, 1986.
- [Booch 91] Booch, G., *Object-Oriented Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., 1991.
- [Borgida 85] Borgida, A., "Features of Languages for the Development of Information Systems at the Conceptual Level," *IEEE Software*, Vol. 2, No. 1, January, 1985.
- [Boy 91] Boy, G. A., *Is there Rationale for Storing and Retrieving*, Technical Report, NASA-Ames Research Center, 1991.
- [Bradshaw 92a] Bradshaw, J. M., and Boose, J. H., "Mediating Representations for Knowledge Acquisition," *AAAI Spring Symposium on Reasoning with Diagrammatic Representations*, Stanford University, March 24-27, 1992.
- [Bradshaw 92b] Bradshaw, J. M., et al., "eQuality: A Knowledge Acquisition Approach to Enterprise Integration," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Bravoco 85a] Bravoco, R. R., and Yadav, Surya, B., "Requirement Definition Architecture—An Overview," *Computer in Industry*, Vol. 6., pp. 237-251, 1985. (Also appears in [Chadha 91].)
- [Bravoco 85b] Bravoco, R. R., and Yadav, Surya, B., "A Methodology to Model the Information Structure of an Organization," *Computer in Industry*, Vol. 6., pp. 345-361, 1985. (Also appears in [Chadha 91].)
- [Brodie 82] Brodie, M. L., and Silva, E., "Active and Passive Component Modeling: ACM/PCM," In T.W. Olle, H. G. SOI and A. A. Verrijn-Stuart (editors), *Information Systems Design Methodologies: A Comparative Review*, Proceedings of the CRIS-1 Conference, North-Holland, pp. 93-142, 1982.

- [Brodie 84a] Brodie, M. L., "On the Development of Data Models," *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pp. 19-47, edited by M. L. Brodie, J. Mylopoulos and J. W. Schmidt, Springer-Verlag, NY, 1984.
- [Brodie 84b] Brodie, M. L., and Ridjanovic, D., *Fundamental Concepts for Semantic Modelling of Objects*, October Report, Computer Corporation of America, Cambridge, MA, 1984.
- [CODASYL 71] Data Base Task Group of the Programming Language Committee of the "Conference on Data Systems Languages" (CODASYL), *CODASYL Data Base Task Group Report*, Conference on Data Systems Languages, ACM, New York, April, 1971.
- [Carey 88] Carey, M. J., et al., *The EXODUS Extensible DBMS Project: an Overview*, University of Wisconsin, Madison, Computer Sciences, Technical Report No. 808, November 1988. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)
- [Cardenas 90] Cardenas, A. F., and McLeod, D., "An Overview of Object-Oriented and Semantic Database Systems," *Research Foundations in Object-Oriented and Semantic Database Systems*, pp. xvii-xxv, Prentice Hall, 1990.
- [Ceri 86] Ceri, S., "Requirements Collection and Analysis in Information Systems Design," *Proceedings of the IFIP Conference*, edited by H. J. Kugler, North-Holland, 1986.
- [Chadha 91] Chadha, B., et al., "An Appraisal of Modeling Tools and Methodologies for Integrated Manufacturing Information Systems," *Proceedings of the Fifth Symposium on Engineering Databases: An Engineering Resource*, 1991 ASME International Computers in Engineering Conference, American Society of Mechanical Engineers, Santa Clara, CA, August, 1991.
- [Chen 76] Chen, P. P. S., "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1, pp. 9-36, March, 1976.
- [Codd 70] Codd, E. F., "A Relational Model for Large Shared Databanks," *Communications of the ACM*, Vol. 13, No. 6, pp. 377-390, June, 1970.
- [Codd 71a] Codd, E. F., "Normalized Data Base Structure: A Brief Tutorial," *Proceedings of ACM-SIGFIDET Workshop on Data Description, Access and Control*, San Diego, CA, pp. 35-68, November, 1971.
- [Codd 71b] Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus," *Proceedings of ACM-SIGFIDET Workshop on Data Description, Access and Control*, San Diego, CA, pp. 35-68, November, 1971.
- [Codd 72a] Codd, E. F., "Further Normalization of the Data Base Relational Model," *Data Base Systems, Courant Computer Science Symposium 6th*, edited by R. Rustin, pp. 33-64, Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [Codd 72b] Codd, E. F., "Relational completeness of data base sublanguages," *Data Base Systems, Courant Computer Science Symposium 6th*, edited by R. Rustin, pp. 65-98, Prentice-Hall, Englewood Cliffs, NJ, 1972.
- [Codd 79] Codd, E. F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems*, Vol 4, No. 4, pp. 397-434, December, 1979.
- [Conklin 91] Conklin, J. E., and Yakemovic, B., *A Process-Oriented Approach to Design Rationale*, Unpublished Report, Corporate Memory Systems, Inc., Austin, TX, 1991.

- [Cox 86] Cox, B. J., *Object-Oriented Programming, An Evolutionary Approach*, Addison-Wesley Publishing Co., Reading, MA, 1986.
- [Dahl 70] Dahl, O.-J., Myhrhaug, B., and Nygaard, K., *The SIMULA 67 Common Base Language*, Publication S22, Norwegian Computing Centre, Oslo, 1970.
- [Date 90] Date, C. J., *An Introduction to Database Systems - Volume I*, 5th ed., Addison-Wesley, 1990.
- [Davis 85] Davis, R., "Diagnostic Reasoning based on Structure and Function," *Qualitative Reasoning about Physical Systems*, edited by Bobrow, D., MIT Press, Cambridge, MA, 1985.
- [Dayal 87] Dayal, U., et. al., "Simplifying Complex Objects: The PROBE Approach to Modelling and Querying Them," *Proceedings German Database Conference*, Burg Technik und Wissenschafts, Darmstadt, April 1987. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)
- [de Kleer 84] de Kleer, J., and Brown, J. S. Brown, "Mental Models of Physical Mechanisms and their Acquisition," *Cognitive Skills and their Acquisition*, edited by Anderson, J. R., pp. 285-309, 1980.
- [De Marco 82] De Marco, T., *Structured Analysis and System Specification*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.
- [Dixon 86] Dixon, J., "Artificial Intelligence and Design: A Mechanical Engineering View," *Proceedings of AAAI-86*, pp. 872-877, 1986.
- [Dixon 88] Dixon, J., et al., "A Proposed Taxonomy of Mechanical Design Problems," *Proceedings of Computers in Engineering 1988*, 1988, pp. 41-46.
- [EPRI 87] Electric Power Research Institute (EPRI), *Guidelines for Specifying Integrated Computer-Aided Engineering Applications for Electric Power Plants*, Final Report No. EPRI NP-5259M, Project 2514-3, May, 1987.
- [Eastman 78] Eastman, C. M., "The Representation of Design Problems and Maintenance of their Structure," *Artificial Intelligence and Pattern Recognition in Computer-Aided Design*, North Holland, 1978.
- [Eastman 91] Eastman, C. M., Bond, A. H., and Chase, S. C., "A data model for design databases," *Artificial intelligence in design '91*, edited by J. S. Gero, Butterworth & Heinemann, pp. 339-365, 1991.
- [Eastman 92] Eastman, C. M., "Modeling of buildings: evolution and concepts," *Automation in Construction*, pp. 99-109, 1992.
- [Eisenhardt 89] Eisenhardt, K. M., "Building Theories from Case Study Research," *Academy of Management Review*, Vol. 14, No. 4, pp. 532-550, 1989.
- [Ellis 79] Ellis, C. A., "Information Control Nets: A Mathematical Model of Office Information Flow," *Proceedings of ACM Conference on Simulation Modeling and Measurement of Computer Systems*, 1979.
- [Fenves 88] Fenves, S. J., et al., "An Integrated Software Environment for Building Design and Construction," *Proceedings of the Fifth ASCE Conference on Computing in Civil Engineering: Microcomputers to Supercomputers*, pp. 21-32, Alexandria, VA, March, 1988.
- [Fischer 89] Fischer, G., and McCall, R., "JANUS: Integrating hypertext with a knowledge-based design environment," *Hypertext*, 1989.
- [Fishman 87] Fishman, D. H., et al., "Iris: An Object-Oriented Database Management System," *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, pp. 48-69, January, 1987.

- [Flores 70] Flores, I., *Data Structure and Management*, Prentice-Hall, Englewood Cliffs, NJ, 1970.
- [Fox 92] Fox, M., "The TOVE Project: Toward A Common-Sense Model of the Enterprise," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Franke 91] Franke, D. W., "Deriving and Using Descriptions of Purpose," *IEEE Expert*, Vol. 2, No. 2, pp. 41-47, April, 1991.
- [Fritchman 90] Fritchman, B. L., et al., "SIM: Design and Implementation of A Semantic Database System," *Research Foundations in Object-Oriented and Semantic Database Systems*, pp. 242-265, edited by A. F. Cardenas and D. McLeod, Prentice Hall, 1990.
- [Froese 92] Froese, T. M., *Integrated Computer-Aided Project Management Through Standard Object-Oriented Models*, Ph. D. Dissertation, Department of Civil Engineering, Stanford University, Stanford, CA, June, 1992.
- [Gane 79] Gane, C., and Sarson, T., *Structured System Analysis*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1979.
- [Garcia 92] Garcia, A.C., *Active Design Documents: A New Approach for Supporting Documentation in Preliminary Routine Design*, Ph.D. Dissertation, Department of Civil Engineering, Stanford University, Stanford, CA, August, 1992.
- [Garrett 92] Garrett, J. H., and Hakim, M. M., "Object-Oriented Model of Engineering Design Standards," *Journal of Computing in Civil Engineering*, Vol. 6, No. 3, pp. 323-347, July, 1992.
- [Gerardi 88] Gerardi, M., et al., NIDDESC: The Navy/Industry Digital Data Exchange Standards Committee, *Reference Model For Ship Structural Systems*, Version 3.0, ISO TC184/SC4/WG1 Document 3.2.2.5, October, 1988.
- [Gielingh 88] Gielingh, W., "General AEC Reference Model (GARM): an aid for the integration of application specific product definition models," *Conceptual Modelling of Buildings*, *CIB Proceedings*, Publication 126, edited by P. Christiansson and H. Karlsson, The Swedish Building Centre, October, 1988.
- [Goel 89a] Goel, V., and Pirolli, P., "Motivation of the Notion of Generic Design within Informatin-Processing Theory: The Design Problem Space," *AI Magazine*, Spring, 1989.
- [Goel 89b] Goel, A., and Chandrasekaran, B., "Functional Representation of Designs and Redesign Problem Solving," *Proceedings of the 11th International Joint Conference on AI*, Detroit, August, 1989.
- [Goldberg 85] Goldberg, A., and Robson, D., *Smalltalk-80, the Language and its Implementation*, Addison-Wesley Publishing Co., Reading, MA, 1985.
- [Gotthard 92] Gotthard, W., Lockemann, P. C., and Neufeld, A., "System-Guided View Integration for Object-Oriented Databases," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 4, No. 1, February, 1992.
- [Greenspan 86] Greenspan, S. J., Borgida, A., Mylopoulos, J., "A Requirements Modeling Language and its Logic," *Information Systems*, Vol. 11, No. 1, pp. 9-23, 1986.
- [Grosos 92] Grosos, B., and Morgenstern, L., "Applications of Logicist Knowledge Representation to Enterprise Modelling," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Gruber 90a] Gruber, T. R., *Model-based Explanation of Design Rationale*, Technical Report KSL 90-33, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, CA, March, 1990.

- [Gruber 90b] Gruber, T. R., *Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use*, Technical Report KSL 90-45, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, CA, July, 1990.
- [Gruber 91] Gruber, T. R., et al., "Design Rationale Capture as Knowledge Acquisition: Tradeoffs in the Design of Interactive Tools." *Machine Learning: Proceedings of the Eight International Workshop*, Morgan Kaufmann, San Mateo, CA, pp. 3-12, 1991.
- [Gruber 92] Gruber, T., Tenenbaum, J., and Weber, J., "Toward a Knowledge Medium for Collaborative Product Development," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Gustavsson 82] Gustavsson, M. R., Karlsson, T., Bubenko, J. A., "A Declarative Approach to Conceptual Information Modeling," in T.W. Olle, H. G. SOI and A. A. Verrijn-Stuart (editors), *Information Systems Design Methodologies: A Comparative Review*, Proceedings of the CRIS-1 Conference, North-Holland, pp. 93-142, 1982.
- [Hammer 81] Hammer, M., and McLeod, D., "Database Description with SDM: A Semantic Database Model," *ACM Transactions on Database System*, Vol. 6, No. 3, 1981. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)
- [Hardwick 89] Hardwick, M., and Spooner, D. L., "The ROSE Data Manager: Using Object Technology to Support Interactive Engineering Applications," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 2, June, 1989.
- [Howard 86] Howard, H. C., and Rehak, D. R., *Interfacing Databases and Knowledge Based Systems for Structural Engineering Applications*, Technical Report EDRC-12-06-86, Engineering Design Research Center, Carnegie-Mellon University, Pittsburgh, PA, November, 1986.
- [Howard 88] Howard, H. C., *Reasoning about Multiple Views of AEC Data*, CIFE Working Paper, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, October 1988.
- [Howard 89a] Howard, H. C., et al., "Computer-Integrated Design and Construction: Reducing Fragmentation in the AEC Industry," *Journal of Computing in Civil Engineering*, ASCE, January, 1989.
- [Howard 89b] Howard, H. C., and Levitt, R. E., "Linking Design Data with Knowledge-Based Construction Systems," *A Flaship Project Proposal to CIFE*, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, October 1989.
- [Howard 89c] Howard, H. C., and Rehak, D. R., "KADBASE: A Prototype Expert System-Database Interface for Engineering Systems," *IEEE Expert*, 1989.
- [Howard 91a] Howard, H. C., "Project-Specific Knowledge Bases in AEC Industry," *Journal of Computing in Civil Engineering*, Vol. 5, No. 1, pp. 25-41, January, 1991.
- [Howard 91b] Howard, H. C., and Abdalla, G. A., *Object-Oriented Database Workshop Tutorial*, Symposium on Databases, Seventh ASCE Conference on Computing in Civil Engineering, Washington, D. C., May 6-8, 1991.
- [Howard 92] Howard, H. C., Abdalla, G. A., and Phan, D. H., "A Primitive-Composite Approach for Structural Data Modelling," *Journal of Computing in Civil Engineering*, Vol. 6, No. 1, January, 1992.
- [Hsieh 82] Hsieh, Y. Y., *Elementary Theory of Structures*, 2nd edition, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1982.

- [Hull 90] Hull, R., and King, R., "A Tutorial on Semantic Database Modeling," *Research Foundations in Object-Oriented and Semantic Database Systems*, pp. xvii-xxv, Prentice Hall, 1990.
- [Hurson 93] Hurson, A. R., Pakzad, S. H., and Cheng, J., "Object-Oriented Database Management Systems: Evolution and Performance Issues," *IEEE Computer*, pp. 48-60, 1993.
- [IGES 90] *IGES: Initial Graphics Exchange Specification*, Version 5.0, U.S. Department of Commerce, National Bureau of Standards, National Engineering Laboratory, Center for Manufacturing Engineering, Automated Production Technology Division, Washington, D. C., 1990.
- [Itasca 90] Itasca Systems, Inc., *ITASCA Distributed Object-Oriented Database Management System — Technical Summary*, MN, 1990. (Also appears in [Ahmed 90].)
- [Jain 90] Jain, D., Luth, G. P., Krawinkler, H., and Law, K., *A Formal Approach to Automating Conceptual Structural Design*, CIFE Technical Report No. 31, Center for Integrated Facility Engineering, Stanford University, Stanford, CA, August, 1990.
- [Jagannathan 92] Jagannathan, V., et al., "Information Sharing System," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Johnson 83] Johnson, H. R., Schweitzer, J. E., and Warkentine, E. R., "A DBMS Facility For Handling Structured Engineering Entities," *ACM SIGMOD/IEEE, Engineering Design Applications*, 1983.
- [Kersten 86] Kersten, M., and Schippers, F. H., "Towards an Object-Centered Database Language," *IEEE*, pp. 104-112, 1986.
- [Katabchi 88] Katabchi, M., and Berzins, V., "Mathematical Model of Composite Objects and Its Application for Organizing Engineering Databases," *IEEE*, Vol. 14, No. 1, pp. 71-83, January, 1988.
- [Keuneke 91] Keuneke, A. M., "Device Representation — The Significance of Functional Knowledge," *IEEE Expert*, April, pp. 22-25, 1991.
- [Khoshafian 86] Khoshafian, S. N., and Copeland, G. P., "Object Identity," *ACM Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, Portland, OR, September, 1986. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)
- [Kim 90] Kim, W., *Introduction to Object-Oriented Databases*, The MIT Press, Cambridge, MA, 1990.
- [Kim 93] Kim, W., "Object-Oriented Databases: Definition and Research Directions," *IEEE*, pp. 327-341, 1993.
- [King 84] King, R., and McLeod, D., "A Unified Model and Methodology for Conceptual Database Design," *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pp. 313-331, edited by M. L. Brodie, J. Mylopoulos and J. W. Schmidt, Springer-Verlag, NY, 1984.
- [King 86] King, R., "A Database Management System Based on an Object-Oriented Model," *Expert Database Systems*, Benjamin/Cummings, Menlo Park, CA, pp. 443-468, 1986.
- [Kuffner 91] Kuffner, T. A., and Ullman, D. G., "The information requests of mechanical design engineers," *Design Studies*, pp. 42-50, 1991.

- [LPM 90] LPM (LU, CTICM, IDDC) Working Group, *Logical Product Model for Structural Steelwork*, Eureka Project EU130: CIMSTEEL, Version 2.1, Working draft, January, 1990.
- [La Rota 90] La Rota, J. L., Biswas, G., and Basu, P. K., "A Model-Based Approach to Structural Design," *Applications of Artificial Intelligence in Engineering V, Proceedings of the Fifth International Conference*, Volume I, edited by J.S. Gero, Boston, MA, pp. 3-22, 1990.
- [Lavakare 89] Lavakare, A., and Howard, H. C., *Structural Steel Framing Data Model*, Technical Report No. 012, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, June, 1989.
- [Lave 75] Lave, C. A., and March, J. G., *An Introduction to Models in the Social Sciences*, Harper and Row, NY, 1975.
- [Law 86] Law, K. H., and Jouaneh, M. K., "Data Modelling for Building Design," *Computing in Civil Engineering, Proceedings of the Fourth Conference*, October, 1986.
- [Law 89] Law, K. H., Barsalow, T., and Wiederhold, G., *Management of Complex Structural Engineering Objects in a Relational Framework*, Technical Report No. 19, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, September, 1989.
- [Law 92] Law, K. H., Barsalow, T., and Wiederhold, G., *Managing Design Information in a Shareable Relational Framework*, Technical Report No. 60, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, September, 1992.
- [Litwin 90] Litwin, W., and Mark, L., "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, pp. 267-293, 1990.
- [Loomis 87] Loomis, M. E., Shah, A. V., and Rumbaugh, J. E., "An Object Modeling Technique for Conceptual Design," *European Conference on O-O Programming*, pp. 192-202, 1987.
- [Lorie 83] Lorie, R., and Plouff, W., "Complex Objects and Their Use in Design Transactions," *Proceedings Engineering Design Applications Stream of ACM-IEEE Data Base Week*, San Jose, CA, May, 1983.
- [Luiten 91a] Luiten, B., et al., "Development and Implementation of Multi-Layered Project Models," *Second International Workshop on Computer Building Representation for Integration*, Aix-les-Bains, France, June 1991.
- [Luiten 91b] Luiten, B., and Tolman, F., "Project Information Integration for the Building and Construction Industries," *Proceedings of the 4th IFIP Conference on Computer Applications in Production and Engineering*, September 1991, Bordeaux, France, ISBN 0-444-891-5, North Holland, The Netherlands, 1991.
- [Lundeberg 82] Lundeberg, M., "The ISAC Approach to Specification of Information Systems and its Application to the Organization of an IFIP Working Conference," In T.W. Olle, H. G. Sol and A. A. Verrijn-Stuart (editors), *Information Systems Design Methodologies: A Comparative Review*, Proceedings of the CRIS-1 Conference, North-Holland, 1982.
- [Luth 91] Luth, G. P., *Representation and Reasoning for Integrated Structural Design*, Ph.D. Dissertation, Department of Civil Engineering, Stanford University, Stanford, CA, June, 1991.
- [McEliece 89] McEliece, R. J., Ash, R. B., and Ash, C., *Introduction to Discrete Mathematics*, Random House, NY, 1989.
- [McGee 77] McGee, W. C., "the IMS/Vs System," *IBM Sys. J.* 16, No. 2, June, 1977. (Also appears in [Date 90].)

- [Maher 85] Maher, M. L., and Fenves, S. J., *HI-RISE: A knowledge-based expert system for the preliminary structural design of high rise buildings*, Technical Report R-85-146, Department of Civil Engineering, Carnegie-Mellon University, 1985.
- [Maier 86] Maier, D., Stein, J., Otis, A., and Purdy, A., "Development and Implementation of an Object-Oriented DBMS," *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, pp. 472-482, September, 1986. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)
- [Mayer 92] Mayer, R. J., et al., *Information Integration For Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*, Technical Report AL-TR-1992-0057, Armstrong Laboratory, May, 1992.
- [Meng 90] Meng, B., "Object-Oriented Programming," *MacWorld Magazine*, January, 1990.
- [Meriam 86] Meriam, J. L., and Kraige, L. G., *Engineering Mechanics, Volume 1: Statics*, 2nd edition, John Wiley & Sons, New York, 1986.
- [Meunier 88] Meunier, K. L., Dixon, J. R., "Interactive Respecification: a Computational Model for Hierarchical Mechanical System Design," *Proceedings of Computers in Engineering*, pp. 25-32, 1988.
- [Meyer 88] Meyer, B., "Object-Oriented Software Construction," *Interactive Software Engineering*, Santa Barbara, 1988.
- [Minsky 75] Minsky, M. L., "A Framework for Representing Knowledge," *The Psychology of Computer Vision*, pp. 211-277, edited by P. Winston, McGraw-Hill, NY, 1975.
- [Mittal 86] Mittal, S., Dym, C., and Morjaria, M., "PRIDE: An Expert System for the Design of Paper Handling Systems," *Computer*, July, 1986.
- [Mortensen 85] Mortensen, M., *Geometric Modeling*, John Wiley & Sons, NY, 1985.
- [Mostow 85] Mostow, J., "Toward Better Models of the Design Process," *The AI Magazine*, pp. 44-57, Spring, 1985.
- [Mylopoulos 80] Mylopoulos, J., Bernstein, P. A., and Wong, H. K. T., "A Language Facility for Designing Database-Intensive Applications," *ACM Transactions on Database Systems*, Vol. 5, No.2, pp. 185-207, June, 1980.
- [Navathe 92] Navathe, S. B., "Evolution of Data Modeling for Databases," *Communications of the ACM*, Vol. 35, No. 9, September, 1992.
- [Newell 72] Newell, A., and Simon, H. A., *Human Problem Solving*. Englewood Cliffs, Prentice Hall, NJ, 1972.
- [Object Design 91] Object Design, Inc., *ObjectStore — Technical Overview*, Release 1.1, One New England Executive Park, Burlington, MA 01803, 1991.
- [Objectivity 91] Objectivity, Inc., *Objectivity/DB Product Data Sheet*, 800 El Camino Real, Menlo Park, CA 94025, 1991.
- [Ozsu 91] Ozsu, M. T., and Valduriez, P., *Principles of Distributed Database Systems*, Prentice Hall, Inc., 1991.
- [Parsaye 89] Parsaye, K., et al., *Intelligent Databases - Object-Oriented, Deductive, Hypermedia Technologies*, John Wiley & Sons, Inc., 1989.
- [Peterson 77] Peterson, J. L., "Petri Nets," *Computing Surveys*, Vol. 9, No. 3, pp. 223-252, September, 1977.
- [Phan 90] Phan, D. H., and Howard, H. C., *Evaluation of the Structural Steel Framing Data Model*, Technical Report No. 41, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, November, 1990.



- [Phan 91a] Phan, D. H., *Conceptual Development and Database Applications of the Primitive-Composite Data Model for Structural Engineering*, Unpublished Ph. D. Research Proposal, Department of Civil Engineering, Stanford University, Stanford, CA, July, 1991.
- [Phan 91b] Phan, D. H., Abdalla, J. A., and Howard, H. C., *CIFE Data Inventory: A Report on CIFE Data-Intensive Research Project*, Technical Report No. 57, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, October, 1991.
- [Phan 92] Phan, D. H., *Functional Analysis for Facility Engineering Data Modeling using the Partitioned eNginering DATA flow model (PANDA)*, Technical Report No 77. Center For Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, 1992.
- [Phan 93] Phan, D. H., and Howard, H. C., "Applying Constraint Satisfaction Neural Networks to Multiple Views' Data Partitioning in Building Engineering," Accepted to *the Fifth International Conference on Computing in Civil and Building Engineering*, American Society of Civil Engineers, Anaheim, CA, June 7-9, 1993.
- [Powell 88] Powell, G., et al., "A Database Concept for Computer Integrated Structural Engineering Design," *ASCE Fifth Conference on Computers*, Alexandria, March, 1988.
- [Rafiq 90] Rafiq, T., *Project-Specific Knowledge for Facility Engineering*, Unpublished Ph.D. Research Proposal, Department of Civil Engineering, Stanford University, Stanford, CA, April 1990.
- [Rasdorf 90] Rasdorf, W. J., Lakmazaheri, S., and Abudayyeh, O., "The Development of a Geometric Modeling/Database Management Interface," *International Journal of Advances in Engineering Software*, Computational Mechanics Institute, Vol. 2, No. 2, pp. 84-98, April, 1990.
- [Reed 88] Reed, K. A., "Product Modelling of Buildings for Data Exchange Standards: from IGES to PDES/STEP and Beyond," *Conceptual Modelling of Buildings, CIB Proceedings*, Publication 126, edited by P. Christiansson and H. Karlsson, The Swedish Building Centre, October, 1988.
- [Reiter 84] Reiter, Raymond, "Towards a Logical Reconstruction of Relational Database Theory," *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages*, pp. 191-238, edited by M. L. Brodie, J. Mylopoulos and J. W. Schmidt, Springer-Verlag, NY, 1984.
- [Rettig 89] Rettig, M., Morgan, T., Jascobs, J., and Winberly, D., "Object-Oriented Programming in AI, New Choices," *AI Expert Magazine*, January 1989.
- [Richter 82] Richter, G., and Durchholz, R., "IML-Inscribed High-Level Petri Nets," In T.W. Olle, H. G. SOI and A. A. Verrijn-Stuart (editors), *Information Systems Design Methodologies: A Comparative Review*, Proceedings of the CRIS-1 Conference, North-Holland, 1982.
- [Ross 77a] Ross, D., and Shoman, K., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, pp. 6-15, January, 1977.
- [Ross 77b] Ross, D., "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, pp. 16-34, January, 1977.
- [Rowe 87] Rowe, L., and Stonebraker, M. "The Postgres Data Model," *Proceedings of the XIII International Conference on Very Large Databases*, Brighton, England, September 1987. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)

- [Rumbaugh 91] Rumbaugh, J., et al., *Object-Oriented Modeling and Design*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991.
- [SEA 88] Structural Engineers Association of California, *Recommended Lateral Force Requirements and Tentative Commentary*, Seismology Committee, San Francisco, CA, 1988.
- [SCPUC 69] State of California Public Utilities Commission, *Rules for Overhead Electrical Line Construction — General Order No. 95*, Documents Section of the State of California, Sacramento, CA, 1969.
- [Sanvido 84] Sanvido, V. E., *Designing Productivity Management and Control Systems for Construction Projects*, Ph.D. Dissertation, Department of Civil Engineering, Stanford University, Stanford, CA, June 1984.
- [Sanvido 90] Sanvido, V. E., *An Integrated Building Process Model*, Technical Report No. 1, Computer Integrated Construction Research Program, Department of Architectural Engineering, Pennsylvania State University, University Park, PA, January 1990.
- [Sause 89] Sause, R., *A Model of the Design Process for Computer Integrated Structural Engineering*, Ph. D. Dissertation, University of California, Berkeley, CA, 1989.
- [Sause 92] Sause, R., Martini, K., and Powell, G. H., "Object-Oriented Approaches for Integrated Engineering Design Systems," *Journal of Computing in Civil Engineering*, Vol. 6, No. 3, pp. 248-265, July, 1992.
- [Sembugamoorthy 86] Sembugamoorthy, V., and Chandrasekaran, B., "Functional Representation of Devices and Compilation of Diagnostic Problem-Solving Systems," *Experience, Memory, and Reasoning*, edited by J. L. Kolodner and C. K. Riesbeck, pp. 47-73, 1986.
- [Shema 90] Shema, D. B., et al., "Design knowledge capture and alternatives generation using possibility tables in Canard," *Knowledge Acquisition*, pp. 345-363, 1990.
- [Sheth 90] Sheth, A., and Larson, J. A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No.3, pp. 183-236, September 1990.
- [Shipman 81] Shipman, D. W., "The Functional Data Model and the Data Languages DAPLEX," *ACM Transactions on Database Systems*, Vol. 6, No. 1, pp. 140-173, March, 1981. (Also appears in *Readings in Object-Oriented Database Systems*, edited by S. Zdonik and D. Maier, Morgan Kaufman, 1990.)
- [Smith 77] Smith, J. M., and Smith, D.C.P., "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, Vol. 2, No. 2, pp. 105-133, June, 1977.
- [Spooner 86] Spooner, D. L., "An Object-Oriented Data Management System for Mechanical CAD," *Proceedings of the 1986 International Workshop on Object-Oriented Database Systems*, September, 1986.
- [Srinivasan 92] Srinivasan, K., and Jayaraman, S., "Design and Development of an Enterprise Modeling Methodology," *Workshop Notes of the 1192 Workshop Program on AI in Enterprise Integration*, sponsored by the American Association for Artificial Intelligence, San Jose, July, 1992.
- [Sriram 84] Sriram, D., *Knowledge-Based Approaches for Structural Design*, Unpublished Manuscript, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1984.
- [Sriram 89] Sriram, D., Logcher, R. D., Groleau, N., and Cherneff, J., *DICE: An Object-Oriented Programming Environment for Cooperative Engineering Design*, Technical Report No. IESL-89-03, Intelligent Engineering Systems Laboratory,

Civil Engineering Department, Massachusetts Institute of Technology, Cambridge, MA.

- [Staley 86] Staley, S. M., and Anderson, D. C., "Functional Specification for CAD Databases," *Computer-Aided Design*, Vol. 18, No. 3, pp. 132-138, April, 1986.
- [Stefik 90] Stefik, M., *Introduction to Knowledge Systems*, Unpublished Book Draft, Xerox Palo Alto Research Center, Summer 1990 Version, 1990.
- [Steinberg 87] Steinberg, L. "Design as refinement plus constraint propagation: The VEXED experience." *AAAI*, pp. 830-835, August, 1987.
- [Stonebraker 76] Stonebraker, M., Wong, E., and Kreps, P. "The Design and Implementation of INGRES," *ACM Transactions on Database Systems*, pp. 189-222, 1976.
- [Stonebraker 86a] Stonebraker, M., "Object Management in Postgres Using Procedures," *IEEE*, p. 66, 1986.
- [Stonebraker 86b] Stonebraker, M., and Rowe, L. A., "The Design of POSTGRES," *Proceedings of the International Conference on the Management of Data*, pp. 340-355, June, 1986.
- [Stroustrup 88] Stroustrup, B., "What is Object-Oriented Programming?," *IEEE Software*, May 1988.
- [Sundgren 74] Sundgren, B., "The Infological Approach to Data Bases," *Data Base Management*, North-Holland, 1974.
- [Tatum 90] Tatum, C. B., *Managing Integration Technology for Engineering and Construction*, Unpublished Working Paper, Center for Integrated Facility Engineering (CIFE), Stanford University, Stanford, CA, March, 1990.
- [Thomas 89] Thomas, D., "What's in an Object?," *Byte Magazine*, pp. 231-253, March 1989.
- [Thomas 90] Thomas, G., et al., "Heterogeneous Distributed Database Systems for Production Use," *ACM Computing Surveys*, Vol. 22, No. 3, pp. 237-266, September, 1990.
- [Tiwari 93] Tiwari, S., and Howard, H. C., "The Management of Design: A Design Notification Scheme for Distributed AEC Framework," *The First International Conference on the Management of Information Technology for Construction*, Singapore, August 17-20, 1993.
- [Tong 90] Tong, C., "Knowledge-Based Design as Engineering Science: the Rutgers AI/Design Project," *Applications of Artificial Intelligence in Engineering V, Proceedings of the Fifth International Conference*, Volume I, edited by J.S. Gero, Boston, MA, pp. 3-22, 1990.
- [Tsichritzis 76] Tsichritzis, D. C., and Lochovsky, F. H., "Hierarchical Data Base Management: A Survey," *ACM Comp. Surv.* 8, No. 1, March, 1976. (Also appears in [Date 90].)
- [Tsichritzis 82] Tsichritzis, D. C., and Lochovsky, F. H., *Data Models*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
- [Turner 8a] Turner, J., Architecture/Engineering/Construction (AEC) Building Systems Model, *PDES/STEP*, ISO TC184/SC4/WG1, Document 3.2.2.4, July, 1988.
- [Ullman 91a] Ullman, D. G., "Design Histories: Archiving the Evolution of Products," *Proceedings of the DARPA Workshop on Manufacturing*, Salt Lake City, Utah, February 5-6, 1991.
- [Ullman 91b] Ullman, J. D., et al., "Integrated Data Exchange and Concurrent Design for engineered facilities," Proposal to the National Science Foundation, October, 1991.

- [Umeda 90] Umeda, Y., Takeda, H., Tomiyama, T., and Yoshikawa, H., "Function, Behavior, and Structure." *Applications of Artificial Intelligence in Engineering V, Proceedings of the Fifth International Conference*, Volume I, edited by J.S. Gero, Boston, MA, p.p. 177-194, 1990.
- [Vanegas 87] Vanegas, J. A. P., *A Model for Design/Construction Integration During the Initial Phases of Design for Building Construction Projects*, Ph.D. Dissertation, Department of Civil Engineering, Stanford University, Stanford, CA, 1987.
- [VanLehn 89] VanLehn, K., "Problem Solving and Cognitive Skill Acquisition," *Foundations of Cognitive Science*, edited by M. Posner, pp. 527-579, The MIT Press, Cambridge, MA, 1989.
- [Verheijen 82] Verheijen, G. M.A., and Bekkum, J. v., "NIAM: An Information Analysis Method," In T.W. Olle, H. G. SOI and A. A. Verrijn-Stuart (editors), "Information Systems Design Methodologies: A Comparative Review," *Proceedings of the CRIS-I Conference*, North-Holland, 1982.
- [Visser 89] Visser, W., "More or Less Following a Plan during Design: Opportunistic deviations in Specification," *Special Issue on Empirical Studies of Programmers of the International Journal of Man-Machine Studies*, 1989.
- [Yao 78] Yao, S. B., Navathe, S. B., and Weldon, J., "An Integrated Approach to Logical Database Design," *Proceedings of New York University Symposium on Database Design*, May, 1978.
- [Yourdon 79] Yourdon, E., and Constantine, L. L., *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1979.
- [Zdonik 90] Zdonik, S. B., and Maier, D., "Fundamental of Object-Oriented Databases," *Readings in Object-Oriented DataBase Systems*, The Morgan Kaufmann Series in Data Management Systems, (series) edited by Jim Gray, Morgan Kaufmann Publishers, Inc., 1990.
- [Zweben 89] Zweben, M., and Eskey, M. "Constraint Satisfaction with Delayed Evaluation," *Proceedings of IJCAI-89*, pp. 875-880, 1989.
- [Wang 83] Wang, C. K., *Intermediate Structural Analysis*, McGraw-Hill, Inc., New York, 1983.
- [Warthen 88] Warthen, B., "PDES - A CAD Standard For Data Exchange," *Unix World*, December, 1988.
- [Warthen 90] Warthen, B., " PDES Shapes Data Exchange Technology," *Computer-Aided Engineering*, February, 1990.
- [Webster 86] Merriam-Webster Inc., Publishers, *Webster's Third New International Dictionary of the English Language Unabridged*, Editor in Chief P.B. Grove and the Merriam-Webster Editorial Staff, Springfield, MA, 1986.
- [Wiederhold 80] Wiederhold, G., "The Structural Model for Data Base Design," *Proceedings International Conference on the Entity-Relationship Approach to System Analysis and Design*, 1980.
- [Wiederhold 83] Wiederhold, G., *Database Design*, Computer Science Series, McGraw Hill, New York, NY, 2nd edition, 1983.
- [Wiederhold 84] Wiederhold, G., "Databases," *IEEE Computer*, Centennial issue, Vol. 17, No. 10, pp. 211-223, October, 1984.
- [Wiederhold 85] Wiederhold, G., et al., "Models for Engineering Information Systems," *Proceedings of the 1985 VHSIC Conference*, December, 1985.

- [Wiederhold 86] Wiederhold, G., "Knowledge versus Data," *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*, pp. 77-82, edited by M. L. Brodie, and J. Mylopoulos, Springer-Verlag, NY, 1986.
- [Wiederhold 86b] Wiederhold, G., "Views, Objects, and Databases," *IEEE Computer*, pp. 37-44, December, 1986.
- [Wiederhold 88] Wiederhold, G., "Engineering Information Systems: Prospects and Problems of Integration," *IEEE Spring COMPCON Digest of Papers*, pp. 228-229, March, 1988.
- [Wiederhold 89] Wiederhold, G., "The architecture of future information systems," *Proceedings of the International Symposium on Database Systems for Advanced Applications*, KISS and IPSJ, Seoul, Korea, 1989.
- [Wiederhold 91] Wiederhold, G., "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, March, 1991.
- [Williams 89] Williams, A. L., *Comparative Analysis of the Modeling Languages: IDEF1X, Express, and NIAM*, Information Technology, McDonnell Aircraft Company, May, 1989.
- [Wilson 88] Wilson, P.R., and Kennicott, P.R., *PDES/STEP Integrated Product Information Model (IPIM)*, ISO TC 184/SC4/WG1, Doc. 4.1.2., Working Draft, September, 1988.
- [Wimmer 92] Wimmer, K., and Wimmer, N., "Conceptual Modelling Based on Ontological Principles," Submitted to *Knowledge Acquisition*, Revised Version, May, 1992.
- [Wulf 76] Wulf, W. A., London, R. L., and Shaw, M., "An Introduction to the construction and verification of alphard programs," *IEEE Transactions on Software Engineering*, 1976.

**The Primitive-Composite (P-C) Approach-  
A Methodology for Developing Sharable  
Object-Oriented Data Representations  
For Facility Engineering Integration:**

**Appendices**

by

Dr. D. H. Douglas Phan

Dr. H. Craig Howard

**TECHNICAL REPORT  
Number 85B**

August, 1993

**Stanford University**

© Copyright by Dung Huu Douglas Phan 1993

All Rights Reserved

# Contents

---

<b>APPENDIX A: Detailed Domain Description:</b>	
<b>Electrical Utility Transmission Tower Engineering .....</b>	<b>185</b>
A.1 Tower Domain and Scope of Functional Analysis .....	186
A.2 Functional Decomposition of the Tower Engineering Process .....	186
A.3 Detailed Description of the Process .....	186
A.3.1 Transmission Line Analysis and Design Phase .....	187
A.3.2 Tower Structural Conceptual Design .....	189
A.3.3 Tower Structural Detailed Design .....	191
A.3.4 Tower Construction Planning .....	193
A.3.5 Tower Construction Execution .....	195
A.3.6 Tower Facility Management .....	196
A.4 Graphical Functional Schemata of the Process .....	197
<b>APPENDIX B: DEAL: Pseudo-Codes and</b>	
<b>A Detailed Analysis of “Transmission Tower Members” .....</b>	<b>217</b>
B.1 Pseudo-Codes for Operations in the DEAL-1 Procedure .....	218
B.2 Pseudo-Codes for Operations in the DEAL-2 Procedure .....	222
B.3 Detailed Analysis of the “Transmission Tower Members” Domain Entity .....	226
<b>APPENDIX C: Rules and Guidelines of the P-C Data Modeling Method .....</b>	<b>241</b>
C.1 Rules and Guidelines for the Refinement of Entities .....	242
C.2 Rules and Guidelines for the Design of Object Classes & Class Hierarchies	249
C.2.1 Rules and Guidelines for the Design of Object Classes .....	249
C.2.2 Rules and Guidelines for the Design of Class Hierarchies .....	252
<b>APPENDIX D: Documentation of the Tower Domain Primitive Schema .....</b>	<b>255</b>
<b>APPENDIX E: Documentation of Composite Classes</b>	
<b>Defined in the Testing .....</b>	<b>309</b>





# *List of Figures*

---

FIGURE A.1: Hierarchical Functional Decomposition of the Phases of the Tower Engineering Process into Functions. ....	187
FIGURE A.2: Legend for the Partitioned Data Flow Diagrams that follow. ....	198
FIGURE A.3: Diagram Notes for the Partitioned Data Flow Diagrams that follow. ....	199
FIGURE D.1: Legend for the Graphical Representations of Primitive Characterization Hierarchies that follow. ....	256
FIGURE D.2: Primitive Characterization Hierarchies Describing SPATIAL REFERENCE FORM. ....	257
FIGURE D.3: Primitive Characterization Hierarchies Describing GEOMETRY FORM. ....	266
FIGURE D.4: Primitive Characterization Hierarchies Describing TOPOLOGY FORM. ....	270
FIGURE D.5: Primitive Characterization Hierarchies Describing SHAPE REPRESENTATION FORM. ....	275
FIGURE D.6: Primitive Characterization Hierarchies Describing MATERIAL FORM. P. ....	285
FIGURE D.7: Primitive Characterization Hierarchies Describing PART DETAILING/FABRICATION FORM. ....	288
FIGURE D.8: Primitive Characterization Hierarchies Describing STRUCTURAL ENGINEERING BEHAVIOR. ....	293
FIGURE D.9: Primitive Characterization Hierarchies Describing STRUCTURAL ENGINEERING FUNCTIONS. ....	297
FIGURE D.10: Primitive Characterization Hierarchies Describing REQUIREMENTS. ....	300
FIGURE D.11: Primitive Characterization Hierarchies Describing DESIGN. ....	305
FIGURE E.1: Legend for the Graphical Representations of Composite Classes that follow. ....	312
FIGURE E.2: Composite Classes Representing Different User Views in the Data Uses Considered in the P-C Approach's Testing. ....	313



# Appendix A

---

## ***Detailed Domain Description: Electrical Utility Transmission Tower Engineering***

### ***Abstract:***

This appendix describes in detail the domain of electrical utility transmission towers to which we applied the P-C Approach. Specifically, it presents the six phases of the engineering process in this domain using the following format: (1) general description (time, key project participants, place, work involved and goal), (2) key terms and concepts, (3) description of the functions to which the phase is decomposed and (4) end results of the phase. Finally, it shows the graphical functional schemata portraying that engineering process using PANDA. These schemata present a concise, pictorial description of that process.

### ***Organization:***

*A.1 Tower Domain and Scope of Functional Analysis*

*A.2 Functional Decomposition of the Tower Engineering Process*

*A.3 Detailed Description of the Process*

*A.3.1 Transmission Line Analysis and Design Phase*

*A.3.2 Tower Structural Conceptual Design*

*A.3.3 Tower Structural Detailed Design*

*A.3.4 Tower Construction Planning*

*A.3.5 Tower Construction Execution*

*A.3.6 Tower Facility Management*

*A.4 Graphical Functional Schemata of the Process*

## ***A.1 Tower Domain and Scope of Functional Analysis***

---

Electrical utility transmission towers are large lattice structures supporting wires that transmit electrical power. The tower facility engineering process extends throughout the tower life cycle, from the initial need analysis to the possible final demolition of the structure. The process includes several stages of development. It typically involves participants from various disciplines, including electrical engineering, structural engineering, fabrication and construction management. Computer applications are used to automate certain design functions of the process, such as structural analysis and member design.

In the functional analysis<sup>†</sup> of this domain, we examined the entire process of engineering a transmission tower. However, we put less emphasis on the tower facility management that occurs after the tower is constructed. This facility management involves many scenarios in which various activities are carried out to keep the tower operational. It is difficult to cover all of those scenarios given our time and resource constraints. We also analyzed the initial electrical design stage of the process in less detail than some other stages. In tower structural design, we focused on designing new tower structures rather than retrofitting existing structures.

## ***A.2 Functional Decomposition of the Tower Engineering Process***

---

The tower facility engineering process involves programming, design, construction and long-term management of the facility. As shown in Figure A.1, we divided the entire process into the following phases: (1) *Transmission Line Analysis and Design*, (2) *Tower Structural Conceptual Design*, (3) *Tower Structural Detailed Design*, (4) *Tower Construction Planning*, (5) *Tower Construction Execution* and (6) *Tower Facility Management*. Phase 1 here corresponds to the programming of the facility, Phases 2 and 3 to the design, Phase 4 and 5 to the construction, and Phase 6 to the management. By and large, this breakdown is consistent with the one for commercial high-rise office buildings as defined in [Luth 91]. However, since transmission towers are less complicated than buildings, their life cycle is much simpler. Figure A.1 on the next page summarizes the functions into which the six phases of the tower engineering process are decomposed.

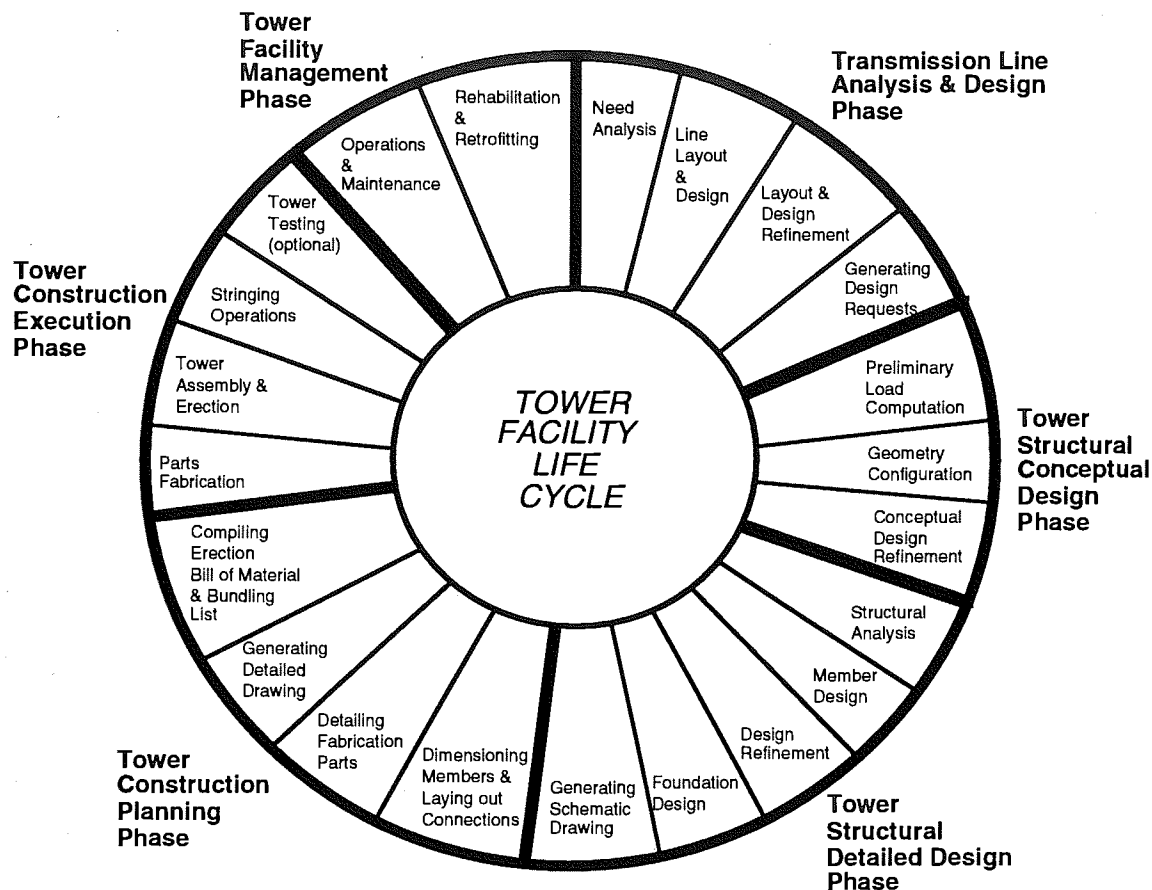
## ***A.3 Detailed Description of the Process***

---

In this section, we describe in detail all six phases of the tower engineering process. We use the following format consistently for all phases: (1) general description (time, key project participants, place, work involved and goal), (2) key terms and concepts, (3) description of the functions to which the phase is decomposed and (4) end results of the phase. This format reflects the way in which a domain expert would identify a particular phase. As an alternative, the reader can review the graphical functional schemata of the process using PANDA that are shown in the next section. Those schemata present a concise, pictorial description of the process.

---

<sup>†</sup> The information used in this functional analysis was provided and verified by design engineers of a utility company and also came from my own work experience in this area.



**FIGURE A.1: Hierarchical Functional Decomposition of the Phases of the Tower Engineering Process into Functions.** The functions are labeled within the sections of the circle. The phases are labeled outside of the circle. The heavier lines mark the beginning and end of each phase and help identify the functions that belong to that phase.

### A.3.1 Transmission Line Analysis and Design Phase

**General Description** This phase occurs at the beginning of the process when a need or economic opportunity for electrical utility services is perceived. The key participants are electrical engineers and structural engineers. This phase occurs at the engineers' workplace. It involves analyzing the perceived need and laying out and designing a new transmission line. The goal is to achieve an economic solution that fills the need and has an acceptable cost.

**Key Terms and Concepts** The following terms and concepts are used in this phase:

**Design Request** — An official document that requests the analysis or design of an existing or new tower type. It also specifies all the requirements needed for designing the tower type.

**Tower Design Requirements** — Statements of what is required of the tower's design (and even construction). For transmission towers, design requirements vary with the tower type cover electrical clearances, loading, strength and serviceability, constructibility, cost, right-of-way and tower dimensions.

**Functions** This phase consists of four functions:

- **Need Analysis:** The electrical engineer analyzes the perceived need for electrical utility services within a region. She considers the current electricity demand and supply in the region. She projects the region's future growth and the increase in its electricity consumption. The engineer then decides whether a new transmission line is needed. If so, he or she determines how much electrical power should be transmitted and what type of conductors should be used to transmit that power. This allows the engineer to determine the transmission line voltage that is necessary.
- **Line Layout and Design:** The engineer sets the direction of the transmission line, considering the geography and topology of the terrain. He then designs the types of towers needed to support the line. Specifically, the engineer determines the global attributes of those tower types, including the tower function classification, line angle, tower setting, etc. He also sets the location and orientation of individual tower structures used in the line.
- **Line Layout and Design Refinement:** After the line is laid out and designed, the electrical engineer approximates its total cost. She decides whether the existing line layout and design needs to be refined to obtain a lower-cost solution. At this point, the structural engineer may get involved by suggesting ways to produce tower structures that are lighter and thus lower cost. The cost of individual structures contributes to the total cost of the transmission line. The electrical engineer may take several iterations before reaching an acceptable solution. She then analyzes the cost/value ratio of the line and decides on its economic feasibility. When the electrical engineer decides that building the line is economically feasible, she finalizes the layout and design.
- **Generating Design Requests for Tower Types:** If the line is to be built, the electrical engineer then generates an official design request for each tower type used in the line. The design request contains all the information necessary to design the tower type, including its design requirements.

**Results** The end results of this phase are: (1) the layout data of the transmission line and (2) the design requests of the tower types. The line layout data includes the location, orientation and tower settings of the towers used in the transmission line. For each tower type, the design request contains the design requirements and global attributes of the tower type. The design requirements help define the constraints on the tower design and construction in subsequent phases. The global attributes describe:

- tower function classification, line angle, static-wire spans and conductor-wire spans,
- tower electrical characteristics, including voltage, number of circuits, circuit arrangement and minimum static shield angle,
- the types and properties of the electrical equipment the tower carries, specifically conductor and static types, conductor and static tensions, the number of conductor and static wires per phase, and the types of insulators and hardware.

A sketch showing a typical tower setting in the line may also be included in the design request.

### **A.3.2 Tower Structural Conceptual Design**

**General Description** This phase occurs as soon as the structural engineers receive the design request. The key participants are structural engineers, and this phase occurs at their workplace. It involves calculating the loads applied to the tower structure and configuring a tower geometry. The goal is to obtain a tower geometry that both meets the requirements (including the loading requirements) and will result in an economical design.

**Key Terms and Concepts** The following terms and concepts are used in this phase:

**Loading Condition** — A description of a scenario in which a tower structure would be subjected to external environmental loading. For utility transmission towers, the environmental loading can be gravity, wind, ice, temperature, ground motion, impact forces on the structure, conductor-wire tension (under normal conditions, broken wire conditions and construction and maintenance conditions), etc.

**Load Case** — A particular way in which the loading condition might occur. For example, a hurricane extreme loading condition with 100 mile-per-hour winds may produce several different load cases, each corresponding to a different wind direction: perpendicular to the transverse face of the tower, perpendicular to the conductor wires, at 45 degrees to the conductor wires, at every 15-degree increment from the tower bisector, etc.

**Load** — An external force applied to a structure in a certain load case. A description of a load includes its magnitude, direction, location, type (i.e., axial, moment, torsion) and form (i.e., concentrated, linear, per area).

**Load Tree** — A schematic representation of loading from a load case on a diagram of the structure. A conventional load tree shows all the load vectors from that load case at the location where they are applied on the structure. Alternatively, a combined load tree shows a resultant load vector at each significant location on the structure.

**Tower Structural Systems** — For transmission towers, the four major types of structural systems are (1) leg systems, (2) lacing systems, (3) arm systems and (4) redundant systems. Leg, lacing and arm systems are the “primary systems” of the structure that resist loading. Redundant systems are the “secondary systems” that mainly increase the stiffness and reliability of the primary load-resisting systems.

**Member** — A conceptualized component of a system that serves a particular function. “Primary members” (e.g., leg and lacing members) are members of primary systems, whereas “secondary members” (e.g., redundant members) belong to secondary systems.

**Functions** This phase consists of three functions: (1) Preliminary Load Computation, (2) Geometry Configuration and (3) Conceptual Design Refinement. These functions are highly interdependent: Loads on the tower structure cannot be computed until a preliminary tower geometry is obtained, and a tower geometry cannot be generated until the loads are computed. This underlines the complexity of the tower design synthesis. In practice, the engineer uses his or her design experience as well as an iterative approach to find a solution. The detailed description of the functions is as follows:



- **Preliminary Load Computation:** Given the tower electrical clearances, the structural engineer first approximates the spatial arrangement of the static level, the conductor levels and the tower body. At this point, she needs a preliminary geometry of the tower in order to compute the loads. She may use the geometry of a similar tower type as a starting point. Alternatively, the engineer may use rules of thumb or past design experience to roughly configure a new geometry. She also determines all the necessary *loading conditions* if they were not completely specified in the design request. Using the loading conditions, she defines the principal load cases. For each load case, the engineer then calculates the loads and generates a load tree (either conventional or combined).
- **Geometry Configuration:** Having generated the load trees, the structural engineer can calculate the optimum base spread (i.e., the larger dimension at the base of the tower) that can sustain the tower loading. Next, he determines the cross-sectional shape of the tower. Following standard practice, the shape can be either square or rectangular. This decision depends on the way loading is applied to the structure and on the way the structure would distribute that loading. The engineer then determines the load paths and the tower structural systems suitable for those load paths. These systems include the leg, arm, lacing and redundant systems of the tower structure. To select the bracing pattern of the redundant systems, the engineer must give great attention to the overall stability of the system. Geometrically unstable redundant systems can cause premature failure of the primary members. In designing all those structural systems, the engineer also works out the complete details of the tower geometry and topology. In short, the geometry of a tower type is determined by several factors, including the number of circuits, the electrical clearances, the type of insulators (i.e., their length and maximum transverse displacement), the amount of vertical deflection of the conductor wires at the attachment points, and the economy that the engineer is trying to achieve.
- **Conceptual Design Refinement:** After one iteration of the above functions, the structural engineer goes back and calculates the load trees based on the existing geometry. She then refines the existing geometry using those load trees. This process continues until a satisfactory geometry is obtained. The aesthetic impact of the tower may be the final consideration in configuring its geometry.

**Results** This phase generates a large amount of data. First, it produces data that describes the individual loads of the load trees from different load cases (i.e., magnitude, direction, location, type and form). Second, this phase produces data of the tower geometry, including: (1) global geometric data of the tower (i.e., tower height, panel heights, cage width, base spread, taper ratio, extension heights, bend line elevation, etc.) and (2) detailed geometric data of the systems and members in the tower body and the static and conductor arms. The second set of data includes spatial data (i.e., coordinates, orientation, spatial envelope dimensions), geometric data (i.e., shape, dimensions, etc.) and topological data (i.e., topological representation and connectivities). The structural engineer also knows about the functions of the tower's systems and members. (These functions are resisting loads, implementing load paths, transferring loads, supporting members, bracing members, etc.) Unfortunately, this knowledge may not be represented in any format or documented in any source.

### **A.3.3 Tower Structural Detailed Design**

**General Description** This phase occurs when the structural engineer has a satisfactory tower geometry and is in a position to design the structure. The key participants are structural engineers and foundation engineers (civil engineers who specialize in the field of foundation engineering). This phase occurs at the workplace or workplaces of these participants. It involves carrying out the detailed design of the structure and communicating the design information, by means of drawings and written specifications, to those who will detail, fabricate and erect the structure. Given the tower geometry determined during the previous phase, the goal is to obtain a light-weight structure that meets all the strength and serviceability requirements.

**Key Terms and Concepts** The following terms and concepts are used in this phase:

**Tower Anchoring Devices** — Devices used to connect the structure to the foundation. There are two common types: (1) base shoes used with anchor bolts and (2) stub angles. A “base shoe” is a welded assembly that consists of an angle and a base plate. “Anchor bolts” are special bolts used to anchor the tower structure to the foundation. A “stub angle” is a special angle member that is embedded in the concrete foundation and bolted to the tower legs.

**Schematic Drawing** — A drawing of the tower type’s structure that the structural engineer produces at the end of the detailed design phase. The drawing is intended to communicate the tower design information to the detailer, fabricator and construction crew. Generating a schematic drawing involves putting the data generated in the required presentation format, and making any special notes or specific details to the detailer, fabrication and construction crew.

**Element** — An analysis component of a system that corresponds to a particular member of the system.

**Functions** This phase consists of five functions: (1) Structural Analysis, (2) Member Design, (3) Design Refinement, (4) Foundation Design and (5) Generating Schematic Drawing. The first three functions are highly interdependent: The structural analysis cannot be carried out until the member sizes are known, and the member design cannot be done until the members’ stresses and deflections from the analysis are obtained. The design refinement involves iterating over the preceding two functions to improve the design. In practice, the structural engineer also uses his or her design experience to find a solution. A detailed description of these functions is as follows:

- **Structural Analysis:** The structural engineer decides on the material (e.g., steel) and material grade (e.g., A-36) of the tower members. (Transmission towers today are built out of steel or aluminum.) The engineer also assumes the member sizes and analysis element (i.e., truss, beam, column, beam-column, etc.). To do this, the structural engineer uses his or her design experience and the data used in the design of similar tower structures in the past. He or she then carries out a structural analysis of the tower. Since transmission towers are highly indeterminate structures, they are analyzed using commercial finite-element analysis programs. The engineer also uses the loads and the tower geometry from the previous phase to prepare data for the input file of the analysis program. By running the analysis, the engineer obtains the behavior of the structure under the specified load cases. The output data includes the stresses, deflections and end reactions of the members.

- **Member Design:** Using the data generated by the analysis, the structural engineer checks the assumed member sizes for strength and serviceability. The members' capacity must sustain the controlling stresses and deflections from all load cases. The engineer considers the strength and serviceability requirements of standard design codes such as [SCPUC 69], [ASCE 71], [ANSI 82], [SAE 88] and [AISC 89]. If a member fails, she tries another size until a satisfactory size is obtained. She also verifies the end conditions—i.e., the analysis model—of the members assumed in the structural analysis. A re-analysis is necessary if there is a large discrepancy between the members and elements assumed and those designed. Then, for each structural member, the engineer also determines the number, pattern and diameters of bolt holes required to transmit the member's internal loads. This information will be used to lay out the connections between members and to specify the fabrication details in the next phase.
- **Design Refinement:** Third, after one iteration of structural analysis and member design, the structural engineer might iterate over those two functions in order to design a structure that uses smaller member sizes and thus has a lighter weight. This function of the process is called design refinement rather than design optimization because its objective is to find a "good enough" solution rather than the best solution. Such a solution would meet all design constraints, would have acceptable accuracy in the analysis, and could be found in a reasonable time using the available human and computational resources. Also note that in this function, as in the previous two functions, loading, strength and serviceability, constructibility and cost requirements are carefully considered.
- **Foundation Design:** Last, using the controlling reaction loads at the tower base from all load cases, the foundation engineer designs the tower foundation. This involves designing the concrete foundation as well as the anchoring devices of the tower.
- **Generating Schematic Drawing:** The engineer produces a schematic drawing of the structure to communicate the information to the detailer, fabricator and construction crew.

**Results** This phase results in a large amount of design information. This information includes the member sizes (e.g., angle L 8 x 8 x 1/2), dimensions, cross-sectional properties, analysis end conditions, stresses, deflections, and reaction loads, and the number and pattern of bolt holes. Moreover, the schematic drawing of the tower type displays the following:

- (1) drawing title, which includes the designation and version number of the drawing as well as the designation of individual sheets,
- (2) a schematic diagram of the tower structure showing its geometry and topology,
- (3) global tower geometric data such as tower height, panel heights, cage width, base spread, taper ratio, extension heights, bend line elevation, etc.
- (4) typical sizes (e.g., L 8 x 8 x 1/2) of the leg, lacing and redundant members,
- (5) loading conditions for which the structure is analyzed and designed, and even references to the load calculation,

- (6) general notes to the detailer about specifications, bolt sizes, steel material grades, member framing, drawing layout, etc., and
- (7) any specific details regarding member framing, member splices, static and conductor wire connections, tower-to-foundation connections, etc.

This schematic drawing is used in the next phase.

### **A.3.4 Tower Construction Planning**

**General Description** This phase occurs once there is a finished schematic drawing that can be used to work out all the necessary construction details. The key participants are structural detailers. However, when detailing problems (e.g., missing design information, uncommon member sizes) arise, the detailers cooperate with the structural designers to resolve the problem. This phase occurs at the workplace of the detailers, who can be either company or contract employees. The phase involves developing all the fabrication and erection details in preparation for tower construction. The goal is to plan the next construction phase in order to minimize errors and maximize efficiency.

**Key Terms and Concepts** The following terms and concepts are used in this phase:

**Fabrication Part** — A single piece to be fabricated and used in the construction of the structure. It corresponds to a member or connection in the structure. For transmission towers, the part can be an angle member or a connection plate. However, a member or connection may have more than one fabrication part.

**Fabrication Feature** — A single specification of how a fabrication part should be made out of raw material. There are numerous fabrication features. For transmission tower members, fabrication features include dimensions, the hole pattern, hole sizes, edge preparation, edge clipping, gage line, etc.

**Mark Number** — An identification mark printed or stamped on a fabrication part.

**Working Point** — A point of reference that is selected from the tower structure's schematic diagram and used to work out all detailed dimensions of the members.

**Detailed Drawing** — A drawing of the tower type's structure produced by the detailer at the end of the construction planning phase. The drawing is intended to communicate to the fabricator and construction crew the detailed information necessary to fabricate and construct the tower in the subsequent phase. Generating a detailed drawing involves putting the data generated in the required presentation format, specifying the fabrication features of the tower members, and making any special notes or specific details to the fabrication and construction crew.

**Functions** This phase consists of three functions:

- **Dimensioning Members and Laying out Connections:** From the schematic drawing, the detailer establishes the main working points of the tower structure. Using these working points, he or she calculates the overall length, slope and bevel of all members. To start laying out the connections, the detailer reviews any special notes about member framing on the schematic drawing. Having established the member lengths, she lays out each connection. The layout data generated for the connection includes its plate shape and size, hole pattern, and required ringfill and bolt length. The detailer also determines the member clearances that are needed to avoid

interferences. In the next function, these clearances are used to determine the exact lengths of the fabrication parts that correspond to the members.

- *Detailing Fabrication Parts:* For each member or connection, the detailer determines the number of fabrication parts needed. He gives each fabrication part a unique mark number. He then details the fabrication part by specifying its fabrication features, such as dimensions, number of holes out, hole pattern, hole diameter, edge preparation, edge clipping, gage line, etc. To do this, the detailer uses the member sizes, steel material grades and any special notes from the schematic drawing. He also uses the connections' layout data from the preceding function.
- *Generating Detailed Drawing:* The detailer first generates the bolt schedule, which includes all the hardware (bolts, nuts, ringfills, etc.) needed to assemble the tower. She then computes the raw (or black) and total (or galvanized) weights of the basic tower and extensions. Last, she puts together the detailed drawing of the tower, which is a detailed graphical and textual representation of the tower structure. It includes all the information necessary to fabricate and construct the structure. As the final result of the design and construction planning, the detailed drawing communicates both the design information and the designer's intention to the fabricator and field construction crew. Therefore, the drawing must be unambiguous, readable, concise and complete as possible. Moreover, almost all detailed drawings currently used are paper-based.<sup>¥</sup>
- *Compiling Erection Bill of Material and Bundling List:* The detailer compiles a detailed list of all fabrication parts. He groups these pieces into separate bundles, taking into account their size, length, quantity and weight. The purpose of this function is to facilitate shipment from the fabrication shop to the site, as well as site handling by the construction crew. The result is the erection bill of material and bundling list that shows by bundles all fabrication parts of the tower structure.

**Results** All design data for the tower structure is generated by the end of this phase. The results are the detailed drawing and the erection bill of material and bundling list. The detailed drawing displays the following information:

- *Erection diagrams*, which include a foundation setting plan and detailed sketches of the basic tower and extensions. These sketches indicate the mark numbers of all fabrication parts as well as bolt counts and bolt lengths at each connection.
- A *bolt schedule*, which lists all the hardware (bolts, nuts, ringfills, etc.) needed to assemble the tower.
- Loading conditions similar to those shown on the schematic drawing.
- Tower weights, including the weights of the basic tower and the tower used with different extensions. Each weight is the sum of the galvanized weight of the tower and the weight of the hardware.

---

<sup>¥</sup> Better computer-aided design tools can significantly improve this function by automating the generation of detailed drawings in electronic form. Such drawings are a more error-free and cost-effective means of communicating design information.

- Details of fabrication parts, which include mark number, quantity, size, dimensions, material type and grade, fabrication features and any special fabrication and erection notes.

The erection bill of material itemizes the fabrication parts, whereas the bundling list shows how these pieces should be grouped together. These two results are usually combined into one deliverable. Together, they show by bundles the mark number, quantity, shape (e.g., L-shape for angle), size (e.g., 8 x 8 x 1/2), length, and approximate weight—since the true weight can only be known accurately after fabrication is completed—of all fabrication parts. It also includes any additional written remarks about these pieces.

Together, the detailed drawing and the erection bill of material and bundling list are a comprehensive, but not complete, representation of the tower design data.

### **A.3.5 Tower Construction Execution**

**General Description** This phase occurs as soon as the fabricator who successfully bid the contract for tower fabrication receives the approved purchase order as well as the detailed drawing of the tower type. The key participants are the fabricator, the material supplier, the construction manager and field construction crew. This phase occurs at the fabricator's shop and at the site where the tower is to be erected. However, the electrical engineers and the structural designers of the tower type are also involved in inspecting the construction work at different stages. In addition, they provide their expertise in solving problems that arise during construction. If prototype testing of the tower type is to be carried out, structural engineers are heavily involved in developing the test specifications and monitoring the test procedure. This phase involves constructing the tower structure, as planned in the previous phase. (Actually, several structures of the same tower type may be installed in one transmission line). The goal is to minimize errors, damage and cost overruns during construction.

**Key Terms and Concepts** The following terms and concepts are used in this phase:

**Black (or Raw or Ungalvanized) Steel Parts** — Steel fabrication parts right after fabrication and before galvanization. The weight of all black steel pieces in the tower is called the *black weight*.

**Galvanized Steel Parts** — Black steel fabrication parts after being subjected to a galvanization process. Galvanization uses zinc coating to protect steel against weather corrosion. During galvanization, the black steel pieces are subjected to extensive surface preparation and prefluxing and are then immersed in molten zinc.

**Functions** This phase consists of four functions:

- **Parts Fabrication:** The fabricator reviews the detailed drawing and the erection bill of material and bundling list. If she detects any errors, inconsistencies or complications (e.g., material grade is not available at the time), she contacts the detailer or possibly the structural designer and works out the problems. After having a working detailed drawing, the fabricator then prepares material take-off lists. At this point, the fabricator procures the necessary raw material from warehouses or steel mills. Fabrication begins after the material supplier delivers the raw material as ordered. Today, fabrication is done using advanced computer-aided manufacturing technology and in particular, numerically controlled programmable equipment.

After making all the parts, the fabricator cleans them up to remove mill scale, dirt and grease. Then, he galvanizes these *black steel parts* using an time-consuming process. He carefully inspects the *galvanized steel parts* for uniformity, appearance and defects. Finally, he bundles and ships them to the site.

- *Supplying Raw Steel Material:* The material supplier gathers the raw steel material as ordered by the fabricator and then makes the delivery. Parts fabrication resumes as soon as the material was delivered.
- *Tower Assembly and Erection:* The construction crew opens the bundles at the site where the tower is to be erected. They assemble as much of the tower on the ground as possible: tower arms, extensions, cage and panels below the bend line. Problems generally arise during the tower assembly. Member fit, for instance, is a major concern. Parts that do not fit together create complications and slow down the construction process. Revisions to the tower's detailed drawing might be required to avoid similar problems in the future. (Constructibility knowledge should be used early in the tower design and detailing to eliminate problems of this nature.) Next, the construction crew lifts the assembled sections by crane, put them in place and connect them. They then lift the assembled tower, place it onto its stub angles or base shoes, and anchor it to the foundation. (Note that small tower structures can be completely assembled and then lifted onto the foundation, whereas larger structures are usually erected section by section.)
- *Stringing Operations:* After erecting enough towers in the transmission line, the crew carries out stringing operations to install the insulators and static and conductor wires on the towers. The wires are then pulled up to the cable tension specified in the original Design Request and in the detailed drawing.

The tower testing function is optional and not discussed here.

**Results** This phase results in the tower structures constructed and the transmission line installed. The basic difference between this phase and the building construction execution phase in [Luth 91] is that the construction sequence and methods used are different for the two kinds of structures.

### **A.3.6 Tower Facility Management**

**General Description** This phase occurs after the tower is constructed. As with buildings, it corresponds to the remaining period in the tower facility life cycle. Depending on the specific work involved, the participants are the people who have been involved in the design and construction of the tower. Therefore, this phase can occur at the engineers' workplace or in the field. Simply put, this phase involves managing the constructed tower. The goal is to make sure that the tower is operational and thus serve its functions throughout its life span.

**Functions** This phase consists of two functions: (1) Operation and Maintenance and (2) Rehabilitation and Retrofitting. These two functions are independent of one another and can be concurrent. A detailed description of these functions follows:

- *Operations and Maintenance:* This function includes post-construction activities aimed at improving or maintaining the conditions of the tower in order to make it operational. For example, a wide-flange member may be added on each side of the

tower cage at each level of conductor arms. This addition enables a construction worker to stand on it and work on the tower. These post-construction activities can also be aimed at protecting the tower structure against weathering effects, structural failures, acts of God, sabotage, etc. Member repair also falls into this category. However, the activities here do not change the purposes for which the tower was designed and constructed. In addition, they do not involve major re-analysis or re-design of the tower structure.

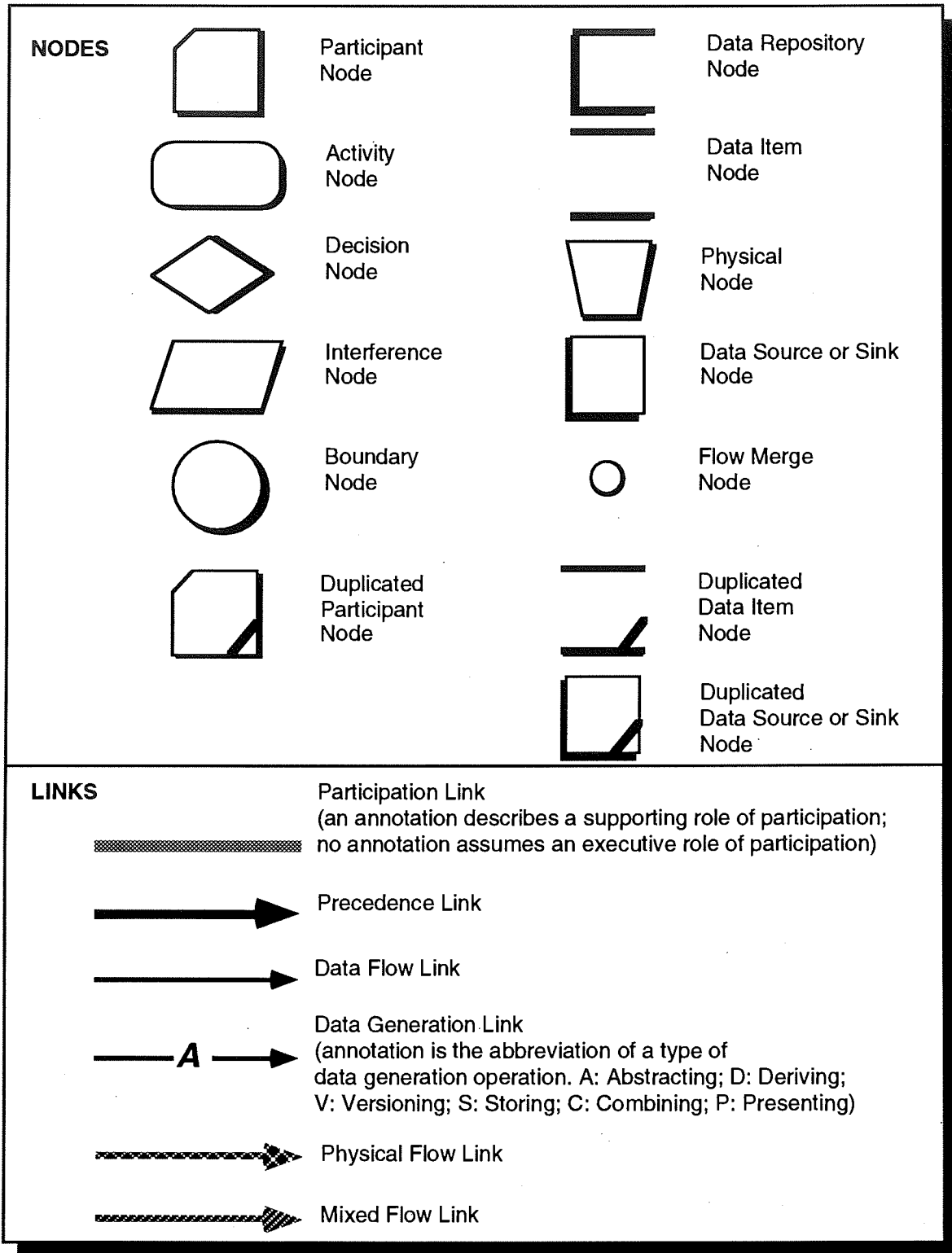
- **Rehabilitation and Retrofit:** In contrast to the above function, this function includes post-construction activities aimed at changing the purposes for which the tower was designed and constructed. These activities generally involve re-analysis and/or re-design of the tower structure. Re-analysis and re-design may be necessary for a number of reasons, including: new design code requirements, different construction methods of the tower structure, addition of new electrical equipment to the structure, a different type of foundation and/or foundation construction methods, a different geographic location, a different electrical facility usage of the tower (i.e., usage of the tower with a different voltage, line angle, tower function classification, static and conductor wires, other electrical equipment, etc.).

**Results** The result of this phase is an operational tower. The basic difference between this phase and the building facility management phase in [Luth 91] is that operation and maintenance are defined here as one coherent function instead of two separate functions.

#### **A.4 Graphical Functional Schemata of the Process**

This section presents a series of Partitioned Data Flow diagrams (or P-diagrams) that are the graphical function schemata of the tower engineering process described in the preceding section. Due to space constraints, Figures A.2 and A.3 first show the legend and diagram notes for all the P-diagrams that follow. The legend includes the graphical symbols for the concepts of PANDA. Diagram notes are general notes that applies to an entire diagram that specifically refers to them. Only reference notes (i.e., annotations of nodes or links using a reference number or symbol) are shown directly on the diagram. Then, the first diagram illustrates the process' highest-level skeleton functional schema with all six phases. The remaining diagrams show the more detailed functional schemata of the first five phases. As explained in the beginning, the last tower facility management phase is not shown here.



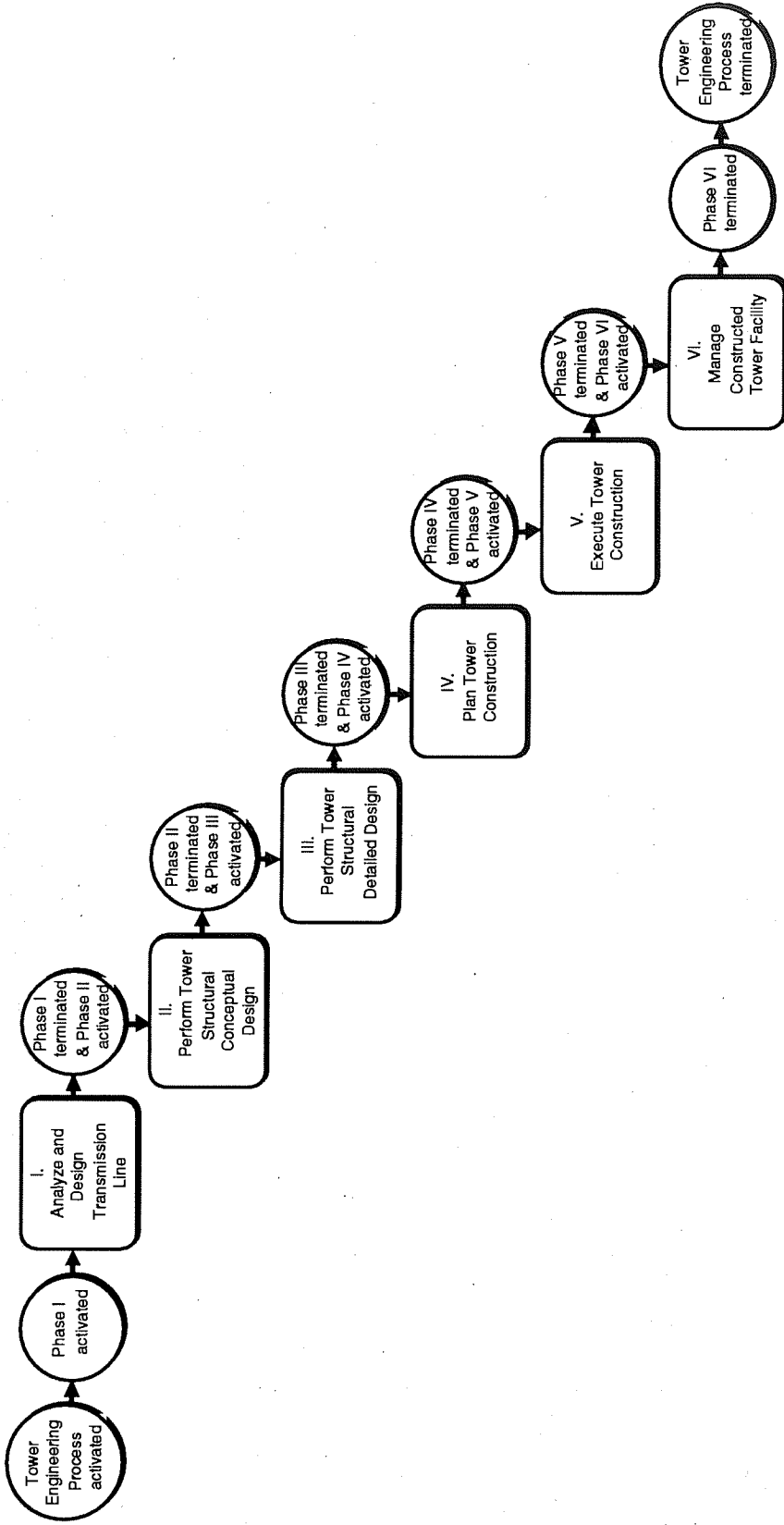


**FIGURE A.2: Legend for the Partitioned Data Flow Diagrams that follow.** The graphical representations of duplicated nodes were inspired by Gane's and Sarson's version of the Data Flow model [Gane 79].

- A. This diagram is the highest-level skeleton graphical functional schema of the process. It illustrates the breakdown of the process into several major phases. This schema results from the first step, 1.1, of the methodology presented in Section 4.4.1 of Chapter 4. At this level, the three partitions do not show the details about the participants, activities, data, material and products. Successive top-down functional decomposition of this skeleton schema would reveal those details. Indeed, the diagrams that follow this would give more details for the individual phases and their functions and activities.
- B. This diagram is an intermediate skeleton graphical functional schema of a phase. It illustrates mainly the breakdown of a phase into several functions. This schema results from the first step, 1.1, of the methodology presented in Section 4.4.1 of Chapter 4. At this level, the three partitions still do not reveal all the details about the participants, activities, data, material and products. Also, the schema does not necessarily show all the design loops and iterations that could possible occur in this phase. In fact, the diagrams that follow this would reveal all of the aforementioned details for the individual functions of this phase.
- C. This diagram is a detailed skeleton graphical functional schema of one or more functions. It illustrates the breakdown of the function or functions into several activities. It shows all the details in three partitions: (1) Participants, (2) Process and (3) Data, Material and Products. This schema results from completing all the steps in the two passes of the methodology presented in Section 4.4.1 of Chapter 4.

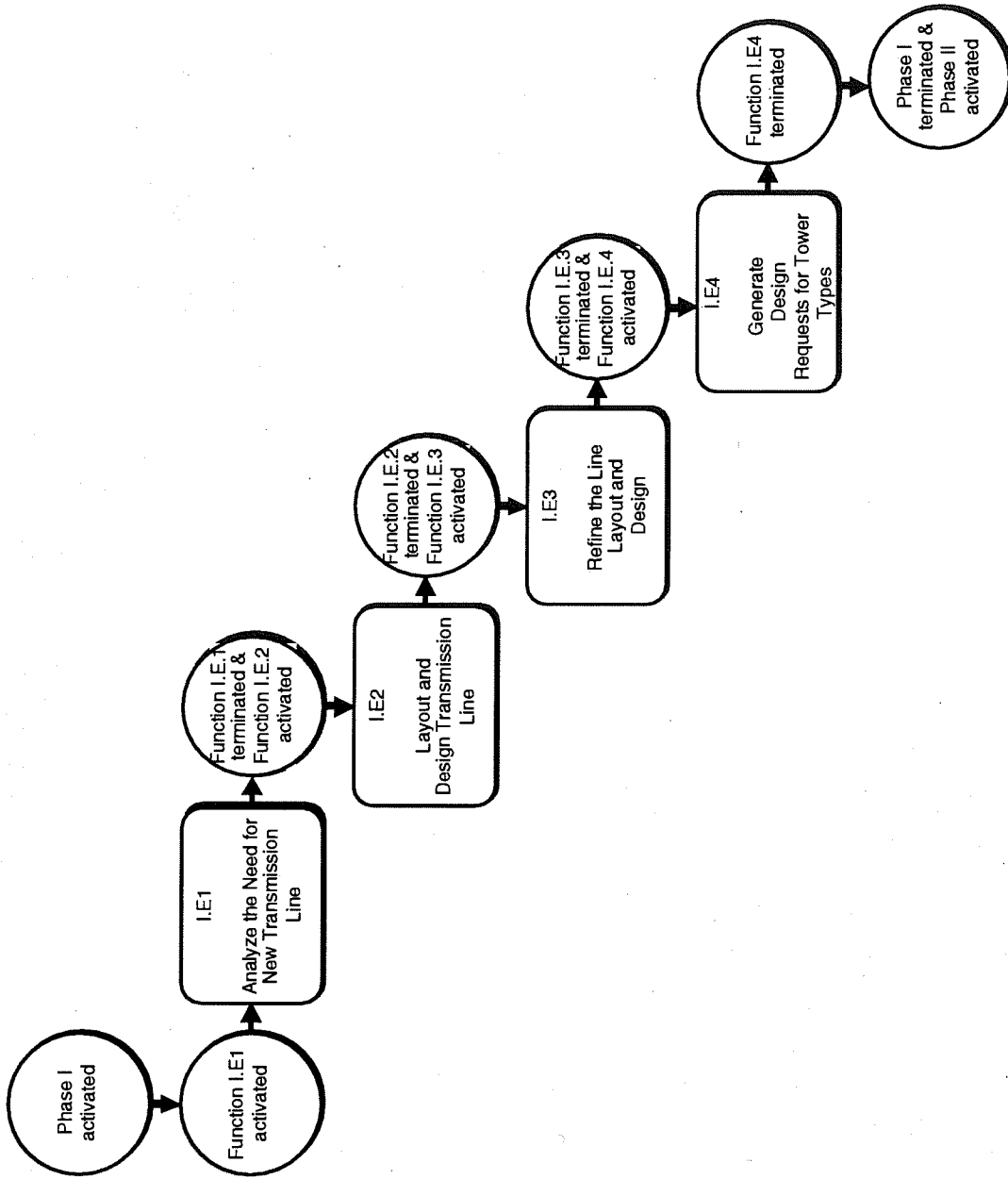
**FIGURE A.3: Diagram Notes for the Partitioned Data Flow Diagrams that follow.**

**PARTITION II: PROCESS**

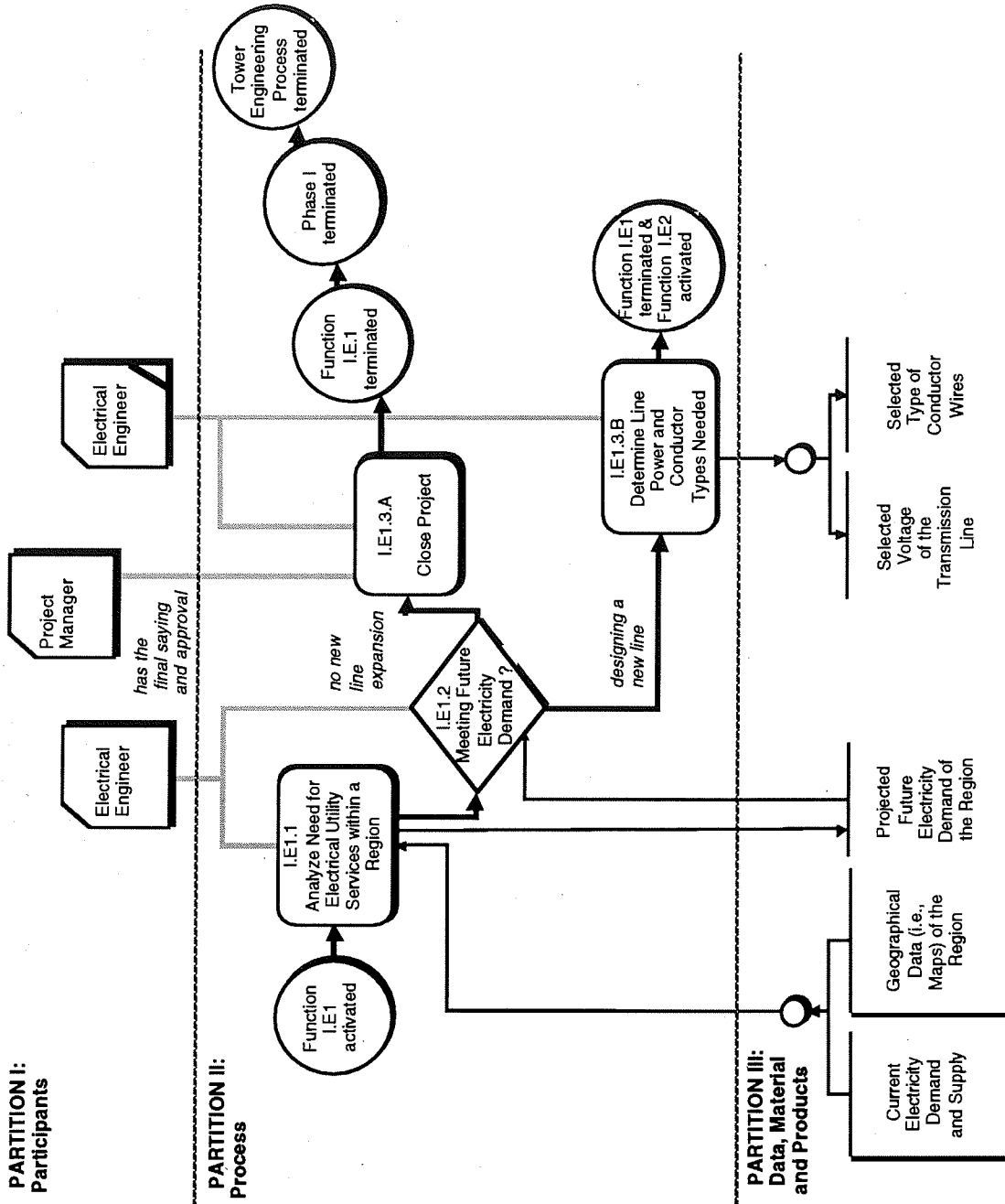


**P-DIAGRAM 0: Highest-level Skeletal Graphical Functional Schema of the Electrical Utility Transmission Tower Facility Engineering Process.** (See Diagram Note A)

**PARTITION II: PROCESS**



**P-DIAGRAM 1: Intermediate Skeletal Graphical Functional Schema of Phase I, Transmission Line Analysis and Design.**  
 (See Diagram Note B)



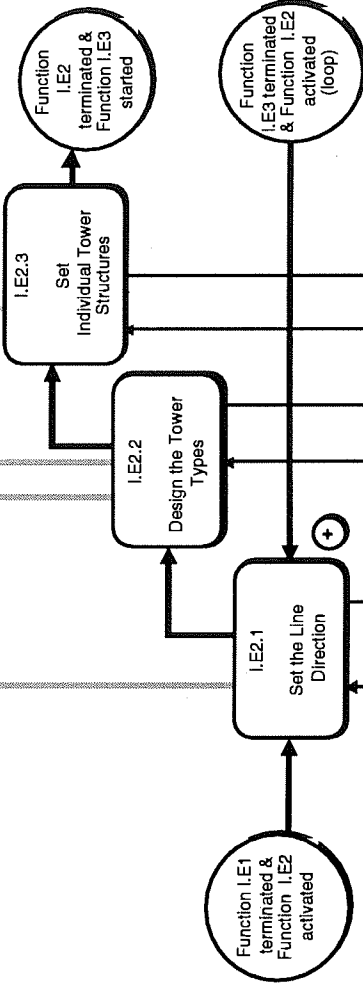
**P-DIAGRAM 1.1: Function I.E1, Need Analysis, of Phase I (Transmission Line Analysis and Design).** (See Diagram Note C)

**PARTITION I:  
Participants**

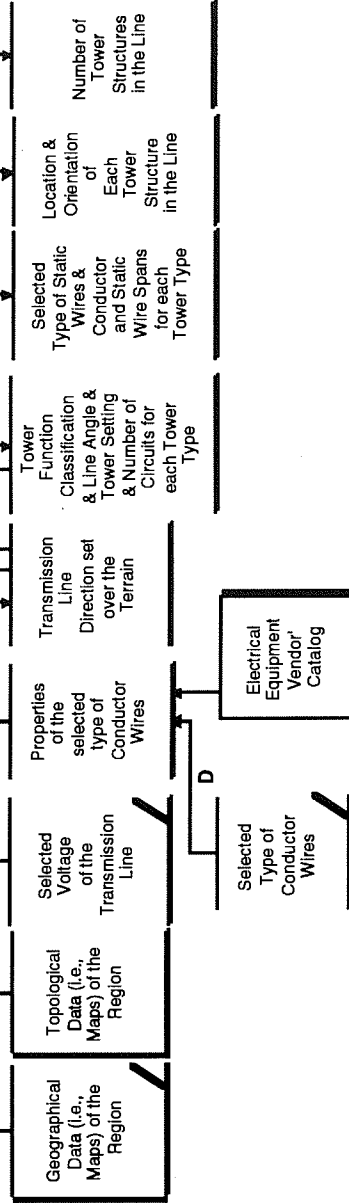


provides expert inputs

**PARTITION II:  
Process**



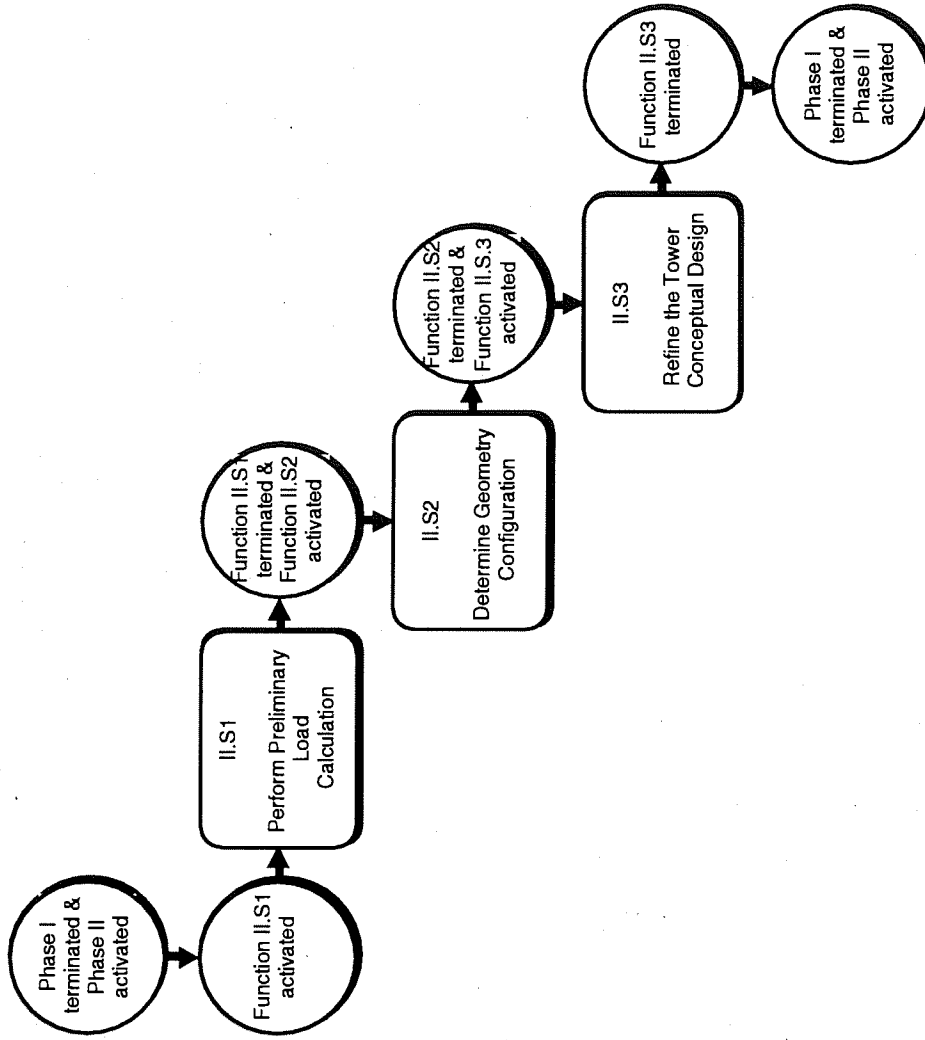
**PARTITION III:  
Data,  
Material  
and Products**



**P-DIAGRAM 1.2: Function I.E2, Line Layout and Design, of Phase I (Transmission Line Analysis and Design).**  
(See Diagram Note C)



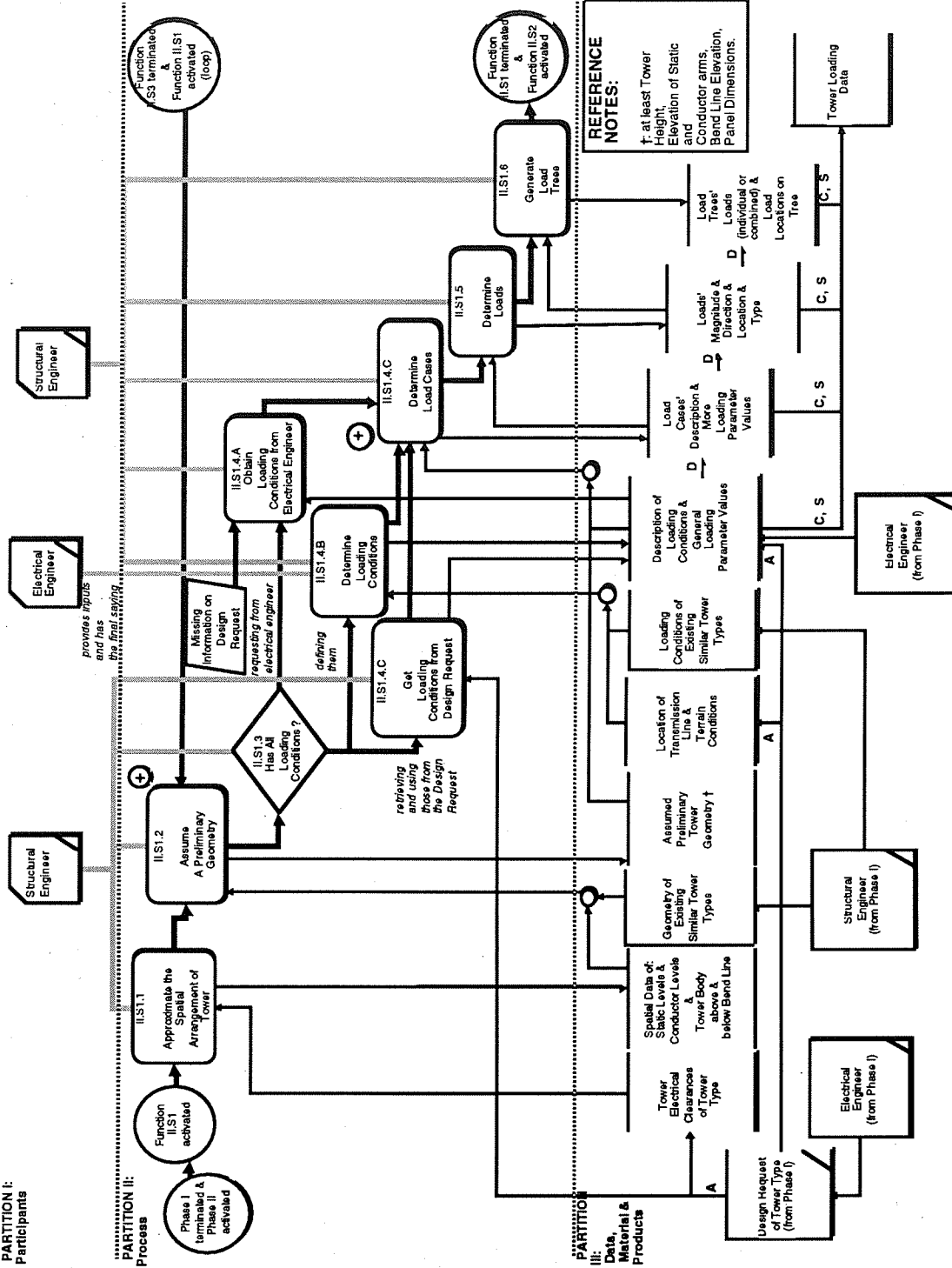
**PARTITION II: PROCESS**



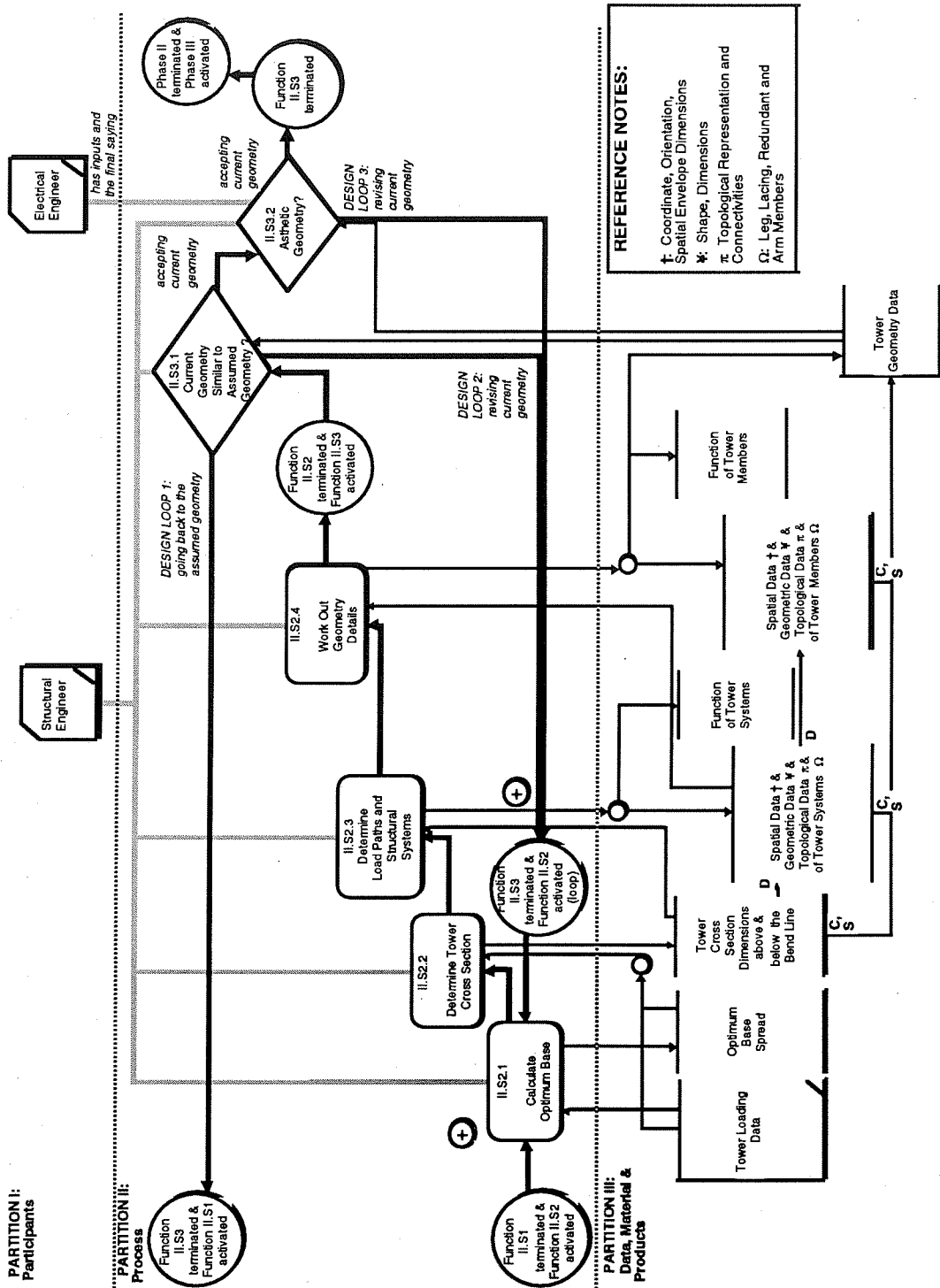
**P-DIAGRAM 2: Intermediate Skeletal Graphical Functional Schema of Phase II, Tower Structural Conceptual Design.** (See Diagram Note B)



**PARTITION I:  
Participants**

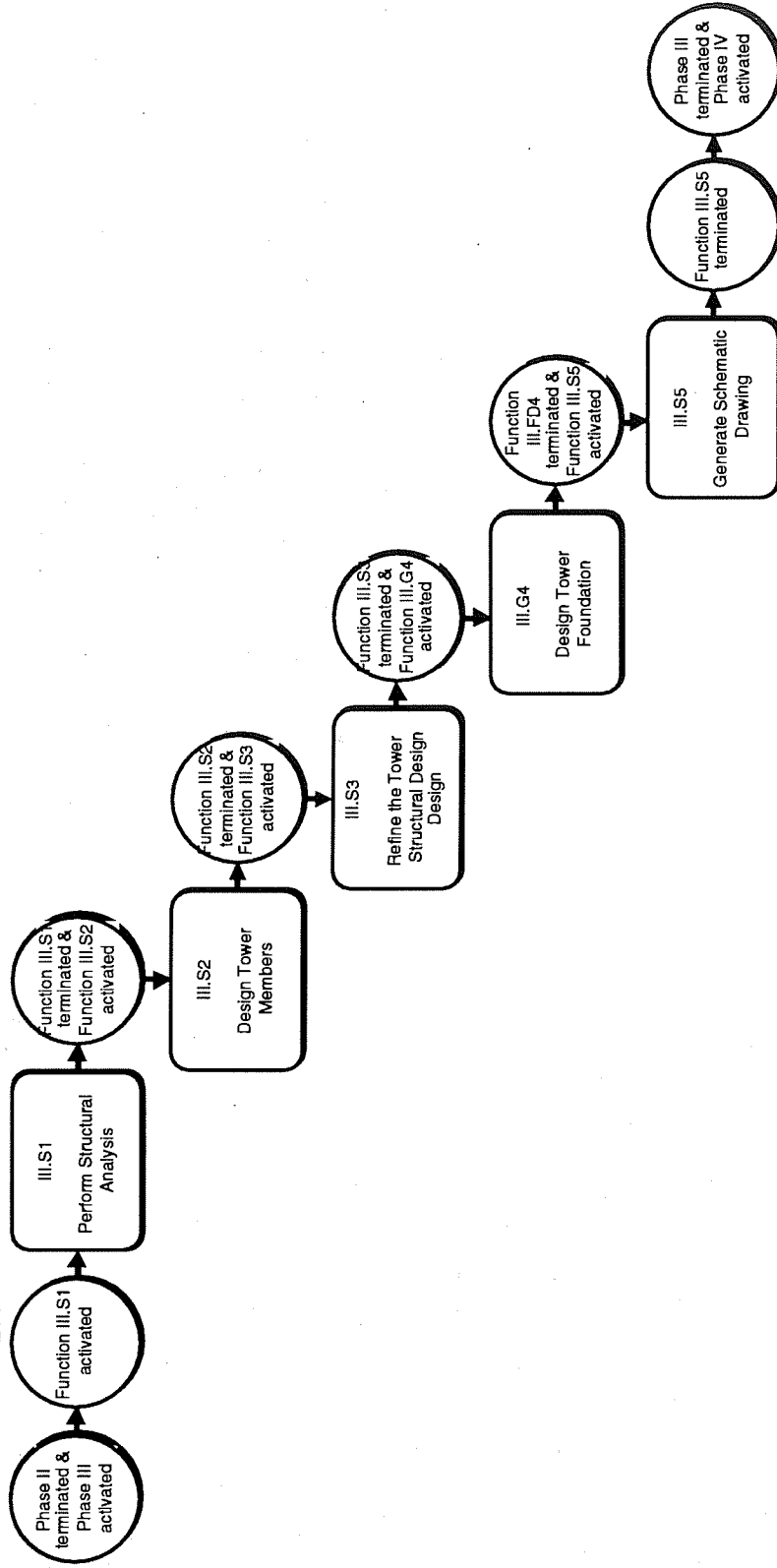


**P-DIAGRAM 2.1: Function II.S1, Preliminary Load Computation, of Phase II. (See Diagram Note C)**



P-DIAGRAM 2.2: Function II.S2, Geometry Configuration, and Function II.S3, Conceptual Design Refinement (Phase II). (See Diagram Note C)

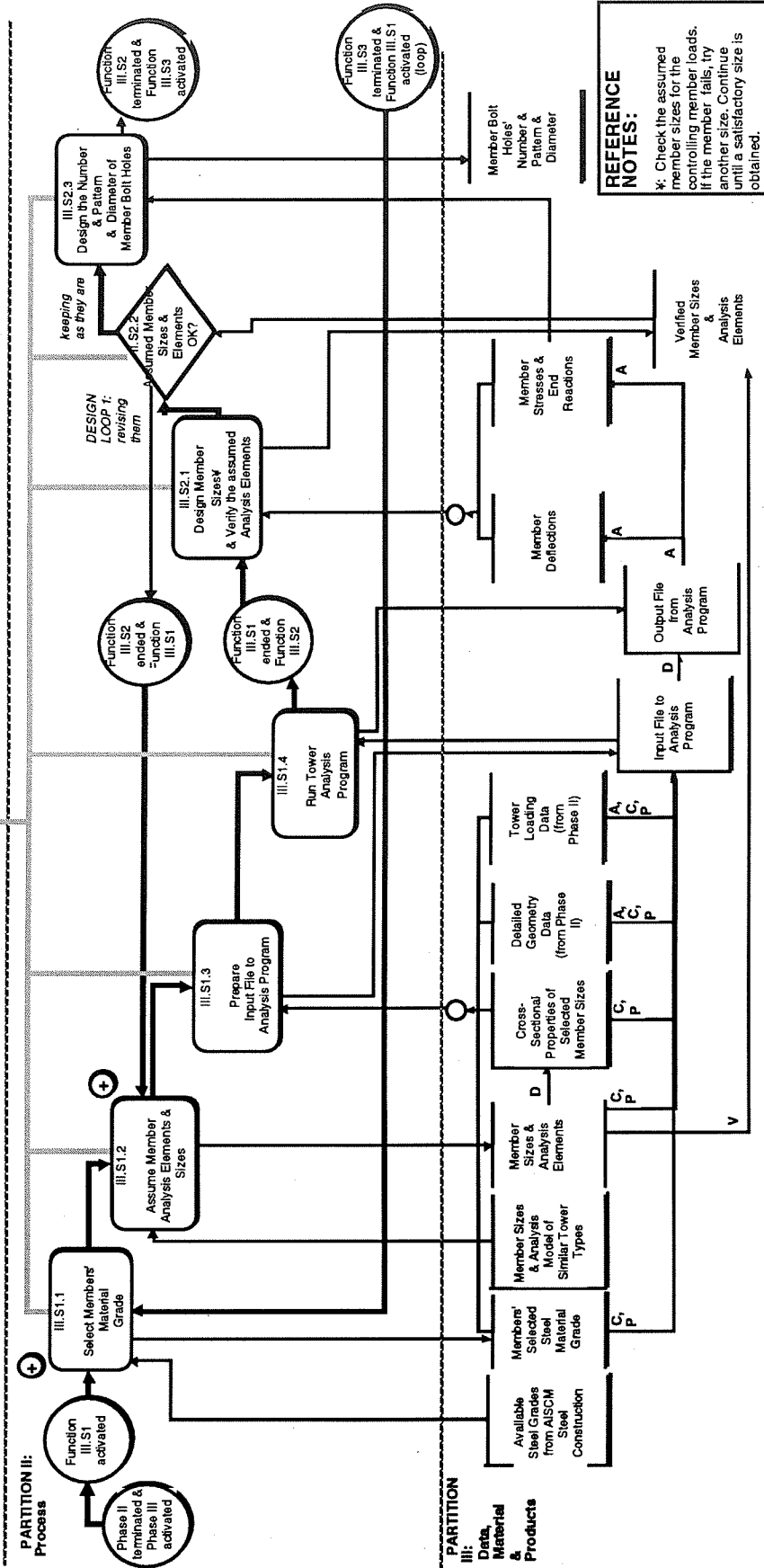
**PARTITION II: PROCESS**



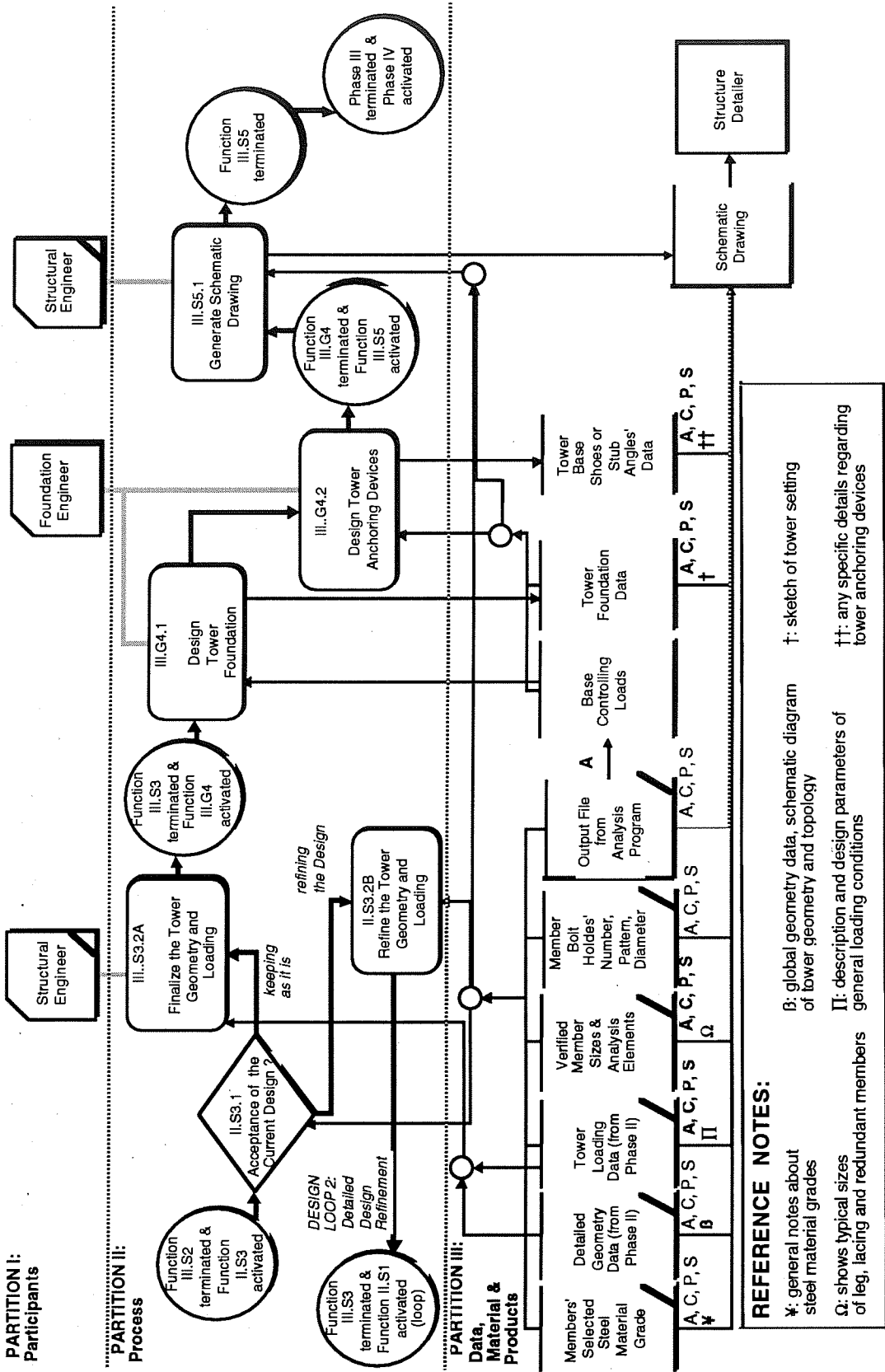
**P-DIAGRAM 3: Intermediate Skeletal Graphical Functional Schema of Phase III, Tower Structural Detailed Design.** (See Diagram Note B)

**PARTITION I:  
Participants**

Structural Engineer

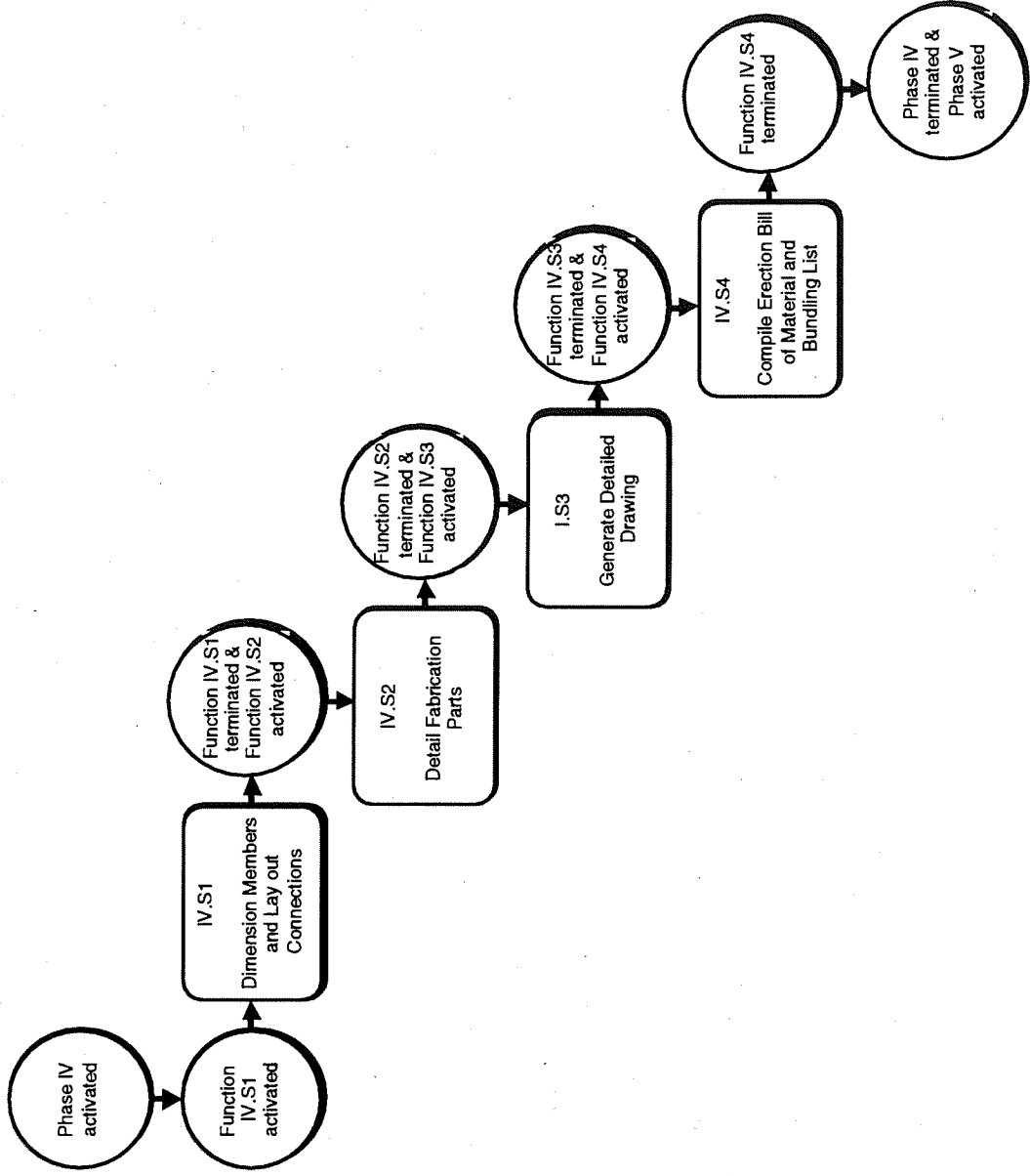


**P-DIAGRAM 3.1: Function III.S1, Structural Analysis, and Function III.S2, Member Design, of Phase III (Tower Structural Detailed Design). (See Diagram Note C)**



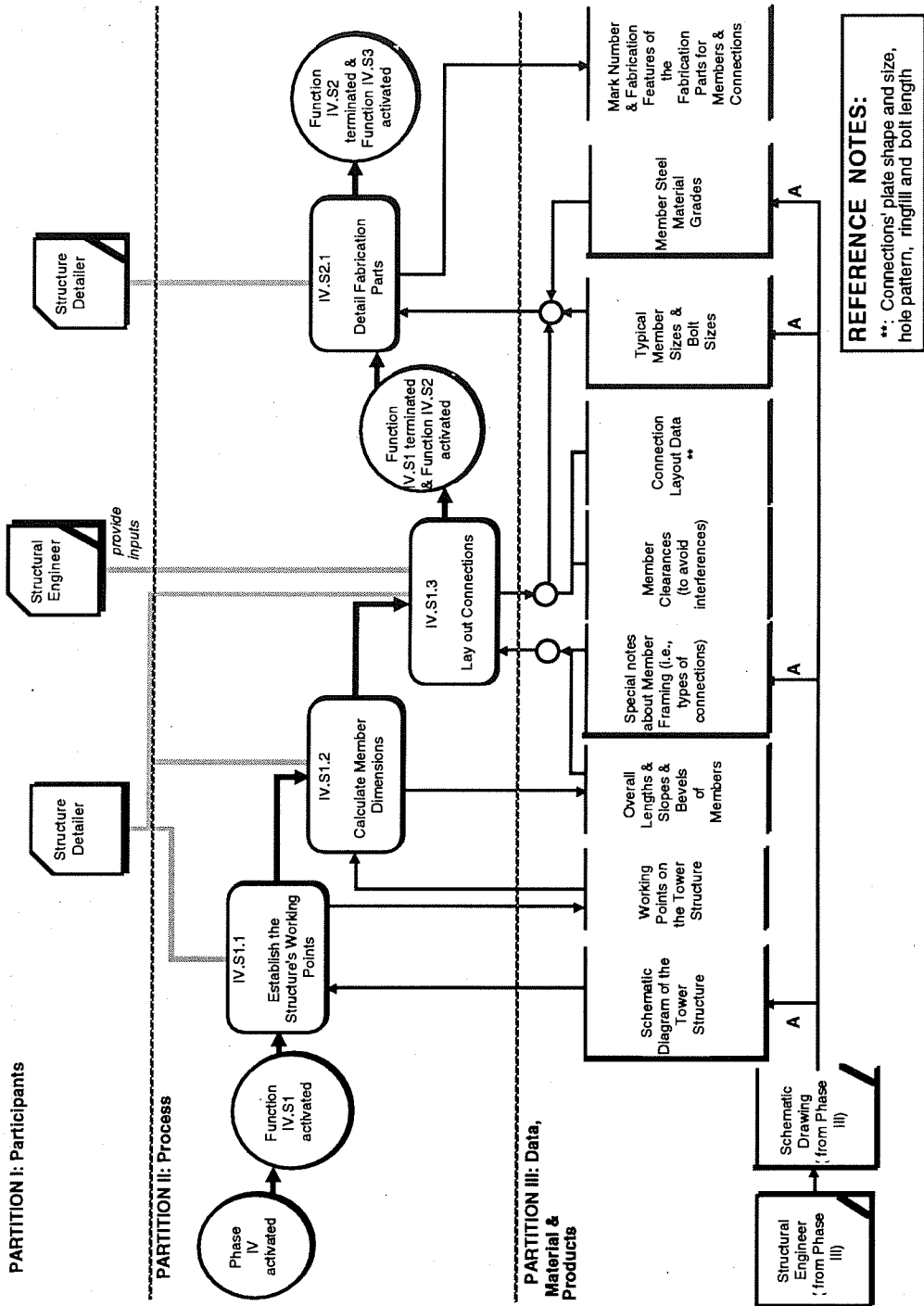
**P-DIAGRAM 3.2: Function III.S3, Design Refinement, and Function III.FD4, Foundation Design, and Function III.S5, Generating Schematic Drawing, of Phase III (Tower Structural Detailed Design). (See Diagram Note C)**

**PARTITION II: PROCESS**



**P-DIAGRAM 4: Intermediate Skeletal Graphical Functional Schema of Phase IV, Tower Construction Planning.**  
 (See Diagram Note B)

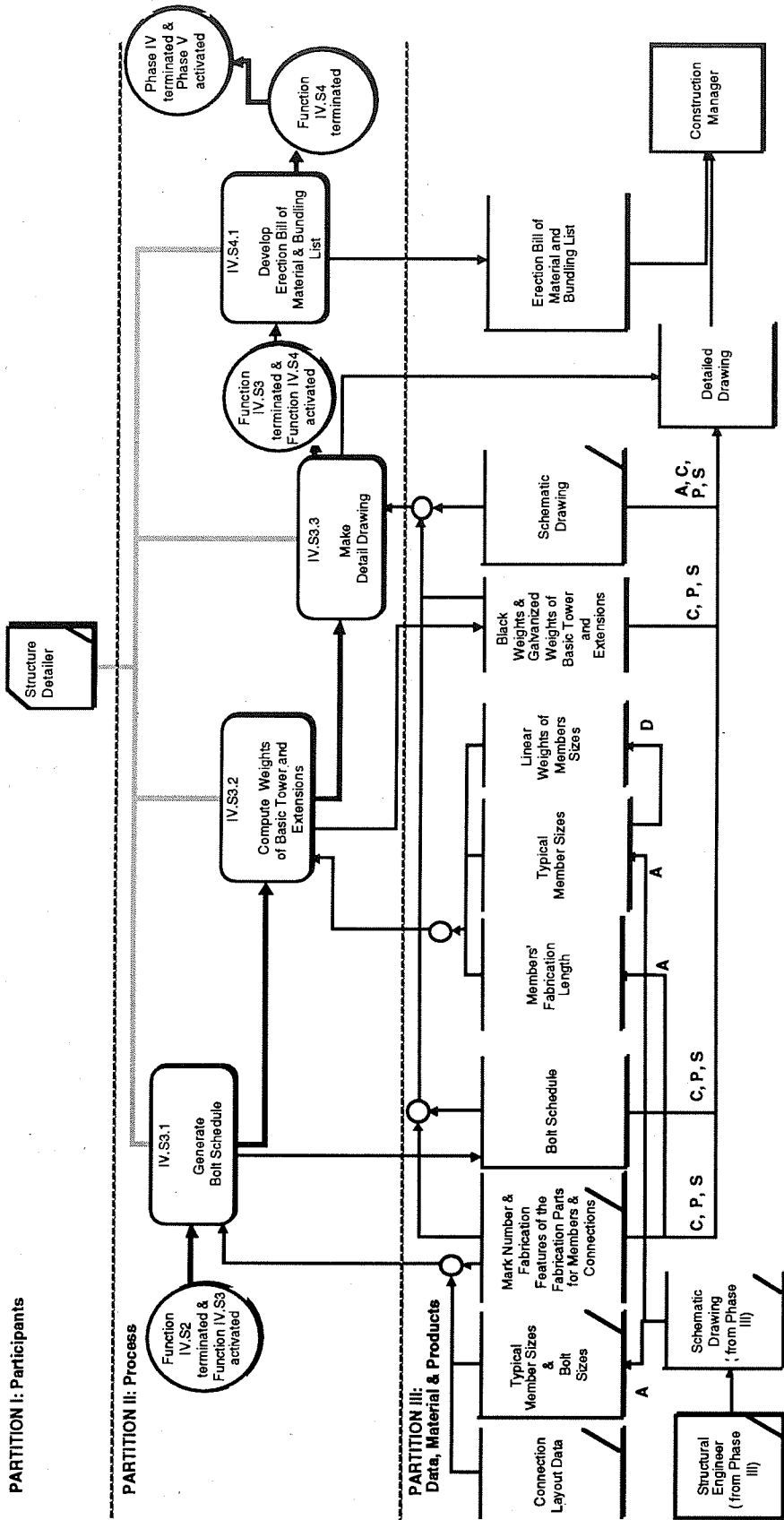
**PARTITION I: Participants**



**REFERENCE NOTES:**  
 \*\*: Connections' plate shape and size, hole pattern, ringfill and bolt length

**P-DIAGRAM 4.1: Function IV.S1, Dimensioning Members and Laying out Connections, and Function IV.S2, Detailing Fabrication Parts, of Phase IV (Tower Construction Planning). (See Diagram Note C)**

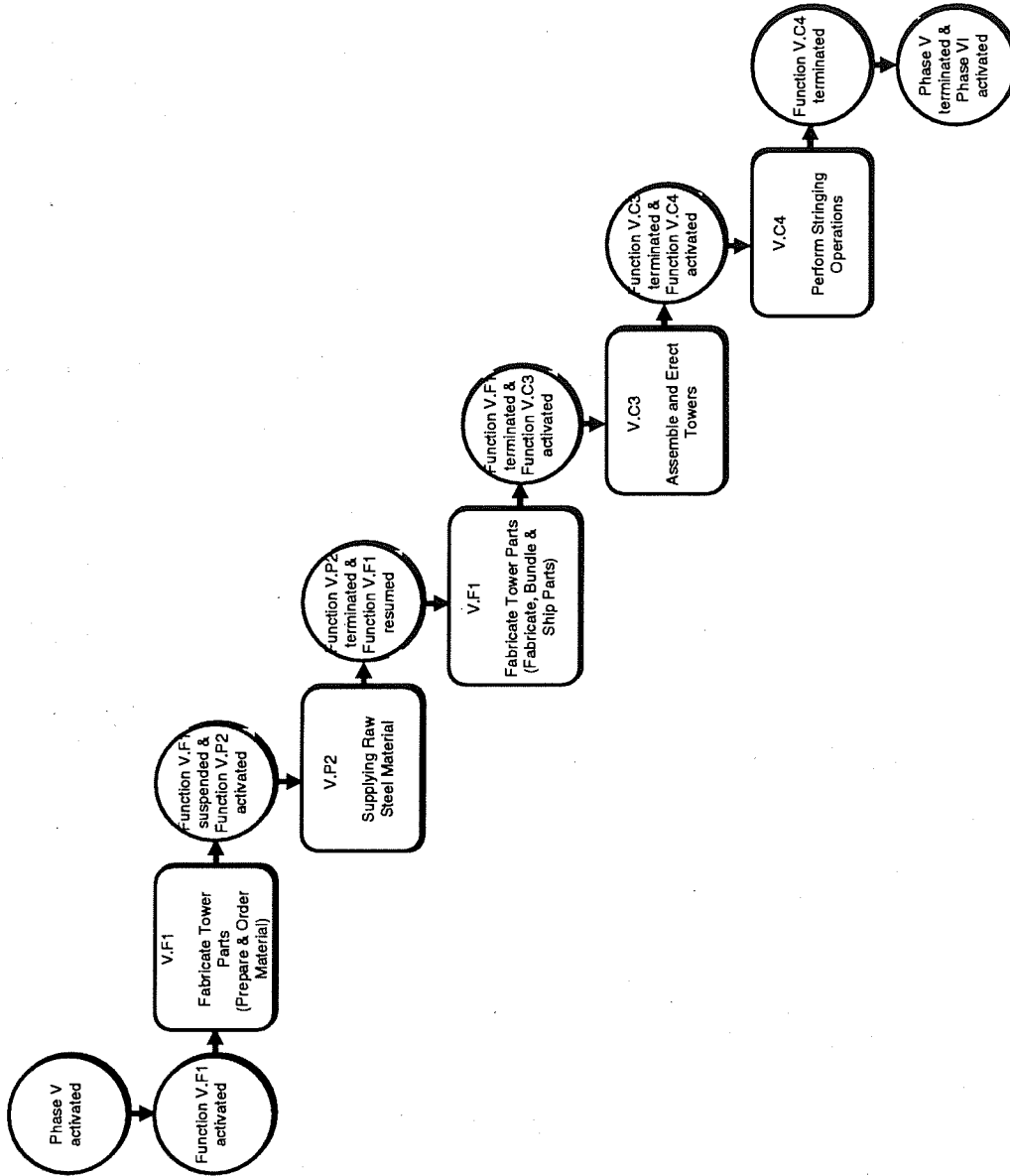
**PARTITION I: Participants**



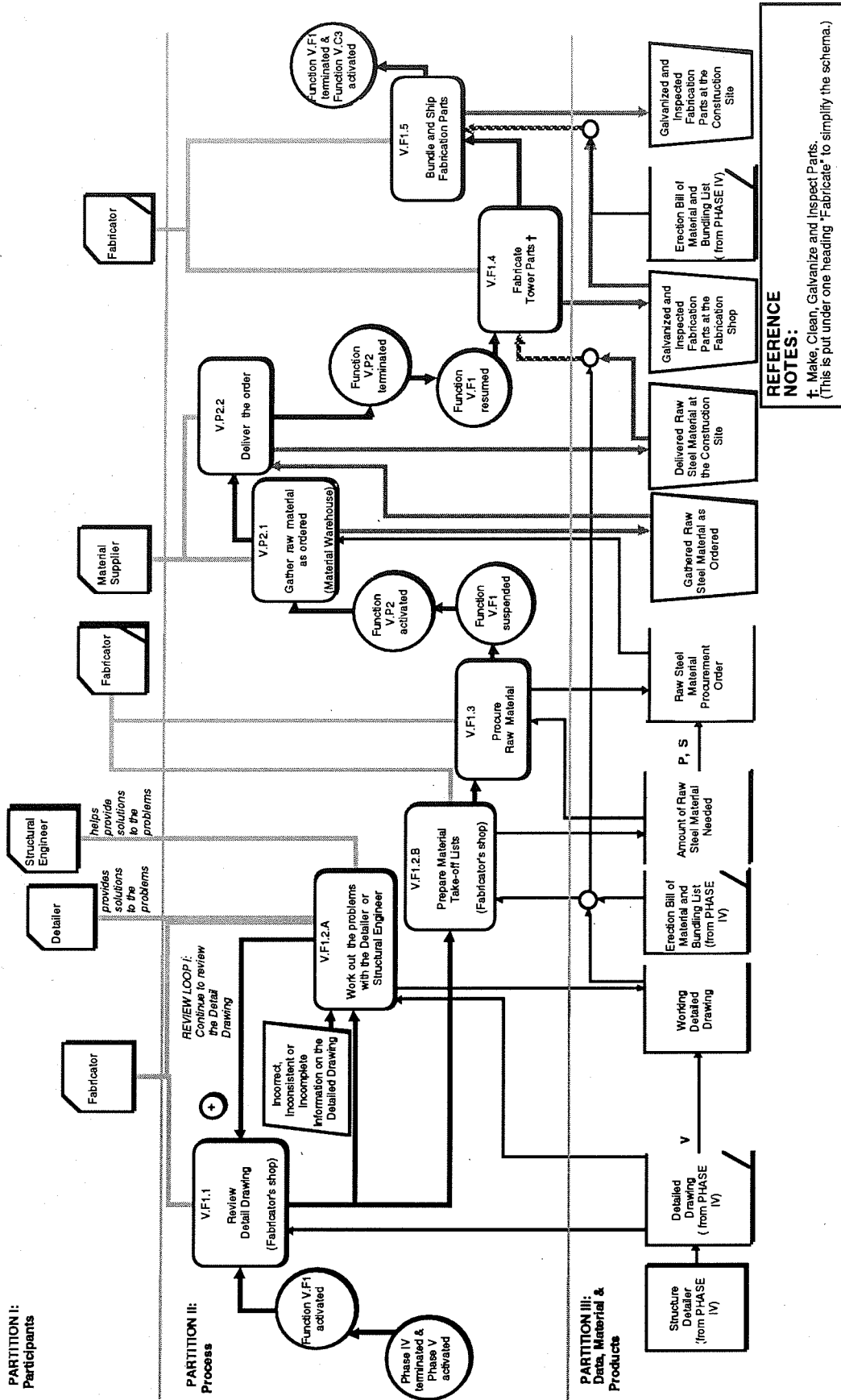
**P-DIAGRAM 4.2: Function IV.S3, Generating Detailed Drawing, and Function IV.S4, Compiling Erection Bill of Material and Bundling List, of Phase IV (Tower Construction Planning).** (See Diagram Note C)



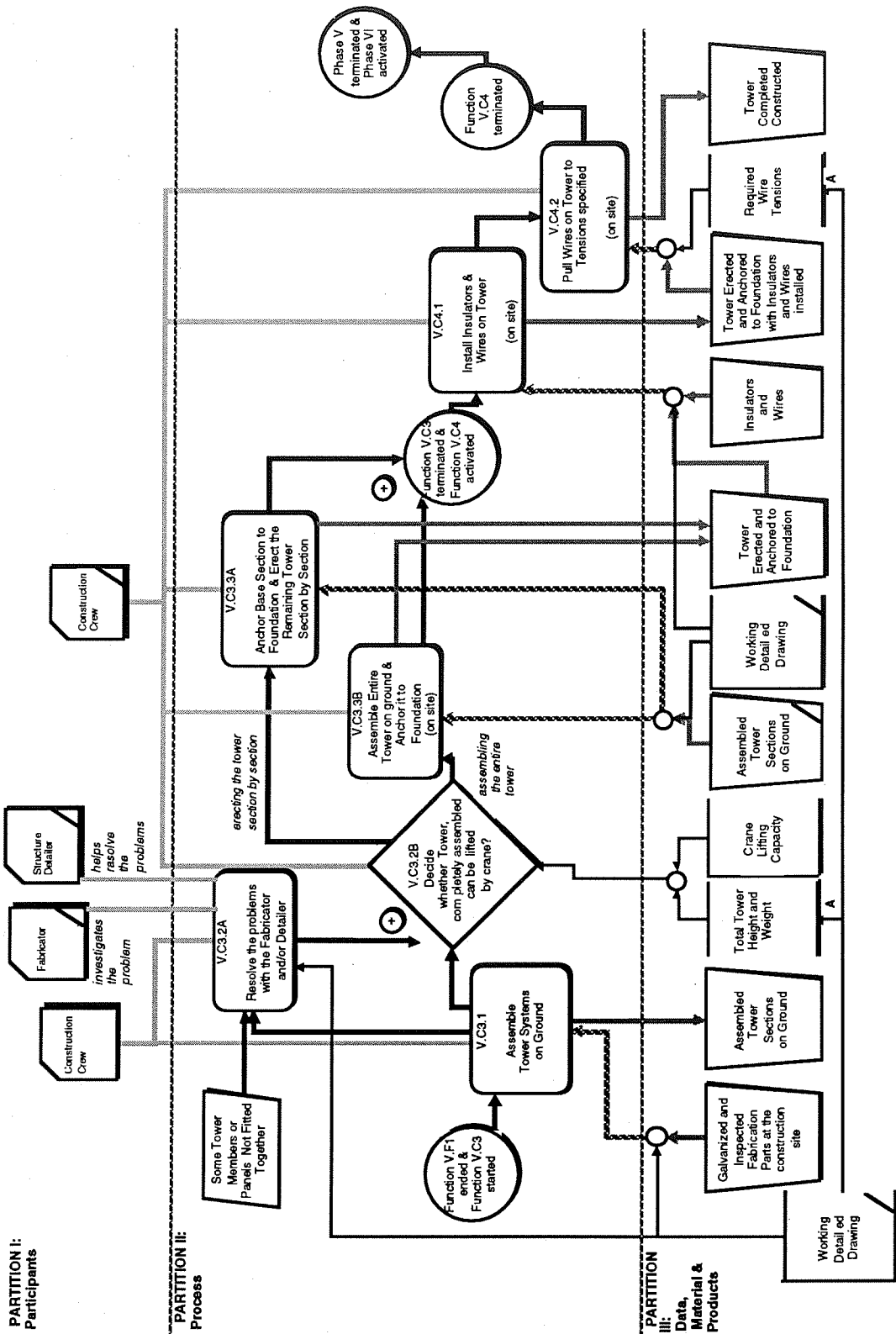
**PARTITION II: PROCESS**



**P-DIAGRAM 5: Intermediate Skeletal Graphical Functional Schema of Phase V, Tower Construction Execution. (See Diagram Note B)**



**P-DIAGRAM 5.1: Function V.F1, Parts Fabrications, and Function V.P2, Supplying Material, of Phase V (Tower Construction Execution).** (See Diagram Note C)



**P-DIAGRAM 5.2: Function V.C3, Tower Assembly and Erection, of Phase V (Tower Construction Execution).**  
 (See Diagram Note C)

# Appendix B

## **DEAL: Pseudo-Codes and A Detailed Analysis of “Transmission Tower Members”**

### **Abstract:**

This appendix shows the pseudo-codes for the operations in the procedures of DEAL-1 and DEAL-2. It also provides a detailed analysis of the “Transmission Tower Members” domain entity of transmission towers studied in this research

### **Organization:**

*B.1 Pseudo-Codes for Operations in the DEAL-1 Procedure*

*B.2 Pseudo-Codes for Operations in the DEAL-2 Procedure*

*B.3 Detailed Analysis of the “Transmission Tower Members” Domain Entity*

## **B.1 Pseudo-Codes for Operations in the DEAL-1 Procedure**

The following are pseudo-codes for the ELABORATE, ASSIGN, COHERE, and DECOMPOSE operations in the DEAL-1 procedure introduced in Section 5.3.5.1 of Chapter 5. The keywords of the pseudo-code format used here are shown in bold.

```

operation ELABORATE(vertex  $v_i$  )
// This operation is used in the first step of Procedure
DEAL-1 for each criterion I. It enables the modeler to
"elaborate" the data items of a given vertex  $v_i$ . The
elaborated data items represent the new assertions that
reflect the modeler's most current state of belief about or
understanding of the problem at hand [VanLehn 89].
begin
    // Define a temporary set  $e_j$  for the new extension of
     $v_i$ .
     $e_j$  = emptyset;

    for each  $d_k$  in EXT( $v_i$ ) do begin
        // The user inputs new data items  $d_l$  to replace the
        existing  $d_k$ . When prompting the user for  $d_l$ , also
        remind the user of the restriction of the elaboration
        task as stated in the description of the procedure.
         $e_j$  =  $e_j$  U ( $d_l$  | user-input  $d_l$ );
    end

    // Save the resulting  $e_j$  back to EXT( $v_i$ ). Function
    SET-EXT( $v_i$ ,  $s_j$ ) sets the extension of  $v_i$  to an input
    set  $s_j$  of data items.
    SET-EXT( $v_i$ ,  $e_j$ );

    // Check and enforce the safe elaboration rule.
    RUN_SAFE_ELABORATION_RULE( $v_i$ );
end

```

```
operation ASSIGN(vertex  $v_j$ , criterion I)
// This operation is used in the first step of Procedure
DEAL-1. It assigns into separate "binders" data items of a
given vertex  $v_j$  that have the same characteristic according
to the criterion I being considered. For instance, under the
access-cohesion criterion, each binder corresponds to a
unique logical path (or combination of paths) for accessing
data items of  $v_j$ .

begin
    // First, the user inputs the element I under
    consideration in the intension of the vertex. Function
    SET-INT-I( $v_j$ , Ij) sets the element I of the intension
    of  $v_j$  to the input set Ij.
    SET-INT-I( $v_j$ , user-input an_I);

    //Then, assign each data item using a function AI:
    EXT( $v_j$ ) -> INT( $v_j$ ).I. The set corresponding to this
    function is:
    AI = {( $d_j$ ,m) |  $d_j$  in EXT( $v_j$ ) & m in INT( $v_j$ ).I};

end
```

```

operation COHERE(vertex  $v_i$ , criterion I)

// This operation is used in the third step of Procedure
DEAL-1. While the previous operation assigns data items,
this operation creates "cohesive sets" of data items
contained in a given  $v_i$  whose elements cohere to the same
binder. For instance, under the access-cohesion criteria,
each cohesive set contains all data items that can be
accessed by the same unique logical access path or
combination of paths. Cohesive sets have unique properties:
They are disjoint equivalence classes of the relation
Cohere $_I$  defined below. In set theory, equivalence classes
form a partition of the underlying set and in this case, of
the entity  $e_i$  represented by  $v_i$  [McEliece 89, p. 28].

begin
    // Define an equivalence relation Cohere $_I$ : EXT( $v_i$ ) ->
    EXT( $v_i$ ) that is the composition of the corresponding
    function  $A_I$  and its inverse function: Cohere $_I = A_I \circ$ 
     $A_I^{-1}$ . The set corresponding to this relation is:
    Cohere $_I = \{(d_i, d_j) \mid (d_i, m) \text{ in } A_I \ \& \ (m, d_j) \text{ in}$ 
     $A_I^{-1} \}$ ;

    // Collect all the distinct equivalent classes  $C_I$ 
    from the equivalence relation Cohere $_I$  into a set
    called all $C_I$ .
    all $C_I = \text{emptyset}$ ;

    for each  $d_i$  in EXT( $v_i$ ) do begin
        // The symbol  $C(d_i)$  represents an equal
        equivalent class of the data item  $d_i$ .  $C(d_i)$  is
        never empty since  $d_i$  belongs to it [McEliece
        89, p. 25].
        if  $C(d_i)$  not in all $C_I$  then all $C_I = \text{allC}_I \cup$ 
         $\{C(d_i)\}$ ;
    end

    // Save the set all $C_I$  using a function SET-all $C_I(v_i$ 
    , all $C_I, I)$ . The set is retrieved in the next
    operation using a function GET-all $C_I(v_i, I)$ .
    SET-all $C_I(v_i, \text{allC}_I, I)$ ;

end

```

```

operation DECOMPOSE(vertex  $v_i$  , criterion I)
// This operation is used in the fourth step of Procedure
DEAL-1. It decomposes a given vertex  $v_i$  into new vertices
 $v_j$ , each of which corresponds to a single cohesive set.
Cohesive sets were explained in the previous operation.
begin
  for each  $C_I$  in GET-all $C_I(v_i, I)$  do begin
    // Create a new vertex  $v_j$ 
     $v_j = \text{NEW}(\text{VERTEX});$ 

    // Make  $v_j$  a child of  $v_i$  by creating a directed
    link from  $v_i$  to  $v_j$ .
    LINK( $v_i, v_j$ ).

    // Equate the extension to  $v_j$  to  $C_I$ . Function
    SET-EXT( $v_i, s_j$ ) sets the extension of  $v_i$  to an
    input set  $s_j$  of data items.
    SET-EXT( $v_j, C_I$ );

    // Copy the intension of the parent vertex  $v_i$ .
    Function, SET-INT( $v_i, t_j$ ), sets the intension
    of  $v_i$  to an input tuple  $t_j$  being ( $A_j, C_j, T_j,$ 
     $S_j, U_j$ ).
    SET-INT( $v_j, \text{INT}(v_i).I$ );

    // Appropriately modify the element I under
    consideration of the tuple in the intension of
     $v_j$ . (U: Union of sets.) Function SET-INT-I( $v_i,$ 
     $I_j$ ) sets the element A of the intension of  $v_i$ 
    to the input set  $I_j$ .
    SET-INT-I( $v_j, \mathbf{U} \{m\}$ ) where  $d_i$  in  $C_I$  and ( $d_i, m$ )
    in  $A_I$ ;

    // Prompt the user for the name of the entity
    represented by  $v_j$ .
    SET-NAME( $v_j, \text{user-input } a\_name$ );

  end
  // Check and enforce the non-loss, disjoint
  decomposition rule.
  RUN_NLD_DECOMPOSITION_RULE( $v_i$ );
end

```



## **B.2 Pseudo-Codes for Operations in the DEAL-2 Procedure**

The following are pseudo-codes for the ELABORATE-2, ASSIGN-2, COHERE-2, and DECOMPOSE-2 operations of the DEAL-2 procedure introduced in Section 5.3.6.1 of Chapter 5. These operations are similar to their counterparts in the DEAL-1 procedure. However, these operations are modified, enabling the moder to reuse the previously identified primitive entities  $ep'$ .

```

operation ELABORATE-2(vertex  $v_i$  , entities  $ep'$ )
//This operation is used in the first step of Procedure
DEAL-2.
begin
    // Define a temporary set  $e_j$  for the new extension of
     $v_i$ .
     $e_j$  = emptyset;

    for each  $d_k$  in EXT( $v_i$ ) do begin
        // MODIFICATION: If  $n$  primitive entities  $ep'$  can be
        reused in the description of the entity represented by
         $v_i$  and the data items of  $ep'$  can replace  $d_k$ , then
        use  $ep'$ .
        if ( $n$  REUSABLE  $ep'$ ) then  $e_j$  =  $e_j \cup_{k=1,n} ep'$ ;
        // Else the user inputs the new data items  $d_l$  to
        replace the existing  $d_k$ . When prompting the user for
         $d_l$ , also remind the user of the restriction of the
        elaboration task as stated in the description of the
        procedure.
        else  $e_j$  =  $e_j \cup \{d_l \mid \text{user-input } d_l\}$ ;
    end

    // Save the result back to EXT( $v_i$ ). Function SET-
    EXT( $v_i, s_j$ ) sets the extension of  $v_i$  to an input set
     $s_j$  of data items.
    SET-EXT( $v_i, e_j$ );

    // Check and enforce the safe elaboration rule.
    RUN_SAFE_ELABORATION_RULE( $v_i$ );
end

```

```

operation ASSIGN-2(vertex  $v_j$ , criterion I, entities  $ep'$ )
//This operation is used in Step I.1 of Procedure DEAL-2.
begin
    // MODIFICATION: If n primitive entities  $ep'$  were
    reused in the previous operation ELABORATE-2, then the
    data items that were elaborated from  $ep'$  will not be
    assigned in this operation. Let  $eelab$  is the set of
    data items that were elaborated from the primitive
    entities  $ep'$ .
        if (REUSABLE  $ep'$ ) then  $eelab = \bigcup_{k=1,n} ep'$ ;
        else  $eelab = \text{empty-set}$ ;

    // First, the user inputs the element I under
    consideration in the intension of the vertex. Function
    SET-INT-I( $v_j$ ,  $I_j$ ) sets the element I of the intension
    of  $v_j$  to the input set  $I_j$ .
    SET-INT-I( $v_j$ , user-input  $an_I$ );

    //Then, assign each data item using a function  $A_I$ :
    EXT( $v_j$ )  $\rightarrow$  INT( $v_j$ ).I. The set corresponding to this
    function is:
     $A_I = \{(d_{j,m}) \mid d_j \text{ in EXT}(v_j) \ \& \ m \text{ in INT}(v_j).I\}$ ;

end

```

```

operation COHERE-2(vertex  $v_j$ , criterion I, entities  $ep'$ )
begin
    // Define an equivalence relation  $Cohere_I: EXT(v_j) \rightarrow$ 
     $EXT(v_j)$  that is the composition of the corresponding
    function  $A_I$  and its inverse function:  $Cohere_I = A_I \circ$ 
     $A_I^{-1}$ . The set corresponding to this relation is:
     $Cohere_I = \{(d_i, d_j) \mid (d_i, m) \text{ in } A_I \ \& \ (m, d_j) \text{ in}$ 
     $A_I^{-1} \}$ ;

    // Collect all the distinct equivalent classes  $C_I$ 
    from the equivalence relation  $Cohere_I$  into a set
    called  $allC_I$ .

    // MODIFICATION: If n primitive entities  $ep'$  were
    reused in the previous operations ELABORATE-2 and
    I-ASSIGN-2, then  $ep'$  serve here as the initial
    cohesive sets in the set  $allC_I$ .

    if (n REUSABLE  $ep'$ ) then  $allC_I(v_j) = \bigcup_{k=1, n} ep'$ ;
    else  $allC_I(v_j) = \text{emptyset}$ ;

    for each  $d_j$  in ( $EXT(v_j) - e_{elab}$ ) do begin
        // The symbol  $C(d_j)$  represents an equal
        equivalent class of the data item  $d_j$ .  $C(d_j)$  is
        never empty since  $d_j$  belongs to it [McEliece
        89, p. 25].

        if  $C(d_j)$  not in  $allC_I$  then  $allC_I = allC_I \cup$ 
         $\{C(d_j)\}$ ;

    end

    // Save the set  $allC_I$  using a function  $SET-allC_I(v_j$ 
    ,  $allC_I, I)$ . The set is retrieved in the next
    operation using a function  $GET-allC_I(v_j, I)$ .

     $SET-allC_I(v_j, allC_I, I)$ ;

end

```

```

operation DECOMPOSE-2(vertex  $v_i$ , criterion I, entities  $ep'$ )
begin
    // This first part is similar to the DECOMPOSE
    operation in DEAL-1.
    for each  $C_I$  in GET-all $C_I(v_i, I)$  do begin
        // Create a new vertex  $v_j$  i.
         $v_j = \text{NEW}(\text{VERTEX});$ 

        // Make  $v_j$  a child of  $v_i$  by creating a directed
        link from  $v_i$  to  $v_j$ .
        LINK( $v_i, v_j$ ).

        // Equate the extension to  $v_j$  to  $C_I$ . Function
        SET-EXT( $v_i, s_j$ ) sets the extension of  $v_i$  to an
        input set  $s_j$  of data items.
        SET-EXT( $v_j, C_I$ );

        // Copy the intension of the parent vertex  $v_i$ .
        Function, SET-INT( $v_i, t_j$ ), sets the intension of
         $v_i$  to an input tuple  $t_j$  ( $A_j, C_j, T_j, S_j, U_j$ ).
        SET-INT( $v_j, \text{INT}(v_i).I$ );

        // MODIFICATION: The element  $m$  in  $\text{INT}(v_i).I$  has
        not been defined if  $C_I$  comes from one of the
        primitive entities  $ep'$  that were reused. Prompt
        the user for  $m$  and modify the element  $I$  under
        consideration in the intension of  $v_j$ . Function
        SET-INT-I( $v_i, I_j$ ) sets the element  $A$  of the
        intension of  $v_i$  to the input set  $I_j$ .
        if (n REUSABLE  $ep'$ ) then begin
            SET-INT-I( $v_j, \text{user-input } \{m\}$ );

            // Set the name of  $v_j$  to the same name as
             $ep'$ .
            SET-NAME( $v_j, ep\_name$ );

            // Then declared  $v_j$  as terminated vertex.
            TERMINATE( $v_j$ );
        end
        else begin
            // Appropriately modify the element  $I$ 
            under consideration of the tuple in the
            intension of  $v_j$ . ( $U$ : Union of sets.)
            SET-INT-I( $v_j, U \{m^*\}$ ) where  $d_i$  in
             $C_I$  and ( $d_i, m^*$ ) in  $A_I$ ;

            // Prompt the user for the name of the
            entity represented by  $v_j$ .
            SET-NAME( $v_j, \text{user-input } a\_name$ );
        end
    end
    // Check and enforce the non-loss, disjoint
    decomposition rule.
    RUN_NLD_DECOMPOSITION_RULE( $v_i$ );
end

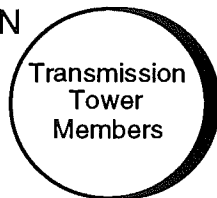
```

### B.3 Detailed Analysis of the “Transmission Tower Members” Domain Entity

This section shows a detailed analysis of a domain entity called “Transmission Tower Members” using DEAL-1. In this analysis, the domain entity is initially described by one data item, tower\_data. Five successive steps were carried out to evaluate the entities’ cohesion. In those steps, the shaded vertices are the non-cohesive one that get decomposed. Data items whose name ends in “\_data” will be elaborated in a later step.

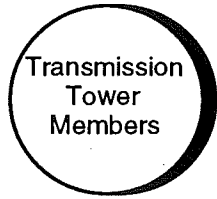
#### STEP 1: ACCESS-COHESION

• ELABORATE: Replace tower\_data with two more specific data items.



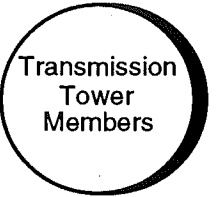
{detailed\_fabrication\_data,  
design\_data  
analysis\_data}

• ASSIGN: Determine the logical access paths & assign each data item to the proper paths.



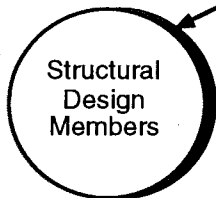
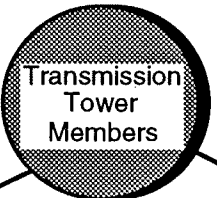
```
(
  { [Project-Folder].Member_ID,
    [Detailing-Documents].Mark_No,
    [Analysis-Documents].Element_No }
  { detailed_fabrication_data,
    design_data,
    analysis_data }
```

• COHERE: Find the cohesive sets. (In this case, there are two.)

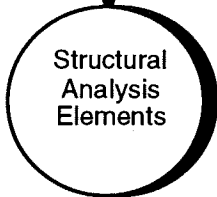


1. { detailed\_fabrication\_data }
2. { design\_data }
3. { analysis data }

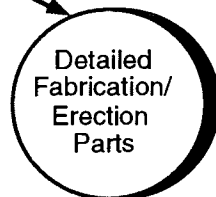
• DECOMPOSE: Decompose the non-cohesive vertex into child vertices, each corresponding to a single cohesive set.



(( [Project-Folder].Member\_ID )  
{design\_data})

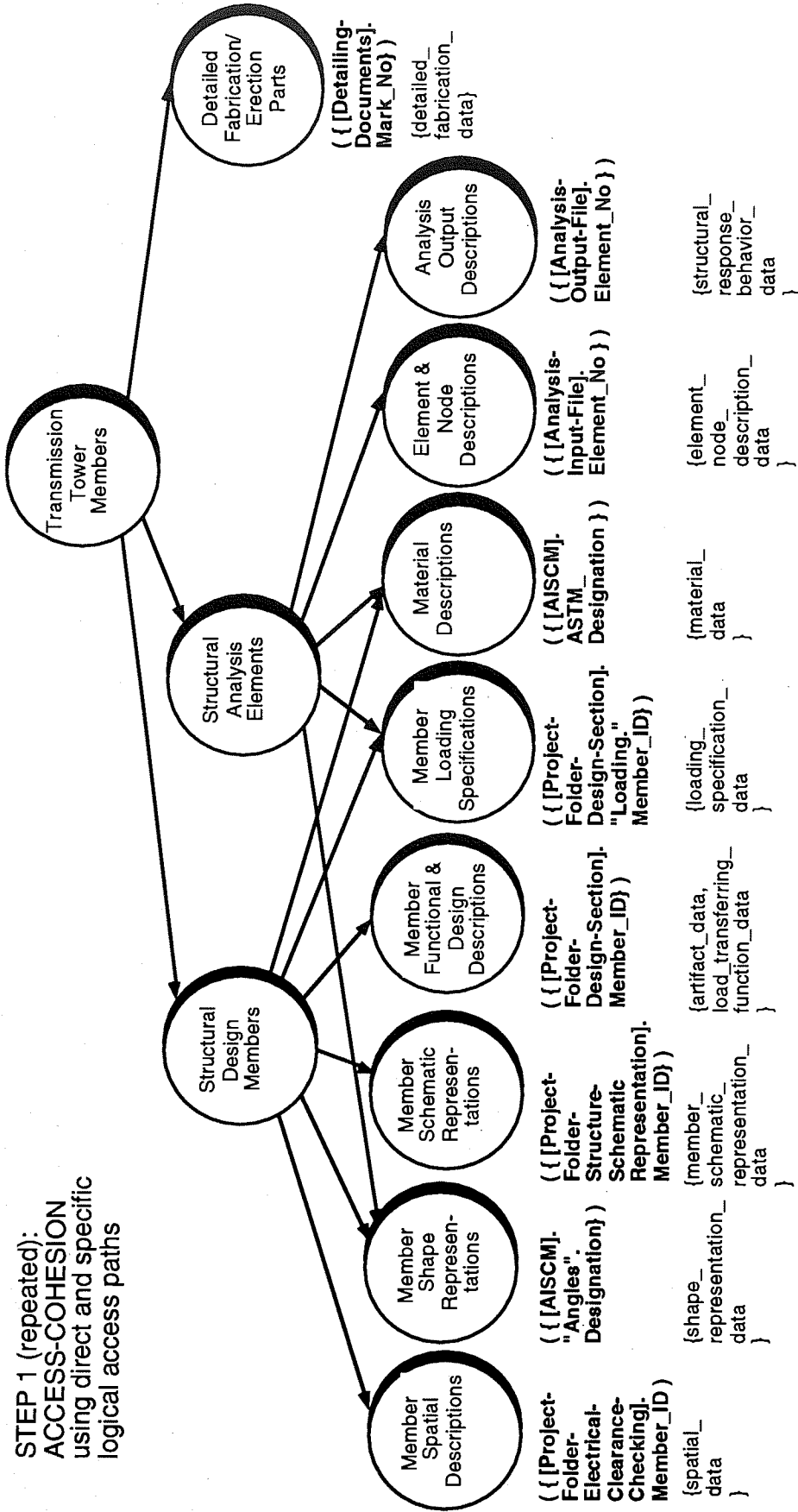


(( [Analysis-Documents].Element\_No )  
{analysis\_data})

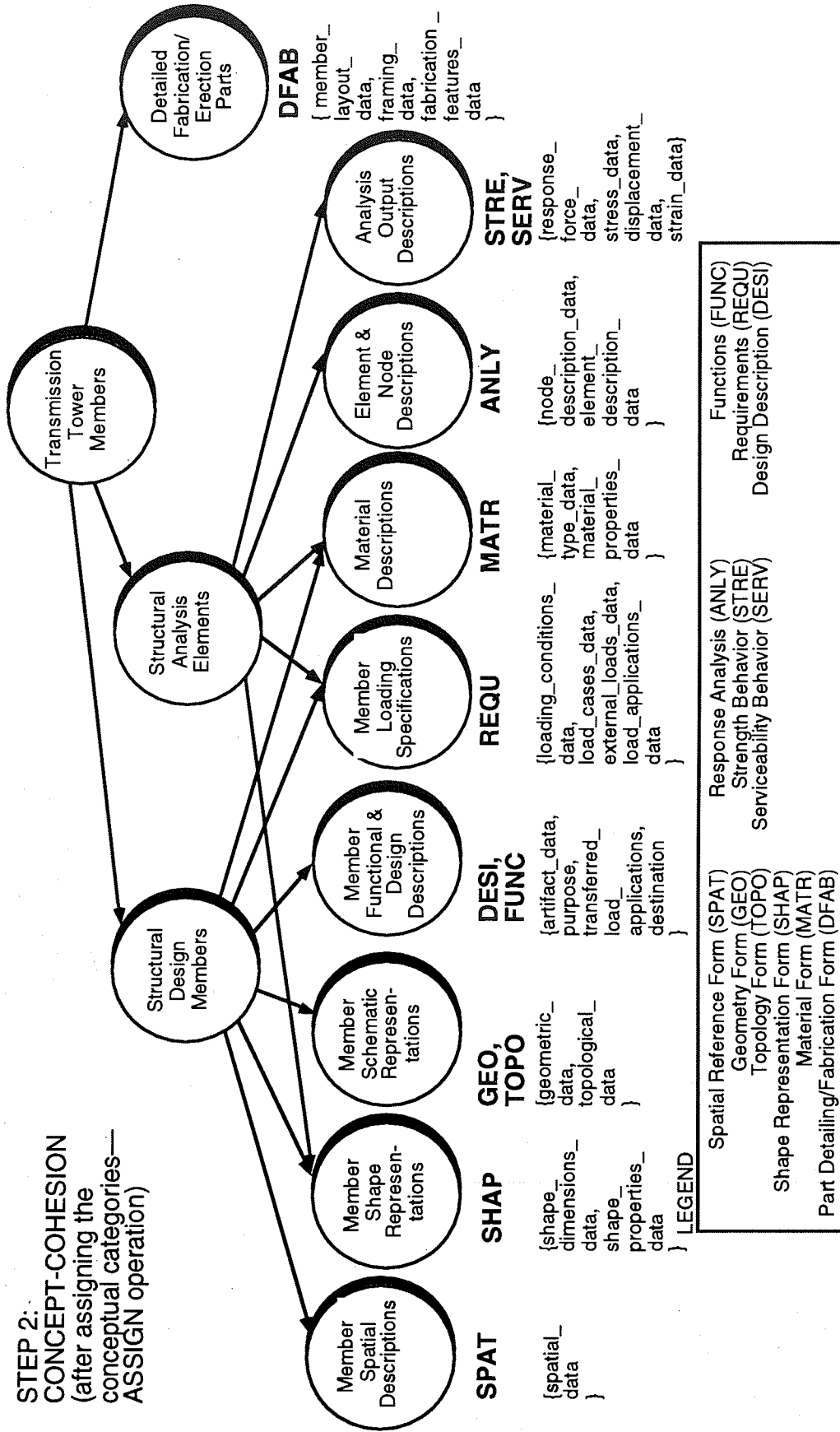


(( [Detailing-Documents].Mark\_No )  
{detailed\_fabrication\_data})

STEP 1 (repeated):  
ACCESS-COHESION  
using direct and specific  
logical access paths

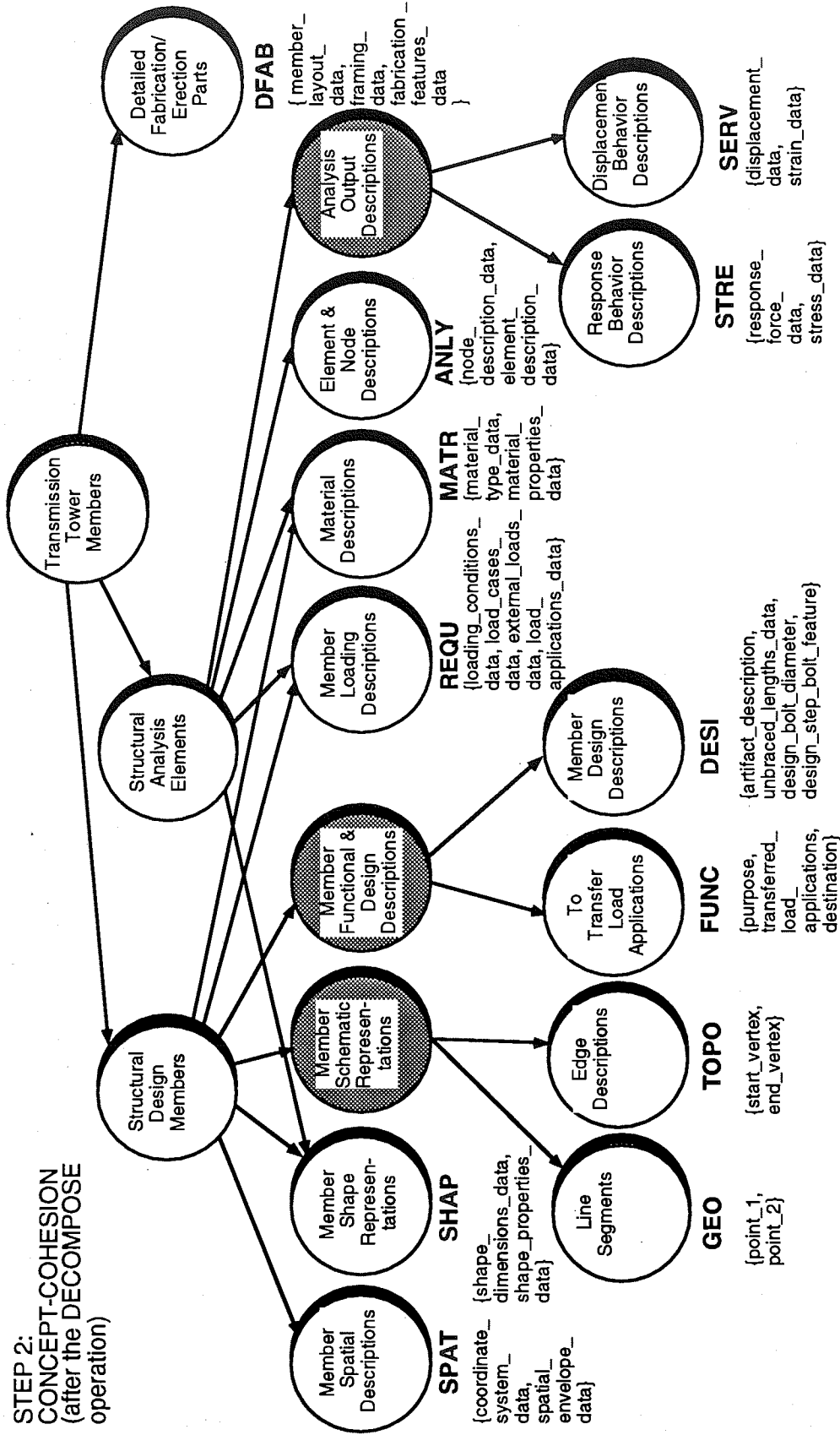


**STEP 2:  
CONCEPT-COHESION  
(after assigning the  
conceptual categories—  
ASSIGN operation)**



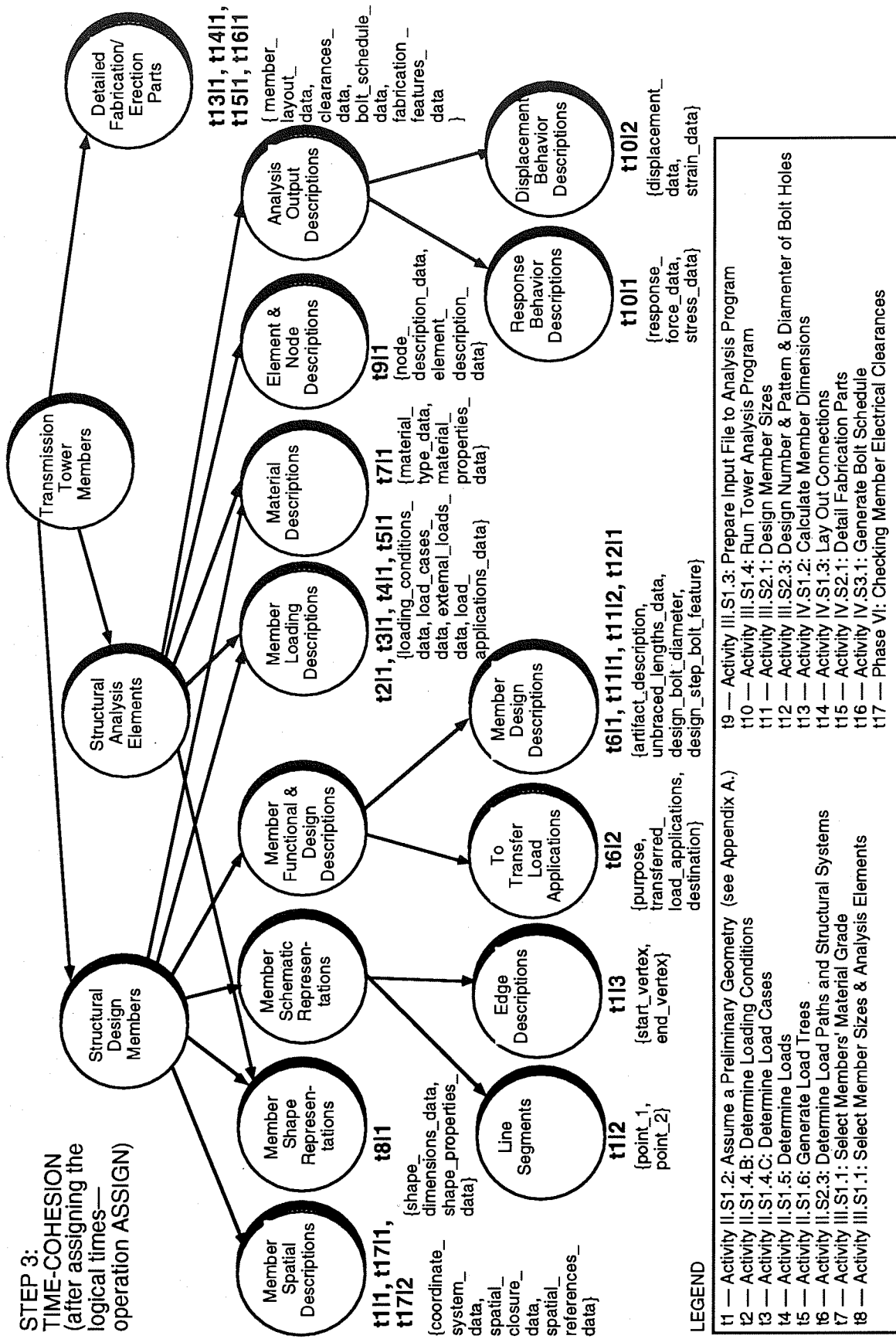
Spatial Reference Form (SPAT)	Response Analysis (ANLY)	Functions (FUNC)
Geometry Form (GEO)	Strength Behavior (STRE)	Requirements (REQU)
Topology Form (TOPO)	Serviceability Behavior (SERV)	Design Description (DESI)
Shape Representation Form (SHAP)		
Material Form (MATR)		
Part Detailing/Fabrication Form (DFAB)		

**STEP 2:  
CONCEPT-COHESION  
(after the DECOMPOSE  
operation)**





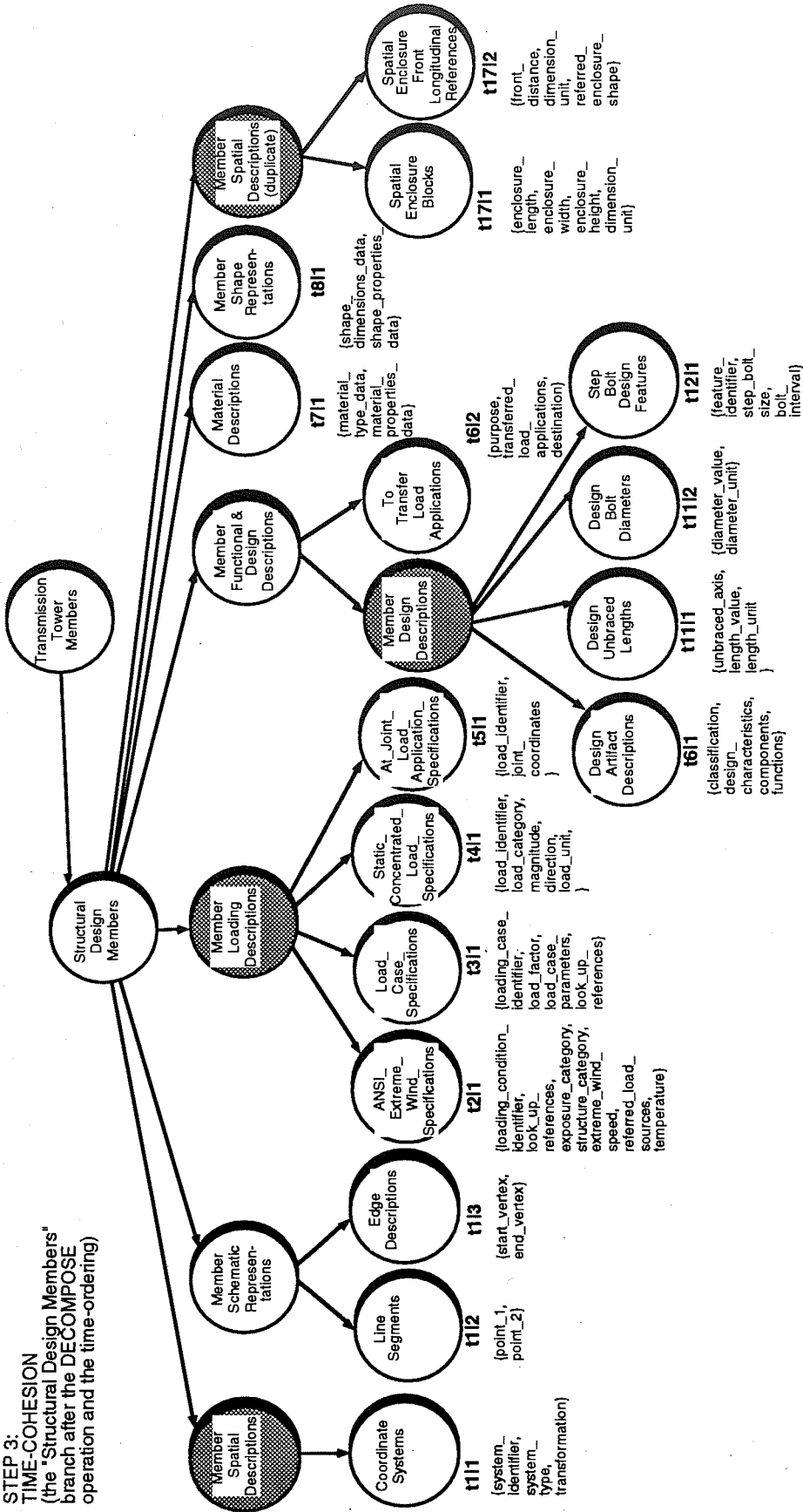
**STEP 3:  
TIME-COHESION  
(after assigning the  
logical times—  
operation ASSIGN)**



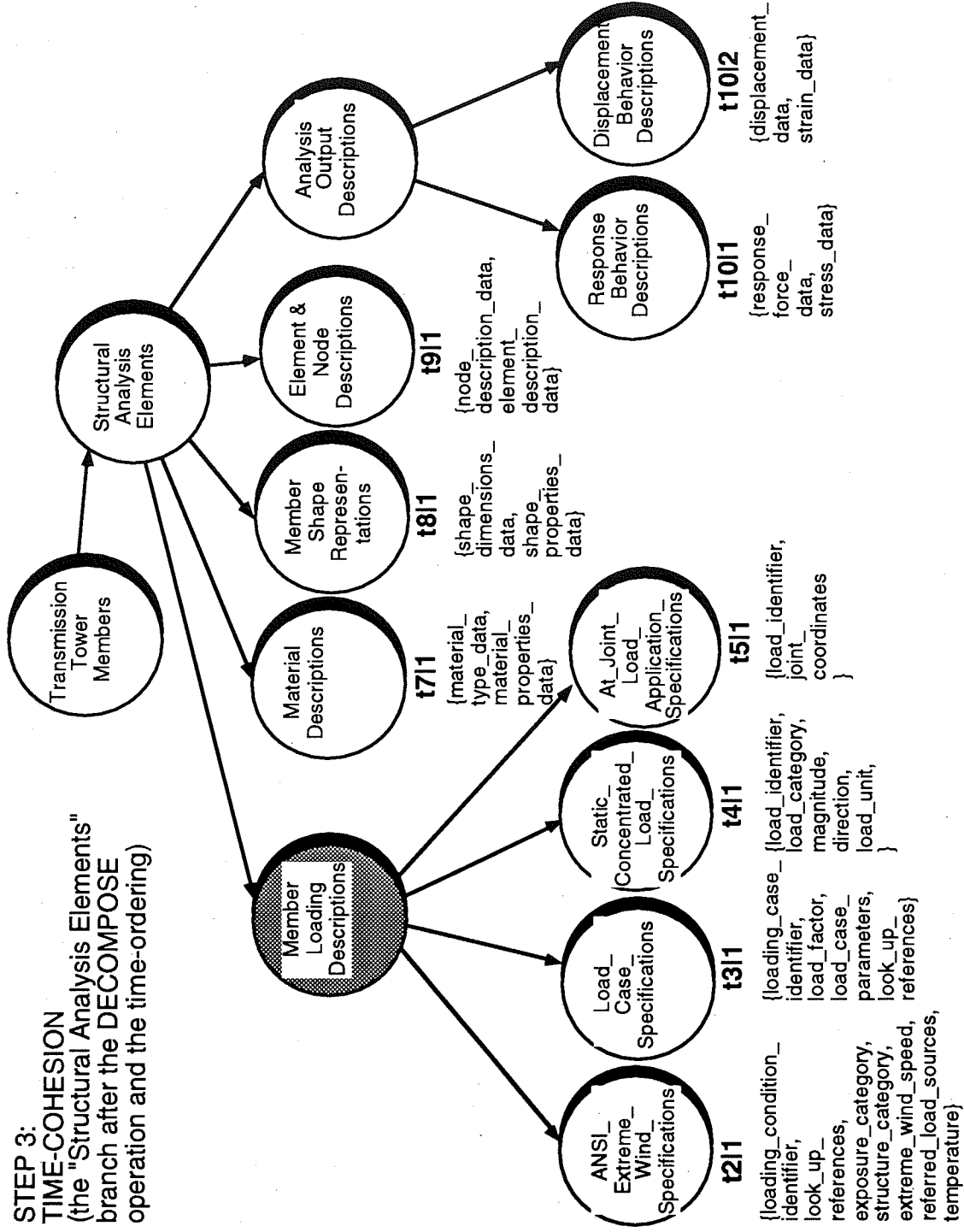
**LEGEND**

- t1 — Activity II.S1.2: Assume a Preliminary Geometry (see Appendix A.)
- t2 — Activity II.S1.4.B: Determine Loading Conditions
- t3 — Activity II.S1.4.C: Determine Load Cases
- t4 — Activity II.S1.5: Determine Loads
- t5 — Activity II.S1.6: Generate Load Trees
- t6 — Activity II.S2.3: Determine Load Paths and Structural Systems
- t7 — Activity III.S1.1: Select Members' Material Grade
- t8 — Activity III.S1.1: Select Member Sizes & Analysis Elements
- t9 — Activity III.S1.3: Prepare Input File to Analysis Program
- t10 — Activity III.S1.4: Run Tower Analysis Program
- t11 — Activity III.S2.1: Design Member Sizes
- t12 — Activity III.S2.3: Design Number & Pattern & Diameter of Bolt Holes
- t13 — Activity IV.S1.2: Calculate Member Dimensions
- t14 — Activity IV.S1.3: Lay Out Connections
- t15 — Activity IV.S2.1: Detail Fabrication Parts
- t16 — Activity IV.S3.1: Generate Bolt Schedule
- t17 — Phase VI: Checking Member Electrical Clearances
- t111, t112, t113, t114, t115, t116, t117, t121, t122, t123, t124, t125, t131, t132, t133, t134, t135, t141, t142, t143, t144, t145, t151, t152, t153, t154, t155, t161, t162, t163, t164, t165, t166, t167, t171, t172, t173, t174, t175, t176, t177

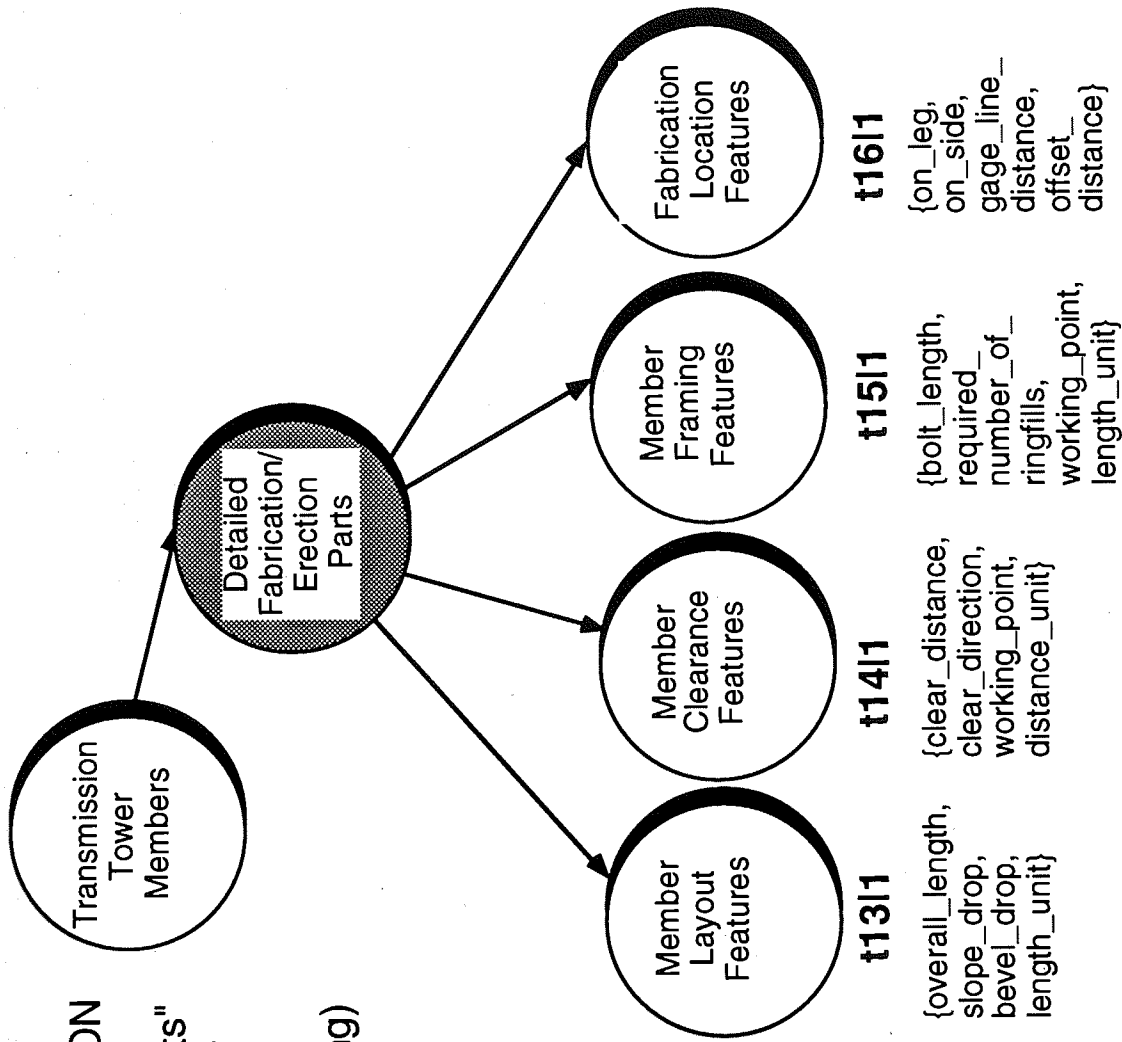
**STEP 3:  
TIME-COHESION**  
 (the "Structural Design Members"  
 branch after the DECOMPOSE  
 operation and the time-ordering)



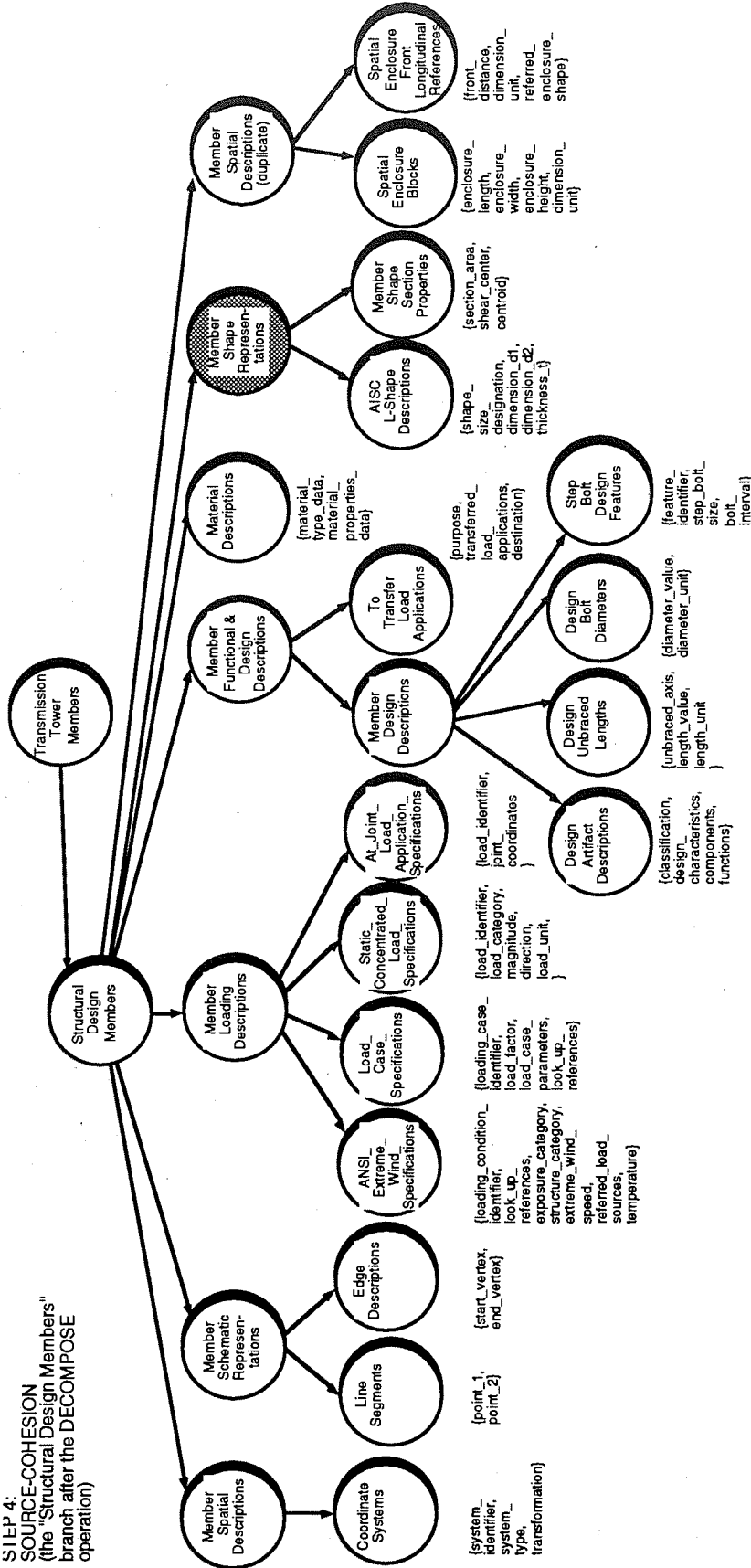
**STEP 3:  
TIME-COHESION**  
 (the "Structural Analysis Elements"  
 branch after the DECOMPOSE  
 operation and the time-ordering)



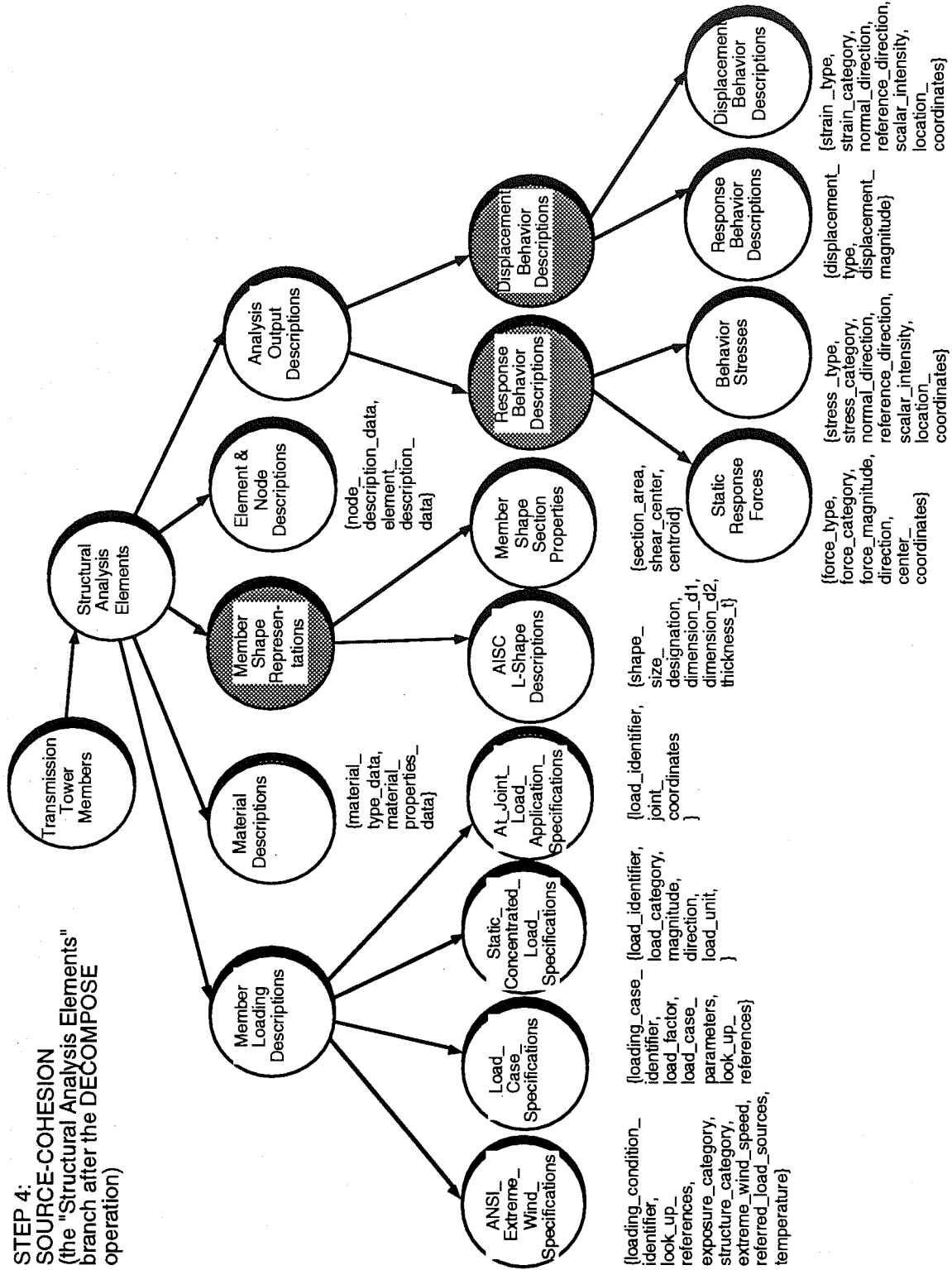
**STEP 3:  
TIME-COHESION**  
(the "Detailed  
Fabrication Parts"  
branch after the  
DECOMPOSE  
operation and  
the time-ordering)



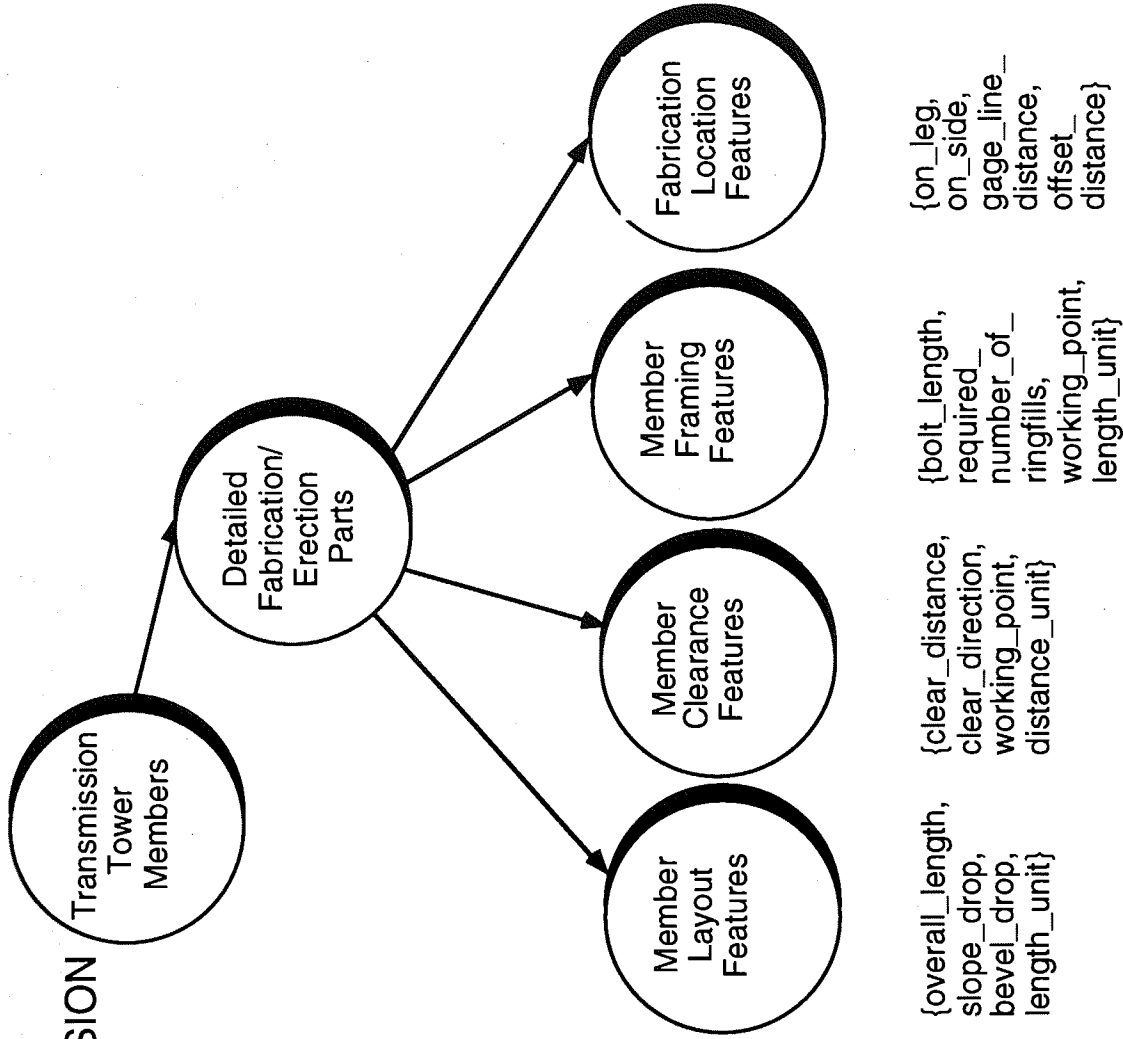
**STEP 4:  
SOURCE-COHESION**  
(the "Structural Design Members"  
branch after the DECOMPOSE  
operation)



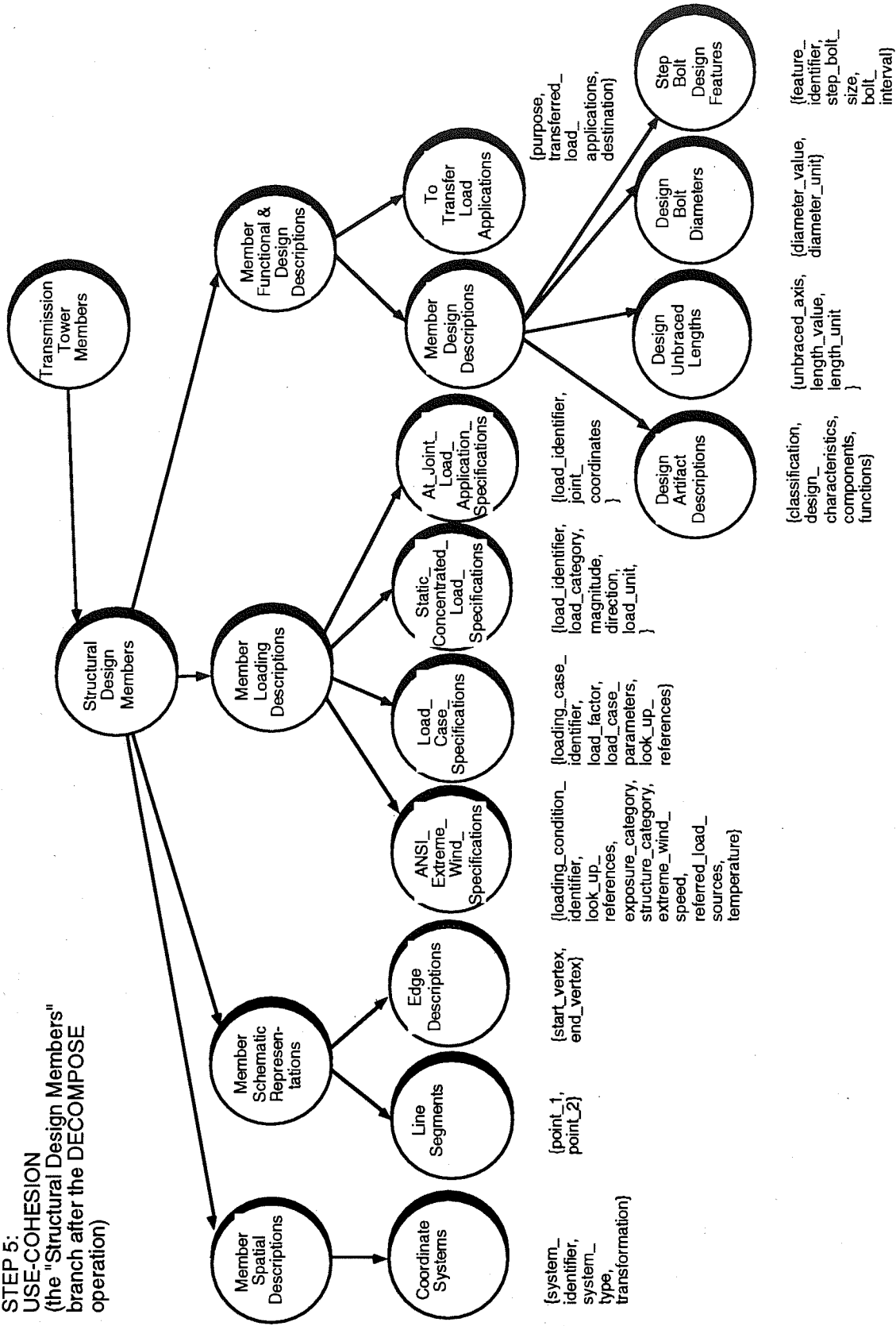
**STEP 4:  
SOURCE-COHESION  
SOURCE-Analysis Elements"  
branch after the DECOMPOSE  
operation)**



**STEP 4:  
SOURCE-COHESION  
(the "Detailed  
Fabrication Parts"  
branch after the  
DECOMPOSE  
operation)**

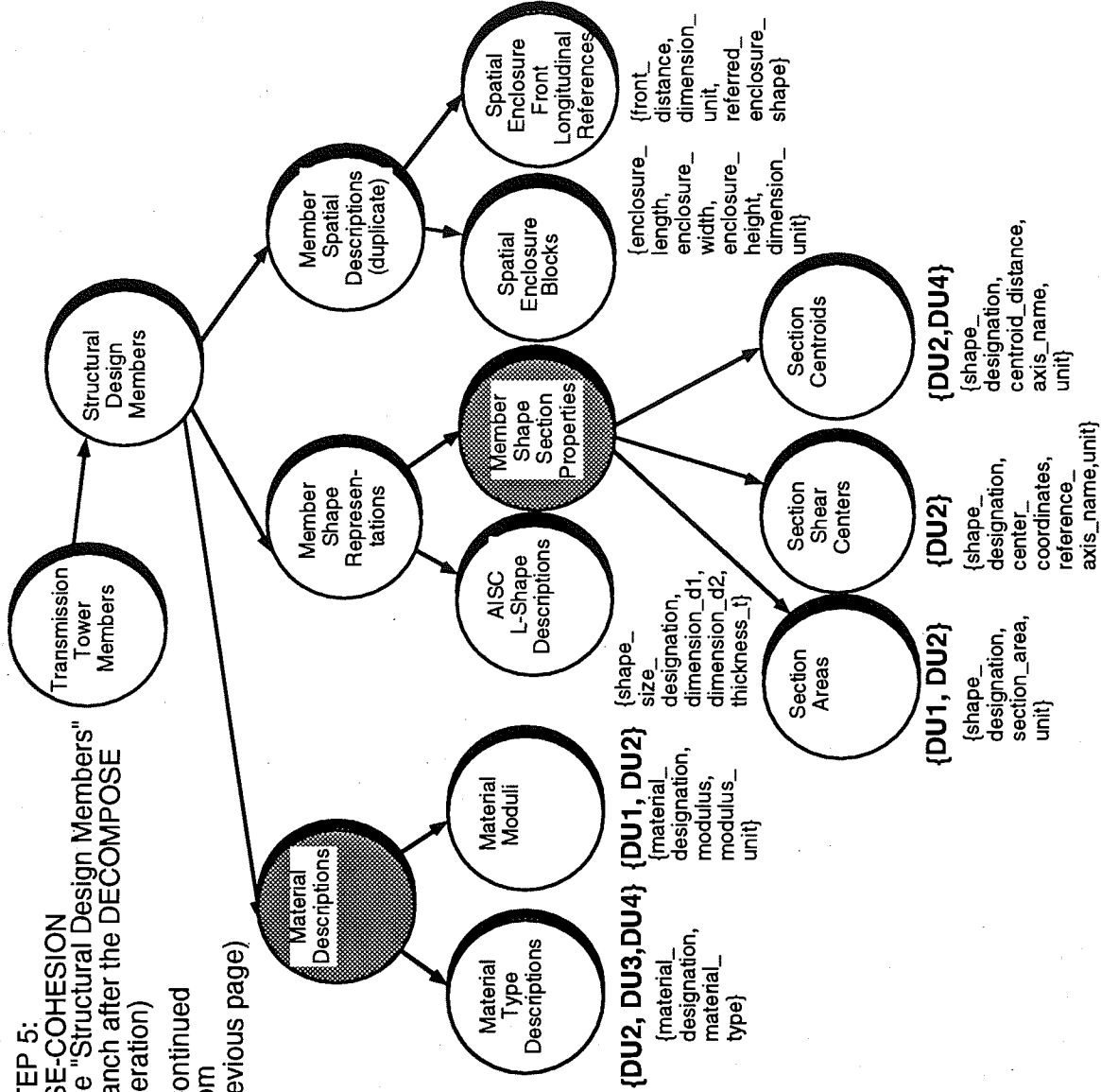


**STEP 5:  
USE-COHESION**  
(the "Structural Design Members"  
branch after the DECOMPOSE  
operation)

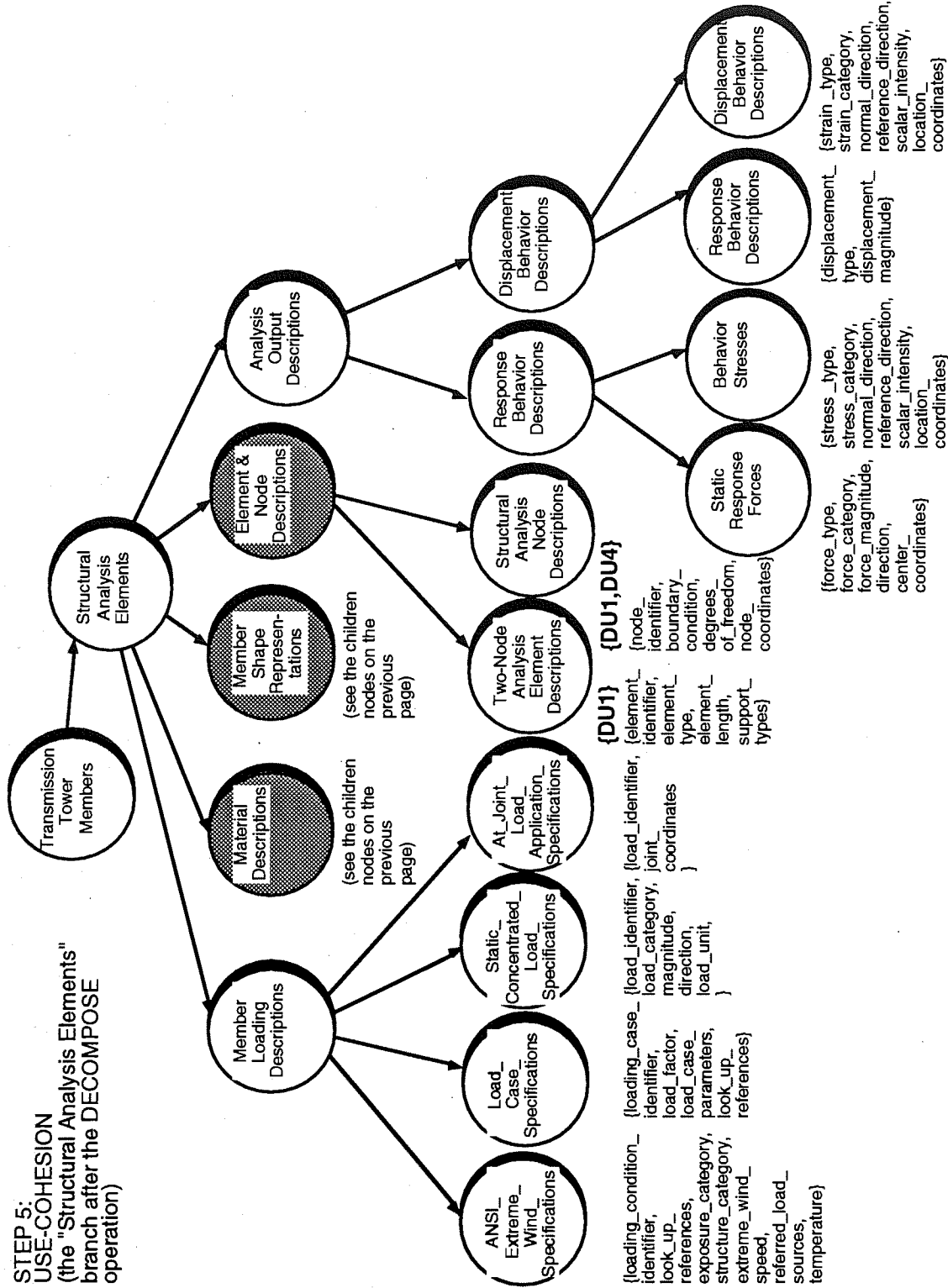




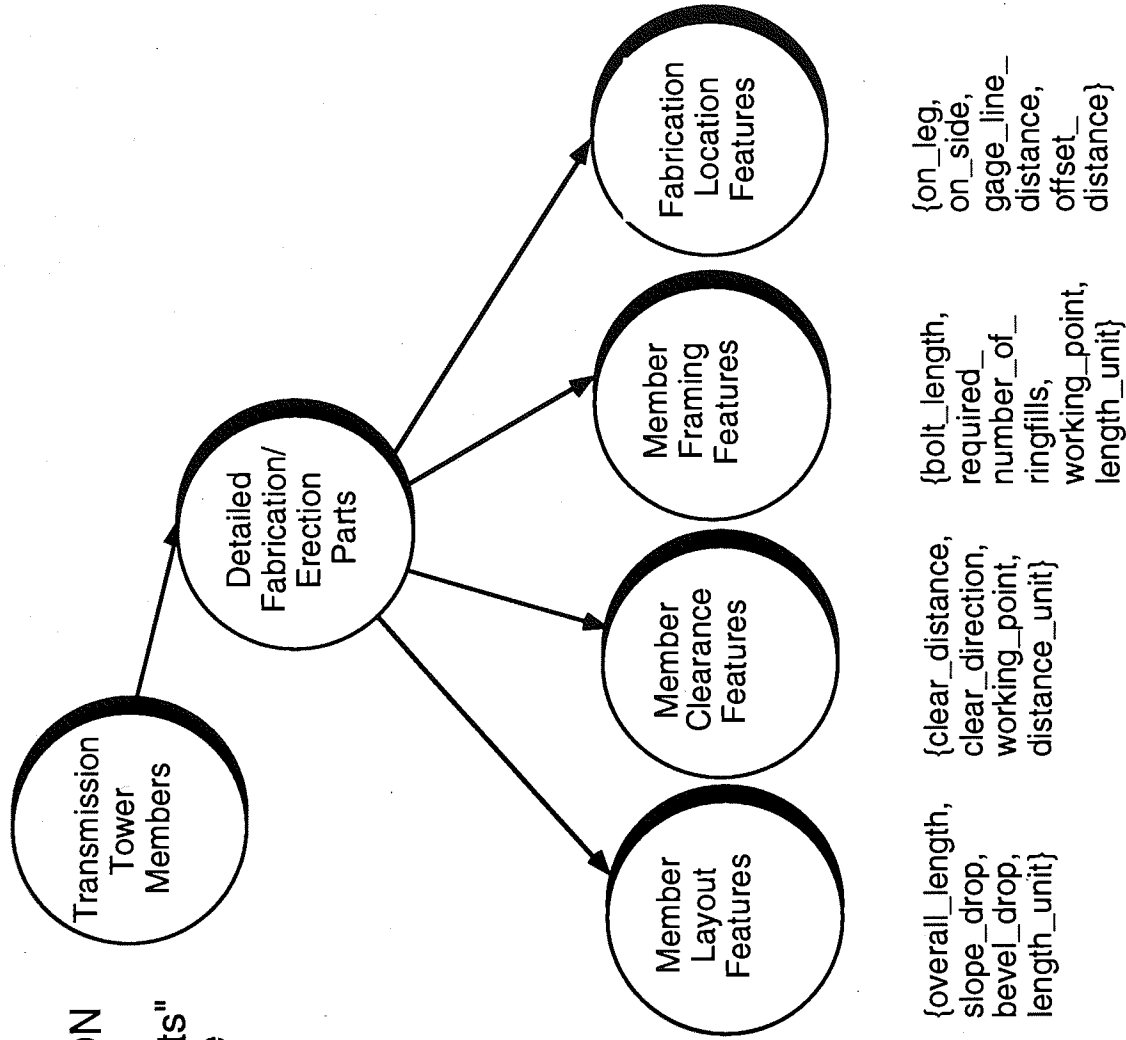
STEP 5:  
 USE-COHESION  
 (the "Structural Design Members"  
 branch after the DECOMPOSE  
 operation)  
 (Continued  
 from  
 previous page)



**STEP 5:  
USE-COHESION  
(the "Structural Analysis Elements"  
branch after the DECOMPOSE  
operation)**



**STEP 5:  
USE-COHESION  
(the "Detailed  
Fabrication Parts"  
branch after the  
DECOMPOSE  
operation)**



# Appendix C

---

## Rules and Guidelines of the P-C Data Modeling Method

### **Abstract:**

This appendix contains the full documentation of all rules and guidelines of the P-C Data Modeling Method. These rules and guidelines are divided into two major groups: (1) for the refinement of entities and (2) those for the design of object classes (including their attributes and methods) and class hierarchies. Whenever possible, documentation includes the following items: label and name (e.g., “R-REFI.1” and “Complete Entity Descriptions”), statement, explanation, example(s), exception(s), counterexample(s), discussion (i.e., applicability, advantages, tradeoffs, and other comments) of the rule or guideline; and references to other relevant rules and guidelines.

### **Organization:**

*C.1 Rules and Guidelines for the Refinement of Entities*

*C.2 Rules and Guidelines for the Design of Object Classes and Class Hierarchies*

*C.2.1 Rules and Guidelines for the Design of Object Classes*

*C.2.2 Rules and Guidelines for the Design of Class Hierarchies*

## ***C.1 Rules and Guidelines for the Refinement of Entities***

---

The following conventions are adopted here to label the rules and guidelines for later reference:

- R for rules and G for guidelines,
- REFI for entity refinement (i.e., the first category of rules and guidelines) and DESI for design (i.e., the second category), and
- O for the design of object classes, A for the design of class atttributes, M for the design of class methods, and H for the design of object class hierarchies.

Component rules and guidelines within each group are labeled using subscripts such as .1, .2, .3, etc.

Batini et al. [92] explains the design schema qualities such as minimality, expressiveness, efficiency, and self-explanation that are mentioned in the rules and guidelines.

### ***R-REFI.1 Complete entity descriptions***

*Statement:* An entity description must include all the data items that the entity needs.

*Explanation:* This rule ensures that all entity descriptions are complete and self-sufficient in terms of their data items. It requires the modeler to review the entity description at hand before taking the next transformation step.

*Example:* An “AISC L-Shape Descriptions” entity description (including “dimension d1,” “dimension d2,” and “thickness t” data items) needs another data item, “dimension unit,” to complement its existing data items.

*Discussion:* If the description of a primitive entity is incomplete, the modeler must also consider the next two rules.

### ***R-REFI.2 Safe addition of necessary data items to a primitive entity description***

*Statement:* An incomplete primitive entity description can include necessary new data items only if they do not affect its cohesion.

*Explanation:* If a primitive entity description needs additional data items, the modeler can add those items only if they will not affect the cohesion of the entity. Otherwise, the modeler must re-evaluate the entity’s cohesion (and possibly decompose the entity as in Phase 3).

*Example:* The modeler can “safely” add the “dimension unit” data item to the “AISC L-Shape Descriptions” primitive entity description in the previous rule.

*Discussion:* Descriptive data items are items such as units, names, verbose descriptions, notes, references, user-defined object identifiers, etc. that further enhance an entity description. “Dimension unit” in the example is such an item. Adding such data items will not affect the cohesion of primitive entities.

**R-REFI.3**     *Addition of optional data items to an entity description*

**Statement:**     *An entity description can include optional data items (i.e., desirable but non-essential), which will be converted into optional attributes in the equivalent class definition. However, these data items must be limited to few (or even one) per primitive entity.*

**Explanation:**     This rule provides the modeler with the flexibility of adding information to an entity description. However, optional data items affect the time-cohesion of primitive entities since the optional data item values need not be assigned at any time. (By definition, a primitive entity is time-cohesive if all its data item values must be specified at the same time.) They also affect the entities' use-cohesion since they may not always be used with the existing data items of the entity. (By definition, an entity is use-cohesive if all or none its data items are used in the "primary data uses" of the domain.) Therefore, optional data items must be limited to few (or even one) per primitive entity.

**Example:**     The description of a primitive entity, "Behavior Response Forces," includes one optional data item called "optional designation" (whose values can be "F<sub>A</sub>," "M<sub>B</sub>", "T<sub>C</sub>," etc.).

**R-REFI.4**     *Include all dependencies in the entity description.*

**Statement:**     *An entity description must capture all the dependencies of the entity on other entities in the schema.*

**Explanation:**     According to this rule, the modeler must examine all possible dependencies of the entity being considered and represent them in the entity description. The subsequent rules specify different representations for various types of entity dependencies.

**Example:**     A "Design History Descriptions" entity has two dependencies: a dependency by reference on "Design Artifact Descriptions" and another by definition on "Design Operation Descriptions."

**R-REFI-5**     *Representing entity dependencies*

This rule includes the following component rules.

**R-REFI-5.1**     *Options for representing entity dependencies by derivation*

**Statement:**     *If a primitive entity, X, is dependent by derivation on another primitive entity, Y, the modeler can choose one of the following options to represent the dependency: She can (1) incorporate a "derives" relationship of derivation type into the description of Y and include a user-defined object identifier in the descriptions of both X and Y, (2) incorporate a "derives" relationship of derivation type into the description of Y and a "derived-from" inverse relationship into the description of X, or (3) eliminate X by incorporating its data items as "derived data items" into the description of Y.*

**Explanation:** No single rule applies to all cases of entity dependencies by derivation. Instead, each of the following three component rules describes its own applicable cases and presents a way to represent the dependency.

**References:** This rule includes the following three component rules.

**R-REFI-5.1.1 Representing an entity dependency by derivation using a “derives” relationship and a user-defined object identifier**

**Statement:** *If a primitive entity, X, is dependent by derivation on another primitive entity, Y, the modeler can incorporate a “derives” relationship of derivation type into the description of Y and can include a user-defined object identifier in the descriptions of both X and Y.*

**Explanation:** Y can include a reference data item (e.g., “derived X”) and convert it into a relationship attribute whose relationship is “derives.” This relationship attribute captures the semantics of derivation between X and Y. (e.g., the deletion of an instance of Y will trigger the deletion of its derived instances of X.) This attribute can be delayed, given that the delay type is specified at each new instance. In X’s description, a user-defined object identifier can take the place of the inverse relationship “derived-from.” This identifier must be converted into a required and unique attribute. In fact, it can be used to search the database for a matching instance of X before storing a new or modified instance of Y. Y’s description must include the same identifier.

**Example:** An “AISC L-Shape Section Properties” entity is dependent by derivation on another “AISC L-Shape Descriptions” entity. The description of the latter includes an identifier (“shape size designation”) and a reference data item (“derived section properties”) to “AISC L-Shape Section Properties,” which also includes the same identifier in its description.

**Discussion:** This rule applies to cases where the user already uses object identifiers such as the shape and size designations in the above example for instances of the deriving entity, Y. (In other cases, requiring a unique identifier for each instance of X or Y can be a burden to the user.) Moreover, in these cases, the attribute values of the derived instance such as standard shape section properties are specified once and will rarely change. The advantage here is that the stored derived instances can be retrieved as often as needed and the matching identifier eliminates the need for implementing and maintaining the inverse relationship in X. The tradeoff is that there is no direct access from X to Y.

**Reference:** This rule also refers to Rule R-DESI-A.2, which deals with user-defined object identifiers.

**R-REFI-5.1.2 Representing an entity dependency by derivation using a pair of relationships of derivation type**

**Statement:** *If a primitive entity, X, is dependent by derivation on another primitive entity, Y, the modeler can incorporate a “derives” relationship of derivation type into the description of Y and an inverse “derived-from” relationship into the description of X.*

**Explanation:** Y can include a reference data item (e.g., “derived X”) and convert it into a relationship attribute whose relationship is “*derives*.” This attribute can be delayed, given that the delay type is specified at each new instance. X can include a reference data item (e.g., “source Y”) and convert it into a required relationship attribute whose relationship is “*derived-from*.”

**Example:** A “Path Connectivities” entity is dependent by derivation on another “Path Descriptions” entity. The description of the latter includes a data item that refers to “Path Connectivities” (“derived path connectivity”), which in turn includes a reference data item, “source path description.”

**Discussion:** This rule applies to cases where users do not always associate an identifier with each instance of the deriving entity, Y, and where the computation in the derivation is rather involved. The advantage of this option over the previous one is that the inverse relationship in X provides direct access to Y. Finding an instance of Y that matches a given instance of X is much more efficient in this case than in the case of the previous rule. The tradeoff is that the modeler carries the burden of implementing the inverse *derived-from* relationship and maintaining both relationships.

### ***R-REFI-5.1.3 Representing an entity dependency by derivation using derived attributes***

**Statement:** *If a primitive entity, X, is dependent by derivation on another primitive entity, Y, the modeler can eliminate X by incorporating its data items as “derived data items” into the description of Y.*

**Explanation:** Y can include the data items in X’s description as its own data items and convert them into derived attributes. These attributes must be implemented as methods in the equivalent class definition of Y. Consequently, they do not affect the source-cohesion of the primitive entity, Y. The attribute values must not be stored in the database, but rather computed on demand using these methods. X needs not be represented as another class in the domain primitive schema.

**Example:** A “Cartesian Vector Direction Cosines” entity is dependent by derivation on another “Cartesian Vectors” entity. The three direction cosines (x, y, z) of the former entity are incorporated as derived data items into the description of the latter entity.

**Discussion:** This rule applies to cases where the attribute values of the derived instances are subject to frequent modification and where the computation in the derivation is rather simple. One advantage over the two previous options is that the modeler eliminates the entity dependency being considered and, as a result, deals with one less entity (i.e., X). Another advantage is that the derived attribute values require no storage space and, when computed on demand, are always up to date. The tradeoff is the high cost of computing those values at each request.

### ***R-REFI-5.2 Options for representing entity dependencies by definition***

**Statement:** *If a primitive entity, X, is dependent by definition on another primitive entity, Y, the modeler can choose between the following options to represent the dependency: (1) representing Y as an abstract data type and*



using this data type in the description of X, or (2) incorporating a “subpart” relationship of aggregation type into the description of X.

**Explanation:** No single rule applies to all cases of entity dependencies by definition. Instead, both of the following rules describe their applicable cases and present a way to represent the dependency.

**References:** This rule includes the following two component rules.

**R-REFI-5.2.1 Representing an entity dependency by definition using an abstract data type**

**Statement:** *If a primitive entity, X, is dependent by definition on another primitive entity, Y, the modeler can represent Y as an abstract data type and use this data type in the description of X.*

**Explanation:** X can include the data items in Y’s description as a single data item and convert it into a required attribute is of abstract data type. In fact, Y can be implemented as an abstract data type rather than a class. In object-oriented programming languages such as C++, an abstract data type corresponds to a “struct” (i.e., a user-defined data structure), which contains fields of data and which, unlike an object class, does not have an object identity and behavior.

**Example:** A “Structural Analysis Node Descriptions” entity is dependent by definition on another “Cartesian Coordinates” entity. An abstract data type can represent “Cartesian Coordinates.” The description of “Structural Analysis Node Descriptions” includes a data item of that data type, namely “node coordinates.”

**Discussion:** This rule applies to cases where it makes sense to represent the auxiliary entity, Y, as an abstract data type rather than as a class. One advantage of this option is that the modeler eliminates the entity dependency being considered and, as a result, deals with one less entity (i.e., Y). Another advantage is that the information about Y can be quickly accessed from within the description of X. The tradeoffs are losing the object identity of Y and losing the ability to incorporate object behavior into Y.

**R-REFI-5.2.2 Representing an entity dependency by definition with a relationship of aggregation type**

**Statement:** *If a primitive entity, X, is dependent by definition on another primitive entity, Y, the modeler can incorporate a “subpart” relationship of aggregation type into the description of X.*

**Explanation:** X can include a reference data item (e.g., “subparts Y” or “own Y”) and convert into a required relationship attribute whose relationship is “subpart.” When no other entities depend on Y, Y can include a reference data item (e.g., “parent X”) and convert it into a required relationship attribute whose relationship is “part-of.” (Otherwise, Y need not include a reference data item for each inverse relationship to those entities that depend on Y.)

**Example:** A “Parameterized Requirement Statements” entity is dependent by definition on another “Design Parameters” entity. The description of the latter includes a data item that refers to “Design Parameters” (“own design

parameters”). However, that entity does not include a reference data item for the inverse relationship since entities such as “Performance Criteria,” “Design Features,” and “Design Constraints” are also dependent on it.

**Discussion:** This rule applies to cases where using the previous rule does not apply (i.e., where Y must be represented as a class rather than an abstract data type). The advantage of this option over the previous one is that the equivalent Y class definition has object identity and can encapsulate behavior in the form of methods. The tradeoff is that the modeler must deal with two entities and must implement and maintain the relationships between them.

### **R-REFI-5.3 Options for representing entity dependencies by reference**

**Statement:** *If a primitive entity, X, is dependent by reference on another primitive entity, Y, the modeler can choose between the following options to represent the dependency: (1) incorporating a “referred-to” relationship of referential association type into the description of X, or (2) including user-defined object identifiers in the descriptions of both X and Y to associate their instances.*

**Explanation:** No single rule applies to all cases of entity dependencies by reference. Instead, both of the following two rules describe their applicable cases and present a way to represent the dependency.

**References:** This rule includes the following two component rules.

#### **R-REFI-5.3.1 Representing an entity dependency by reference with a relationship of association type**

**Statement:** *If a primitive entity, X, is dependent by reference on another primitive entity, Y, the modeler can incorporate a “referred-to” relationship of referential association type into the description of X.*

**Explanation:** X can include a reference data item (e.g., “referred Y”) and convert it into a required relationship attribute whose relationship is “referred-to.” When no other entities depend on Y, Y can include a reference data item (e.g., “referring X”) and convert it into a required relationship attribute whose relationship is “referred-by.” (Otherwise, Y need not include a reference data item for each inverse relationship to those entities that depend on Y).

**Example:** A “Loading Condition Specifications” entity is dependent by reference on another “Load Source Specifications” entity. The description of the former includes a data item that refers to “Loading Condition Specifications” (“referred load sources”). However, that entity does not include a reference data item for the inverse relationship since entities such as “External Load Specifications” are also dependent on it.

**Discussion:** This rule applies to cases where the user frequently requests the information about Y to which X refers. The advantage here is that the relationship in X provides direct access to Y, and therefore, finding a instance of Y to which a given instance of X refers is much quicker than is the case for the next rule. The tradeoff is that the modeler must implement and maintain the relationships.

**R-REFI-5.3.2 Representing an entity dependency by reference with a user-defined object identifier**

**Statement:** *If a primitive entity, X, is dependent by reference on another primitive entity, Y, the modeler can include a user-defined object identifier in the descriptions of both X and Y to associate their instances.*

**Explanation:** Y can include a user-defined object identifier (e.g., “Y identifier”), and X can include a data item for associating its instances with those of Y (e.g., “referred Y identifier”). This data item can be used to search the database for an instance of Y to which a given instance of X refers. Similarly, the required and unique identifier of an instance of Y can be used to search for instances of X that refer to it.

**Example:** An “Axial Element Displacements” entity is dependent by reference on another “Structural Analysis Element Descriptions” entity. The latter entity includes in its description a user-defined object identifier, “element identifier.” The former includes a data item that matches the identifier in “Structural Analysis Element Descriptions,” namely “referred element identifier.”

**Discussion:** This rule applies to cases where the user already uses object identifiers such as the one in the above example for instances of the referred entity, Y. (In other cases, requiring a unique object identifier for each instance of X or Y can be a burden to the user.) Moreover, in these cases, specifying which instance is referred to (as in the above example, referred member identifier) is more important than establishing a link for eventually accessing the information about that instance. The advantage of this option over the previous one are that the modeler does not need to implement and maintain any relationships. The tradeoffs are that user-defined object identifiers do not provide direct access links from X to Y and that searching the database using these identifiers can be computationally costly.

**Reference:** This rule also refers to Rule R-DESI-A.2, which deals with user-defined object identifiers.

## **C.2 Rules and Guidelines for the Design of Object Classes and Class Hierarchies**

---

### **C.2.1 Rules and Guidelines for the Design of Object Classes**

#### **G-DESI-O.1 Use explicit class names.**

*Statement:* The name of a class should explicitly articulate what the class represents.

*Explanation:* This guideline is designed to enhance the self-explanation and expressiveness of the resulting schema (whether it is primitive or composite).

*Example:* “Square Shape Descriptions,” “Solid Rectangle Shape Descriptions,” “Triangle Shape Descriptions,” and “Solid Circle Shape Descriptions” classes represent four common geometric shapes.

*Exception:* In cases where the class name is long and inconvenient to the implementation (e.g., “Cross-Section Modulus of Elasticity Properties”), the modeler can shorten it while making it as explicit as possible (e.g., “Section Elastic Moduli”).

*References:* This guideline goes together with Guideline G-DESI-A.1 and G-DESI-M.1, which deal with explicit names for class attributes and class methods.

#### **G-DESI-O.2 Reconsider classes with few attributes.**

*Statement:* A class with one or a few attributes should be considered for possible elimination using Guidelines G-DESI-H.2 to H.4.

*Explanation:* This guideline is intended to enhance the efficiency of the resulting schema and to make sure that a class is only introduced when it is needed (and thus meaningful to the resulting schema). In general, a class that has few (or even one) attributes and that has relationships with other classes can be eliminated by moving its attributes to the related classes.

*Example:* A “Structural Analysis Element Descriptions” class that has only one attribute, “element identifier,” was reconsidered but not eliminated because it provides a place-holder for adding subclasses representing more specific types of analysis elements.

*References:* Guidelines G-DESI-H.2 to H.4 show when to include a class in a class hierarchy and discuss the special case of abstract superclasses with no attributes.

#### **G-DESI-A.1 Use explicit attribute names.**

*Statement:* The name of an attribute should explicitly articulate the object property that the attribute represents.

*Explanation:* This guideline is designed to enhance the expressiveness and self-explanation of the resulting schema (whether it is primitive or composite).

**Example:** The name of a relationship attribute can include a prefix such as “derived,” “own,” or “referred,” which clearly denotes the relationship type. Examples are “derived load cases,” “own load parameters,” and “referred load sources.”

**Exception:** In cases where the attribute name (e.g., “standard derived cross section properties”) becomes long and inconvenient to the implementation, the modeler can shorten it (e.g., “derived section properties”) while making it as explicit as possible.

**References:** This guideline goes together with Guidelines G-DESI-O.1 and G-DESI-M.1, which deal with explicit names for object classes and class methods.

### ***G-DESI-A.2 When to include a user-defined object identifier***

**Statement:** *A class does not need a user-defined object identifier unless the user uses such an identifier. In that case, such an identifier must be a required and unique attribute.*

**Explanation:** In the object-oriented paradigm, the object identity of each instance is globally unique and is created and maintained by the system. However, if the user chooses to define an explicit attribute as her own identifier of instances, then that attribute must be given a unique value in each instance.

**Example:** “shape size designation,” “material designation” and “requirement identifier” are user-defined object identifiers for “AISC Shape Descriptions,” “Material Properties,” and “General Requirements” classes respectively.

### ***G-DESI-A.3 Remove redundant relationship attributes.***

**Statement:** *Redundant relationship attributes should be removed.*

**Explanation:** This guideline ensures minimality of the resulting schema. Redundant relationship attributes occur when different paths following the relationship links go from one source class to the same destination class and produce the same effect. This guideline applies to relationships of any type. In particular, cycles of relationships must be completely eliminated.

**Example:** A “Load Condition Specifications” class has relationships with two other classes, “Load Source Specifications” and “Load Case Specifications.” “Load Case Specifications,” in turn, has a relationship with “Load Source Specifications.” In this case, the last relationship creates a cycle and must be removed.

### ***G-DESI-A.4 No new attributes in composite classes***

**Statement:** *The introduction of new attributes in composite classes should be minimized.*

**Explanation:** A composite class should not introduce any attributes other than those of primitive or composite classes from which it is defined. This guideline implies that the domain primitive schema should, if possible, provide all

the primitive classes necessary to support different user views in the domain.

**Example:** The definition of the composite class, “Tower Members As Analyzed,” uses the following primitive classes: “Two-Node Analysis Element Descriptions,” “Section Areas,” “Material Moduli,” “Load Application Specifications,” and “Cartesian Coordinate Systems.” It does not add any new attributes.

***G-DESI-M.1 Use explicit method names.***

**Statement:** *The name of a class method should explicitly articulate what the method performs.*

**Explanation:** This guideline is designed to enhance the expressiveness and self-explanation of the resulting schema.

**Example:** The name of a method for setting an attribute value includes a prefix such as “set” to denote the operation of the method clearly. Similarly, the name of a method for retrieving an attribute value includes a prefix such as “get.” For example, the “distance d1” attribute of the “AISC Angle Shape Description” class corresponds to two methods, “set distance d1” and “get distance d1.”

**References:** This guideline goes together with Guidelines G-DESI-O.1 and G-DESI-A.1, which deal with explicit names for object classes and class attributes.

***R-DESI-M.2 Define methods for computing derived attribute values.***

**Statement:** *A class method must be defined for each derived attribute in order to compute its value on demand.*

**Explanation:** A derived attribute is dependent on other attributes; a method is defined to compute that attribute, which is derived on demand. (By contrast, an independent attribute is stored in the database.)

**Example:** The three derived attributes of a “Cartesian Vectors” class, which represent x, y, and z directional cosines, yield three corresponding methods: “compute cosine x,” “compute cosine y,” and “compute cosine z.”

***R-DESI-M.3 Defining methods to update and retrieve independent attribute values.***

**Statement:** *A pair of class methods must be defined for each independent attribute to set and retrieve its value. “Set” methods must have at least one argument, while “get” methods must return at least one response value.*

**Explanation:** Attributes are typically private to their object class. This rule ensures that a class has access to its attributes via methods. In fact, these methods set the object state or provide the means to inquire about the object state. “Set” methods may not return any response, while “get” methods may not have any argument.

**Example:** The two independent attributes of the “Cartesian Vectors” class in the previous rule, which represent the magnitude and direction, yield four

methods: “set magnitude,” “get magnitude,” “set direction,” and “get direction.”

### **C.2.2 Rules and Guidelines for the Design of Class Hierarchies**

#### **R-DESI-H.1 Do not misuse relationships of generalization type in class hierarchies.**

*Statement:* Two classes belong to the same class hierarchy only if one represents a more specialized definition of the concept represented by the other.

*Explanation:* This rule applies to both primitive and composite classes. It is designed to avoid misusing these relationships, as pointed out in [Phan 91b].

*Example:* A “Structural Members” class should not be a subclass of “Buildings.”

#### **R-DESI-H.2 Define a common superclass.**

*Statement:* If two or more classes have one or more attributes in common, these classes must have a new common superclass in the class hierarchy.

*Explanation:* This rule is designed to remind the modeler to take advantage of generalization. However, the modeler must weigh the tradeoffs between introducing a superclass and implementing the code of the common attributes in all classes sharing those attributes.

*Example:* “AISC I-Shape Descriptions,” “AISC T-Shape Descriptions,” and “AISC C-Shape Descriptions” classes can share a common superclass, “AISC Symmetric Rolled Shape Descriptions,” which includes four attributes common to these classes.

#### **G-DESI-H.3 Define a typical class.**

*Statement:* If several instances share two or more common attribute values, they should be instances of a new typical class in the class hierarchy.

*Explanation:* This guideline is designed to remind the modeler to take advantage of typical classes, which are very beneficial in representing data of facility design objects. The new typical class has default attributes whose values are the commonly used values.

*Example:* A “Tower Leg Analysis Element Descriptions” typical class is subclass of a “Two-Node Analysis Element Descriptions” class. The typical class’ attributes “element type,” “start support type,” “end support type,” and “length unit” have the default values of “truss element,” “hinged,” “hinged,” and “feet” respectively.

**G-DESI-H.4 When to introduce a new subclass in a class hierarchy**

This guideline includes the following component guidelines.

**G-DESI-H.4.1 Do not invent a subclass with no attributes.**

**Statement:** A new subclass in a class hierarchy should add at least one new attribute to the set inherited from its superclass.

**Explanation:** This guideline ensures that a new subclass is added only when the subclass has a distinct identity in its hierarchy.

**Example:** Three new subclasses, “AISC I-Shape Descriptions,” “AISC T-Shape Descriptions,” and “AISC C-Shape Descriptions,” are added to the class hierarchy representing AISC standard shape descriptions. Each subclass has at least one attribute.

**Exception:** Superclasses with few (or even one) attribute are useful where they provide a place-holder for: (1) adding more specialized subclasses later, or (2) defining relationship attributes of other classes, which can be linked to any subclasses of the superclass. These superclasses are called “abstract superclasses.” The example given in Guideline G-DESI-O.2 demonstrates the first case. An illustration of the second case is the abstract superclass, “AISC Rolled Shape Descriptions.” This superclass provides a convenient place-holder for an instance of “AISC Combination Shape Descriptions” that can later be linked to instances of subclasses of the superclass (e.g., “AISC L-Shape Descriptions,” “AISC I-Shape Descriptions,” “AISC C-Shape Descriptions”).

**References:** This rule must be used in conjunction with Rule R-DESI-H.1, Rule R-DESI-H.2 and the rules presented hereafter.

**G-DESI-H.4.2 Do not invent a subclass when a relationship will do.**

**Statement:** A new subclass should not be introduced if its only additional feature can be represented by a relationship in its superclass.

**Explanation:** This guideline is designed to avoid replacing relationships with subclasses, as pointed out in [Howard 92].

**Example:** A “Gravity Load Resisting Frames” class should not be introduced as a new subclass of “Frames,” since its only new feature can be represented by a relationship to the appropriate load types supported by those frames.

**G-DESI-H.4.3 Do not add a level when a discriminating attribute will do.**

**Statement:** Another level in the class hierarchy should not be introduced if a single attribute in the superclass can distinguish the different subclasses.

**Explanation:** This guideline is designed to enhance the minimality and efficiency of the resulting schema.

**Example:** An “AISC Combination Shape Descriptions” class includes an attribute to distinguish different types of shape combinations (i.e., “combination shape type”), thereby eliminating the need for another unnecessary level of



subclasses such as “S-C Combination Shape Descriptions,” “C-L Combination Shape Descriptions,” “C-C Combination Shape Descriptions,” etc.

#### ***G-DESI-H.5 Shallow Primitive Characterization Hierarchies***

***Statement:*** *Primitive characterization hierarchies should be kept shallow.*

***Explanation:*** Several shallow primitive class hierarchies are preferred to a few deep hierarchies. As a rule of thumb, a primitive characterization hierarchy should contain at most three levels. Otherwise, the modeler should first review the hierarchy using Rule R-DESI-H.1 and the preceding set of guidelines. She should consider eliminating classes at the intermediate levels if possible. In addition, the modeler should review the overall concept that was used to build the class hierarchy. A deep primitive characterization hierarchy is indicative of a concept that is not sufficiently distinctive. In that case, the concept should be refined into several subconcepts, each of which will lead to a more shallow primitive class hierarchy.

***Example:*** A deep primitive characterization hierarchy on shape descriptions was broken up into two separate hierarchies: one on geometric shape descriptions and the other on AISC standard shape descriptions.

***References:*** Rule R-DESI-H.1 and rule set R-DESI-H.4.

#### ***G-DESI-H.6 No Composite Class Hierarchies***

***Statement:*** *Composite class hierarchies should not be necessary.*

***Explanation:*** Although by definition, a composite class can be a subclass of another composite class, composite class hierarchies are not recommended here for the following reason: The flexibility of the P-C Approach in representing facility design objects comes from allowing the user to define any composite class when needed from a given primitive schema. At any time, the user can instantly customize a composite class by selecting primitive classes from primitive characterization of this schema. Therefore, both predefined composite classes and composite class hierarchies should not be necessary.

***Exception*** A typical class (e.g., “Tower Cage Members As Designed”) can be a subclass of a composite class (e.g., “Tower Members As Designed”) to capture the common values shared by instances of the composite class (e.g., length, member size, material designation).

# **Appendix D**

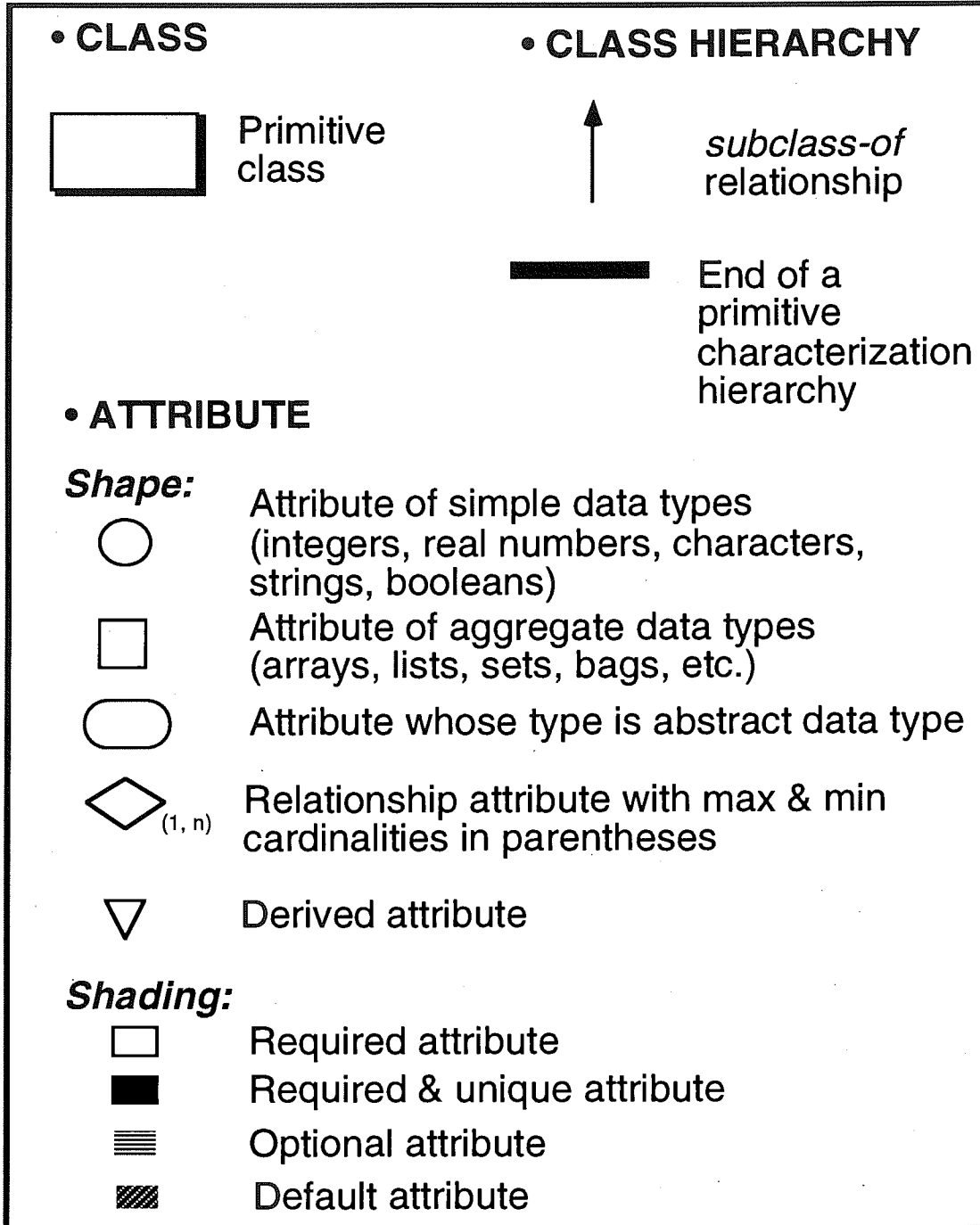
---

## **Documentation of the Tower Domain Primitive Schema**

### **Abstract:**

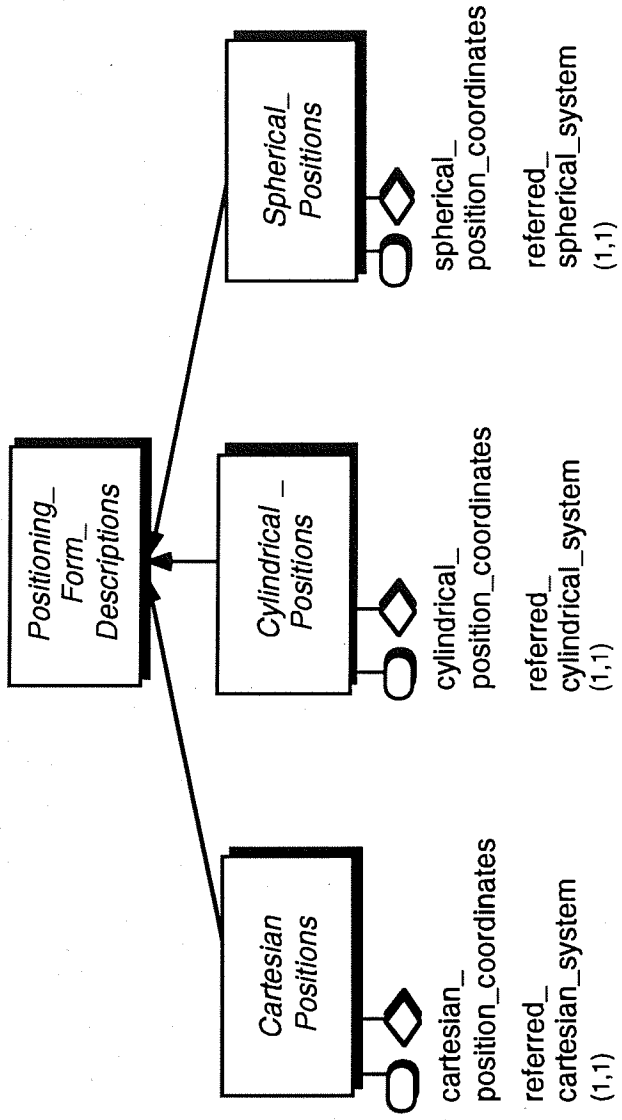
This appendix contains the documentation of the domain primitive schema for transmission towers. Specifically, this appendix shows the graphical representations of all the primitive characterization hierarchies included in that schema. These graphical representations are based on those presented in [Batini 92] for conceptual data modeling. The first figure is the legend for the graphical representations that follow. The definitions of some primitive classes in this schema are based on those of the entities in the PDES/STEP Integrated Product Information Model (IPIM) [Wilson 88].

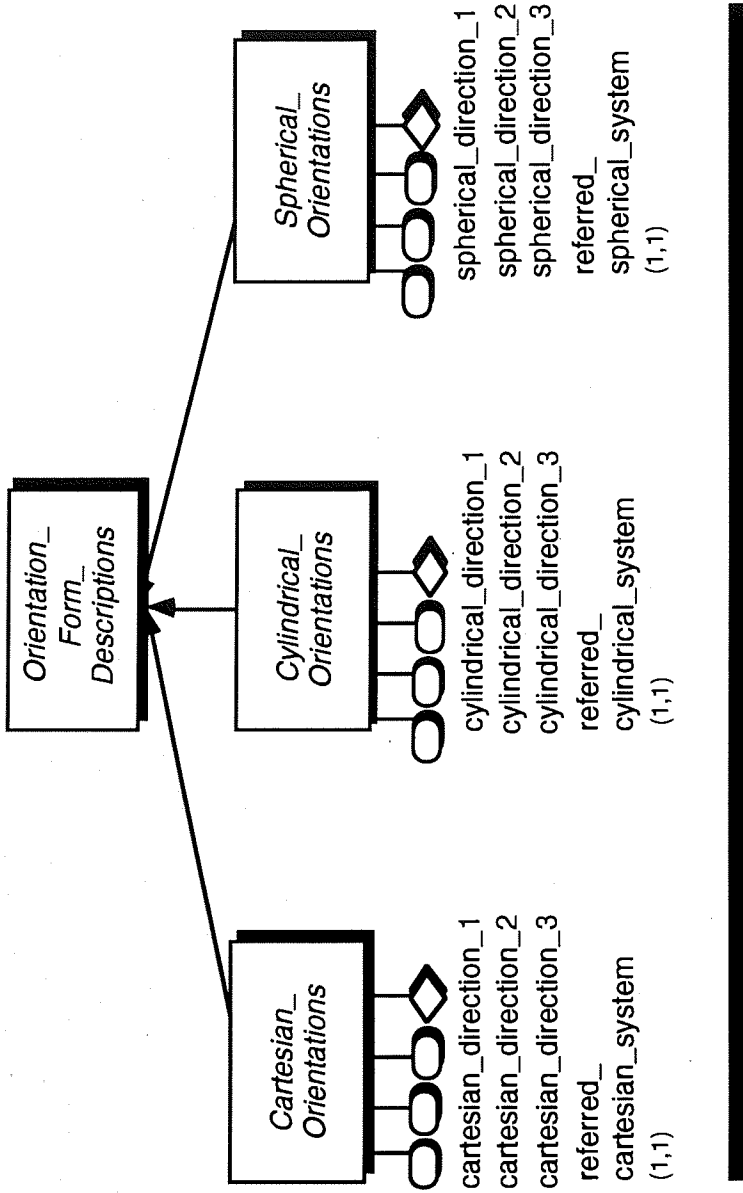
LEGEND

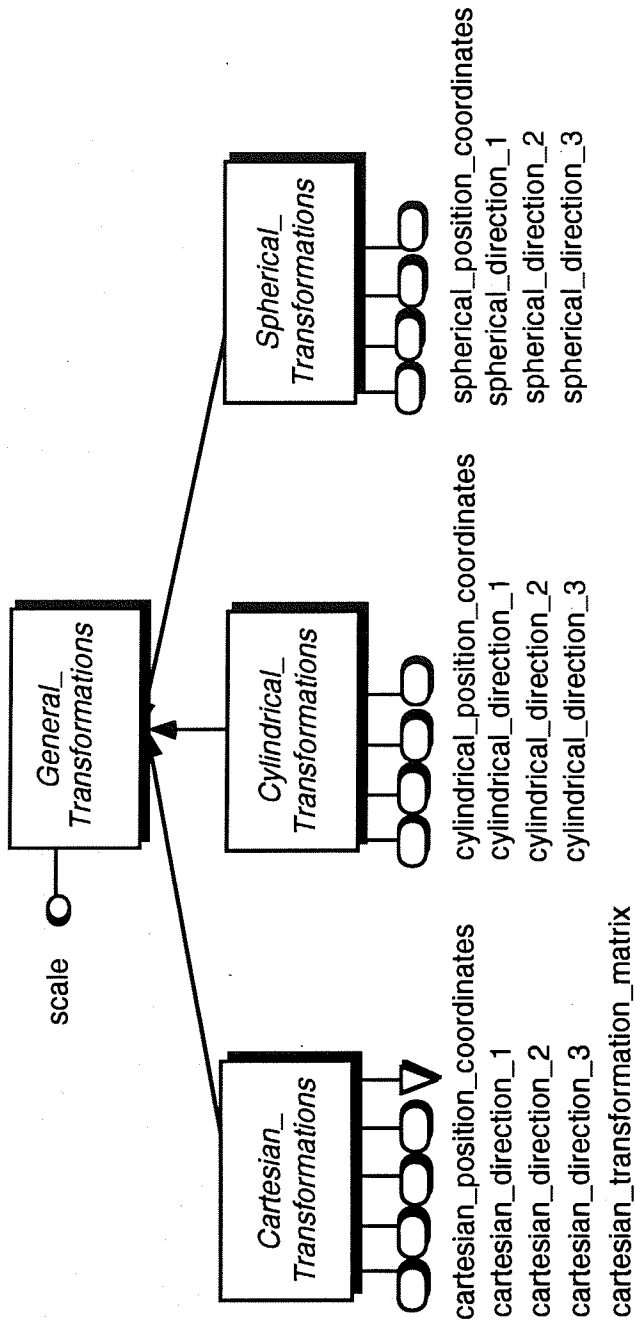


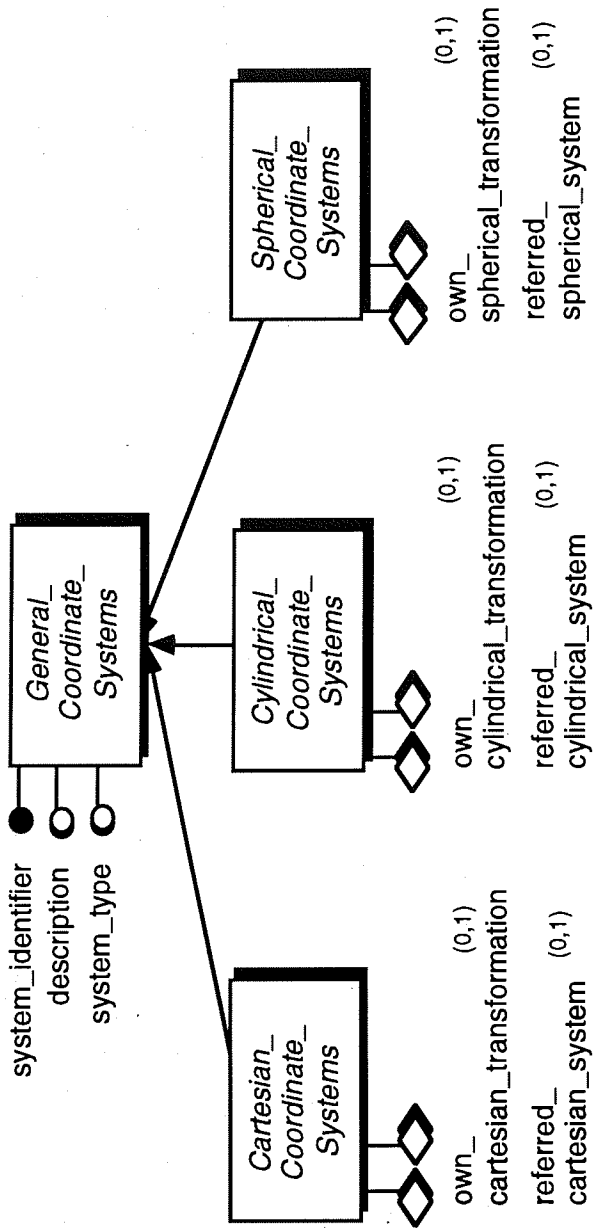
**FIGURE D.1:** Legend for the Graphical Representations of Primitive Characterization Hierarchies that follow.

**FIGURE D.2: Primitive Characterization Hierarchies Describing  
SPATIAL REFERENCE FORM.**

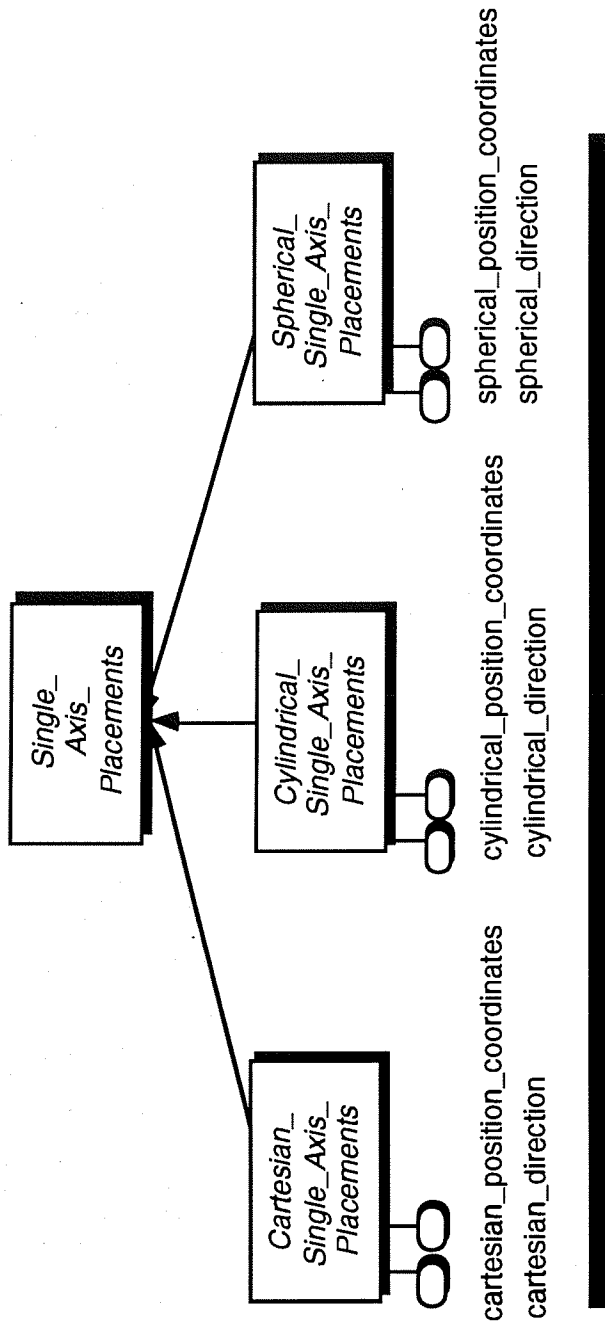


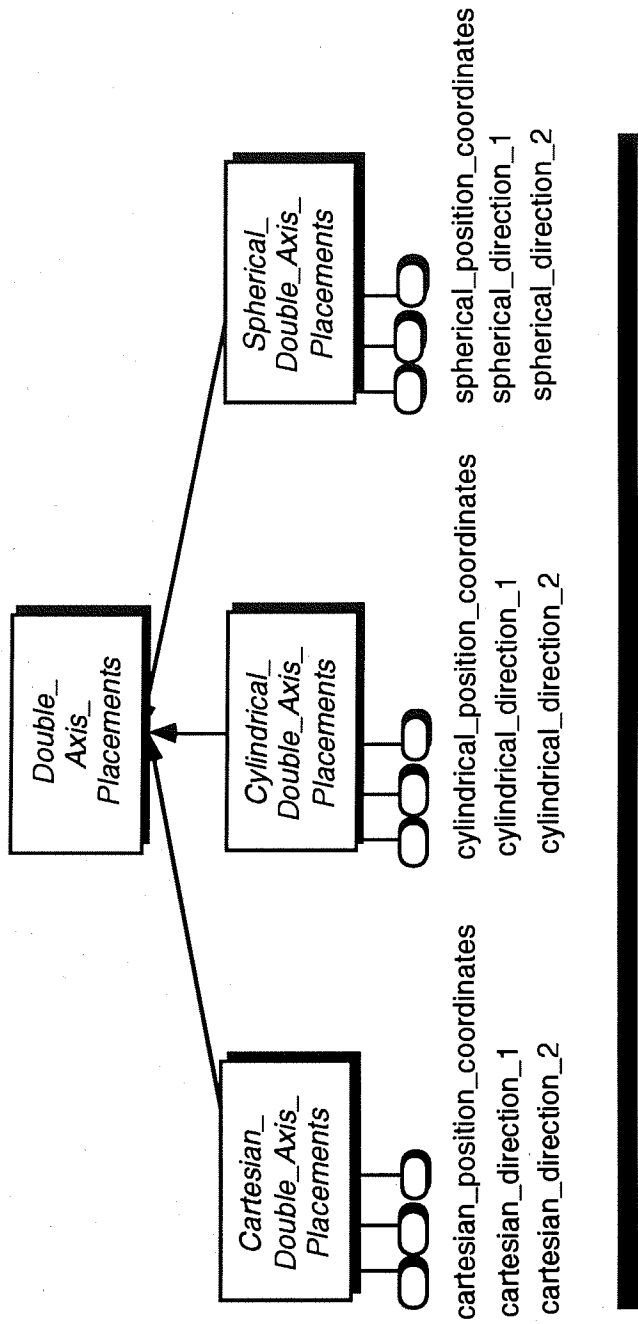


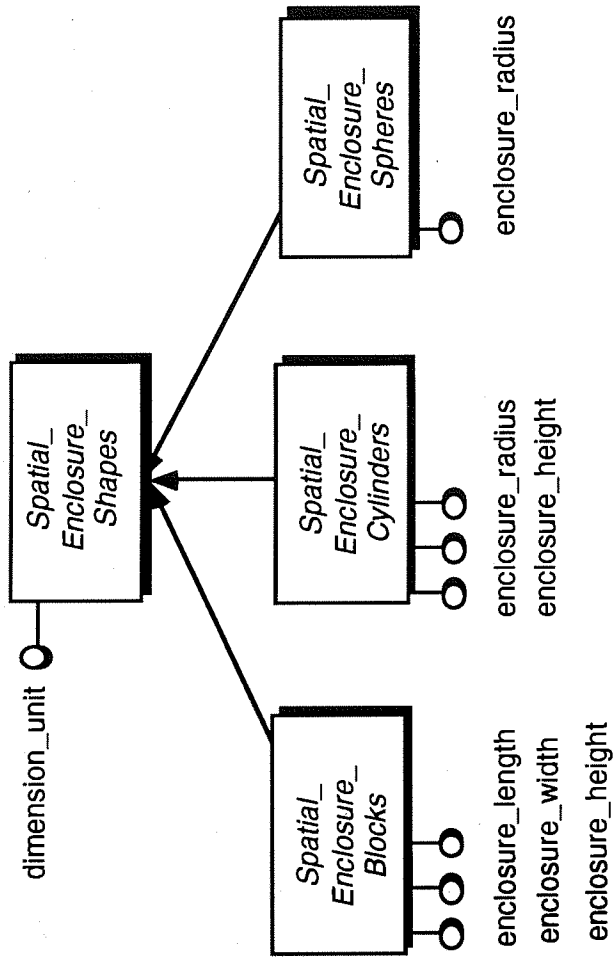


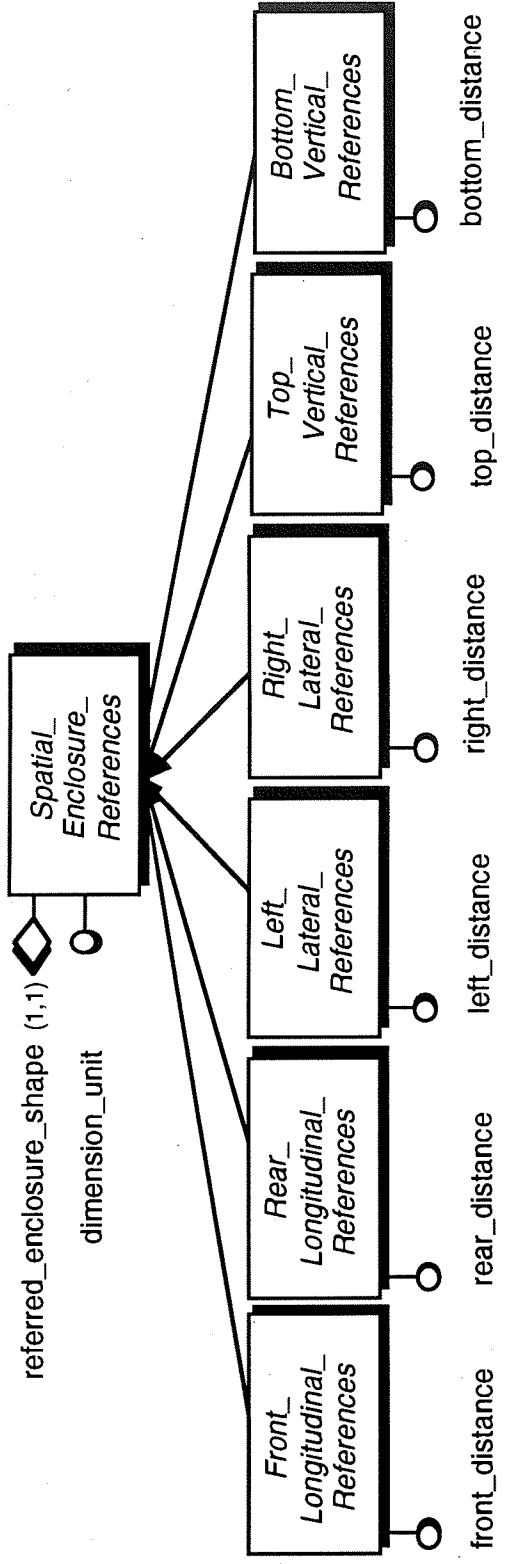




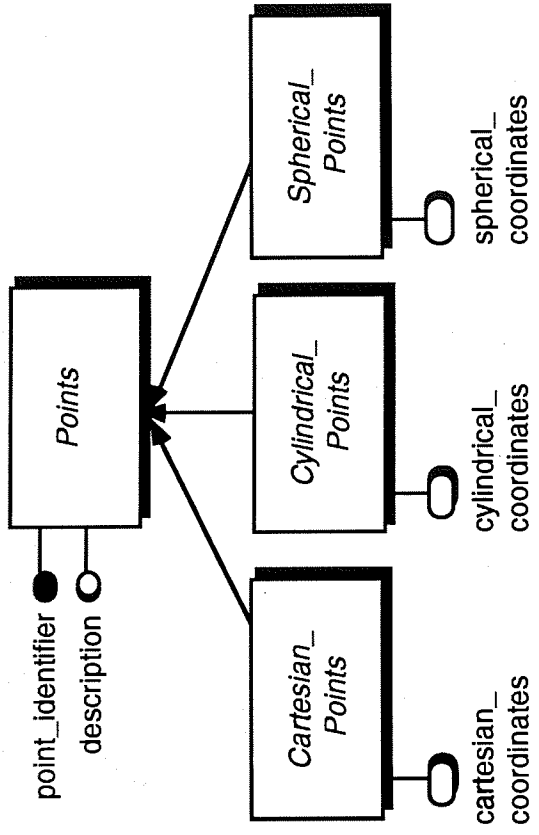


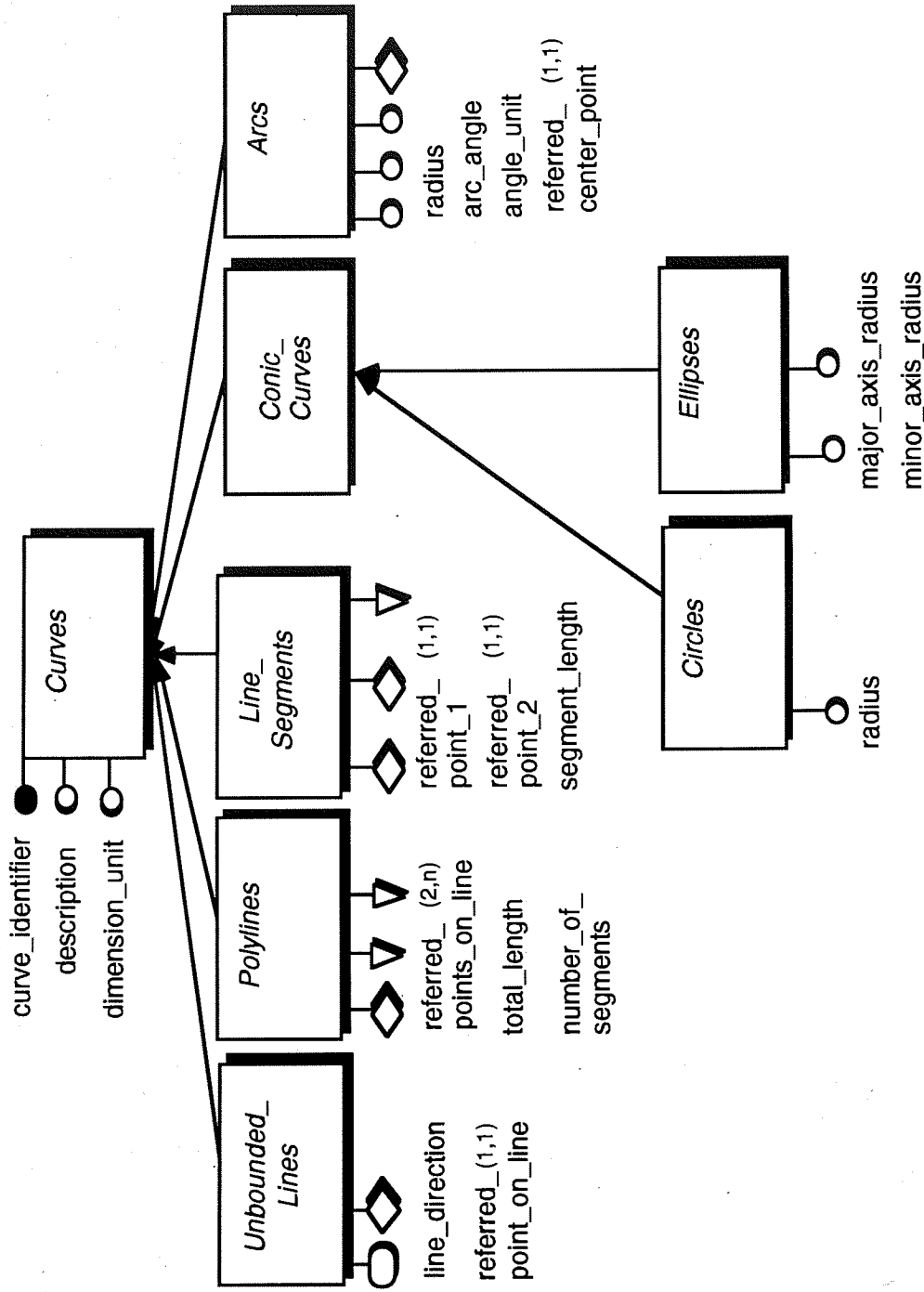


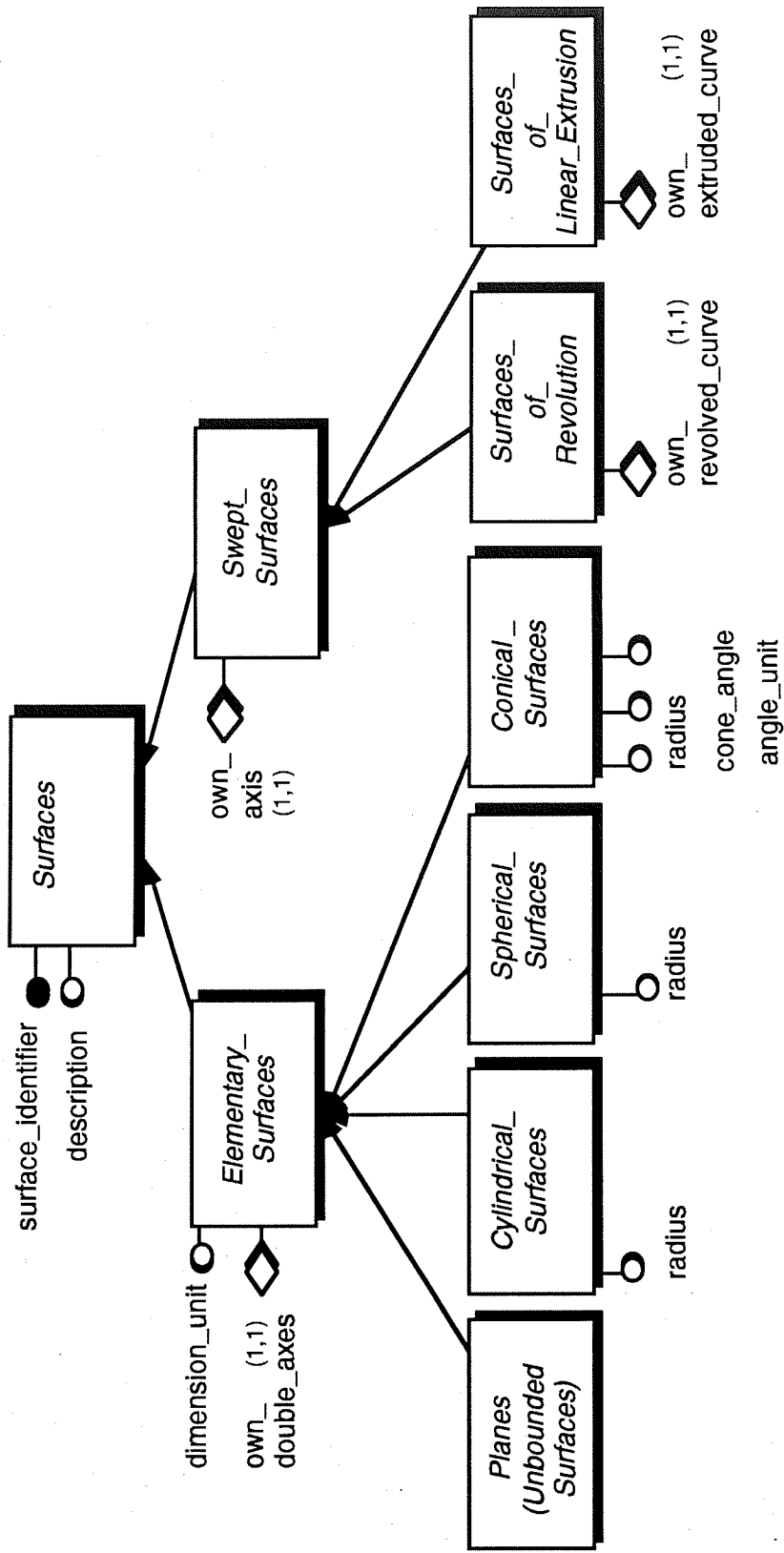




**FIGURE D.3: Primitive Characterization Hierarchies Describing  
GEOMETRY FORM.**

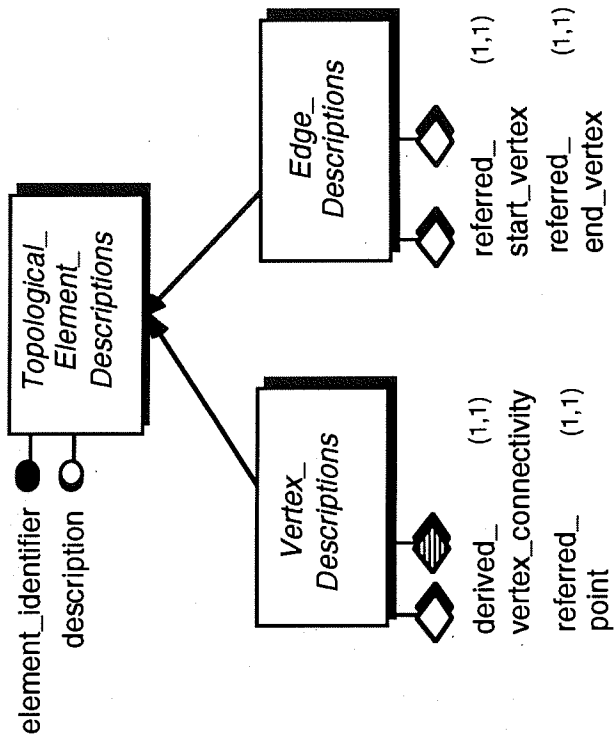




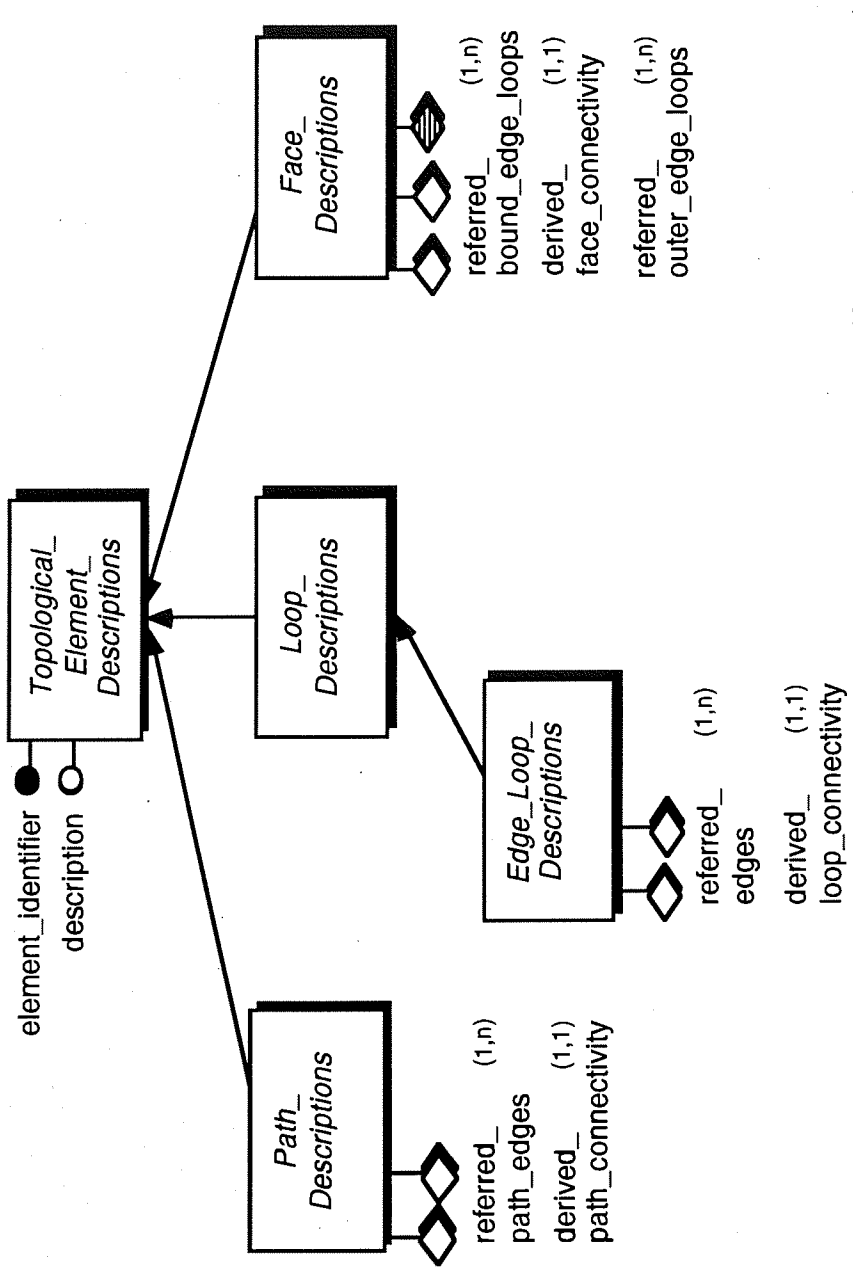




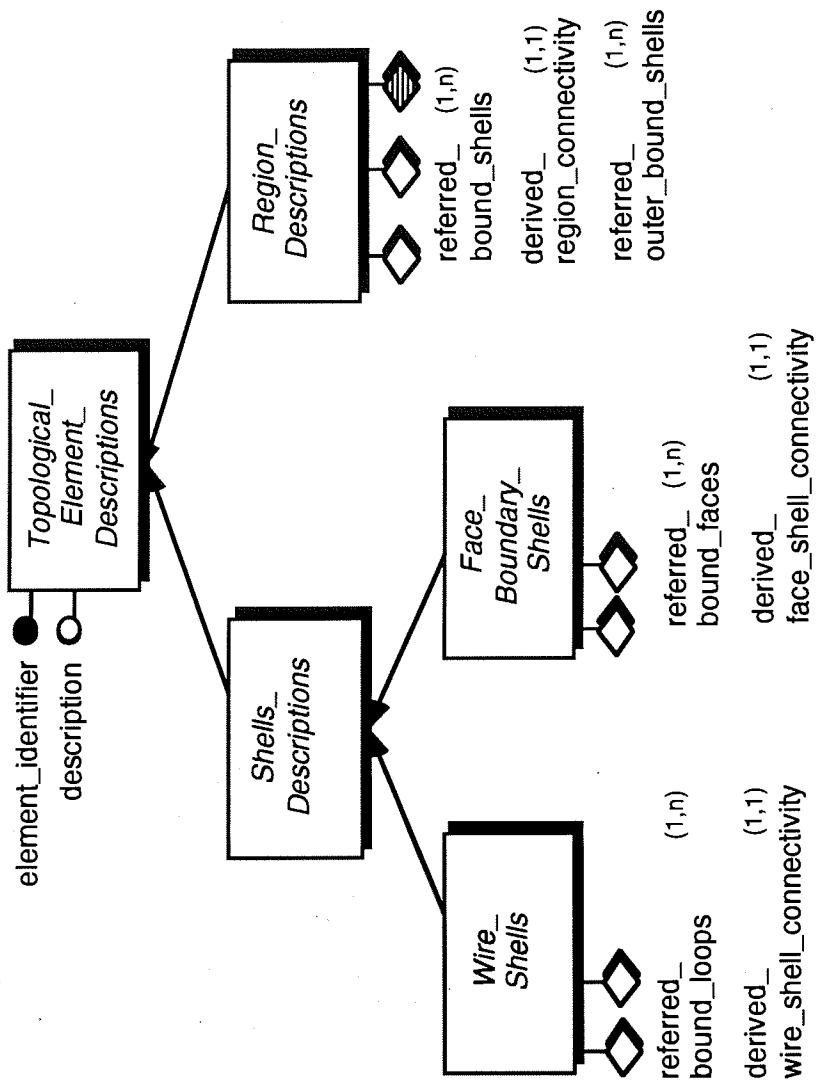
**FIGURE D.4: Primitive Characterization Hierarchies Describing  
TOPOLOGY FORM.**

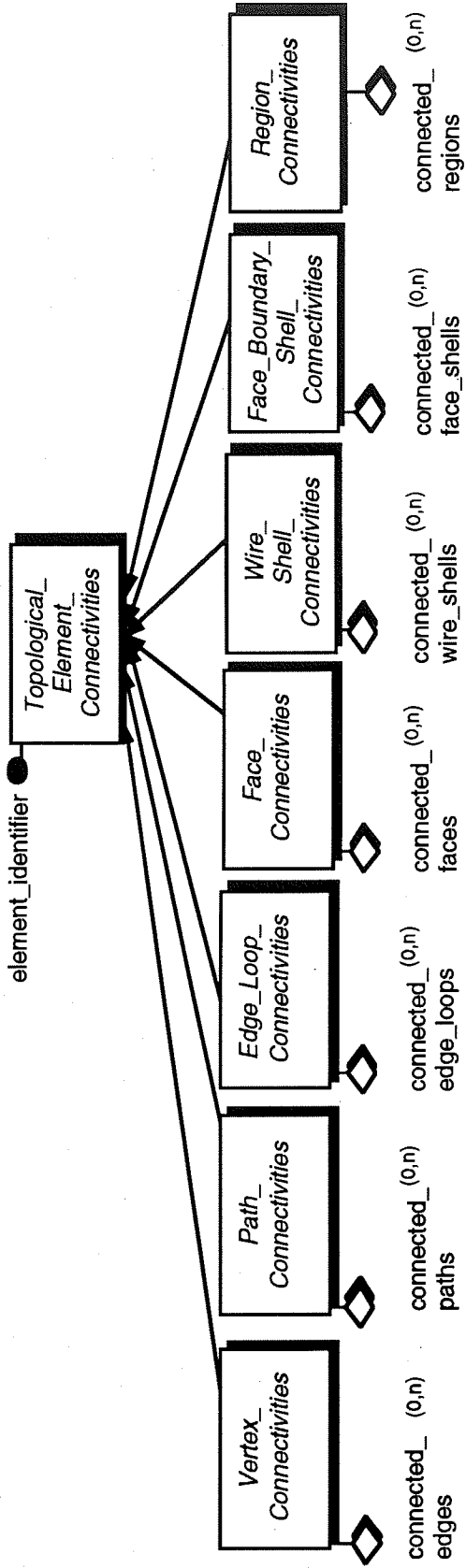


(Continued on next page)

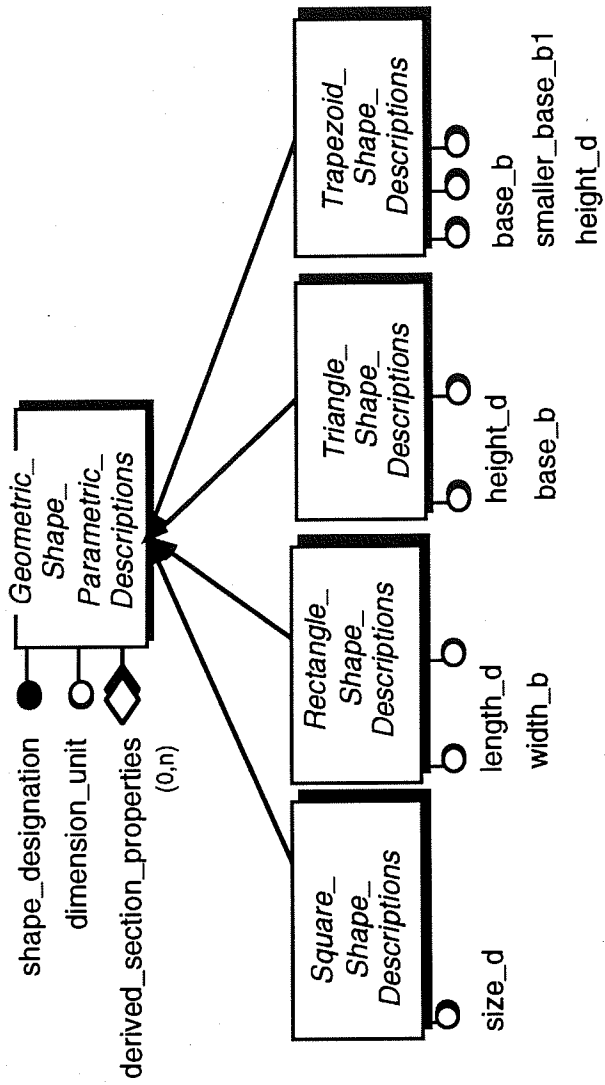


(Continued on next page)

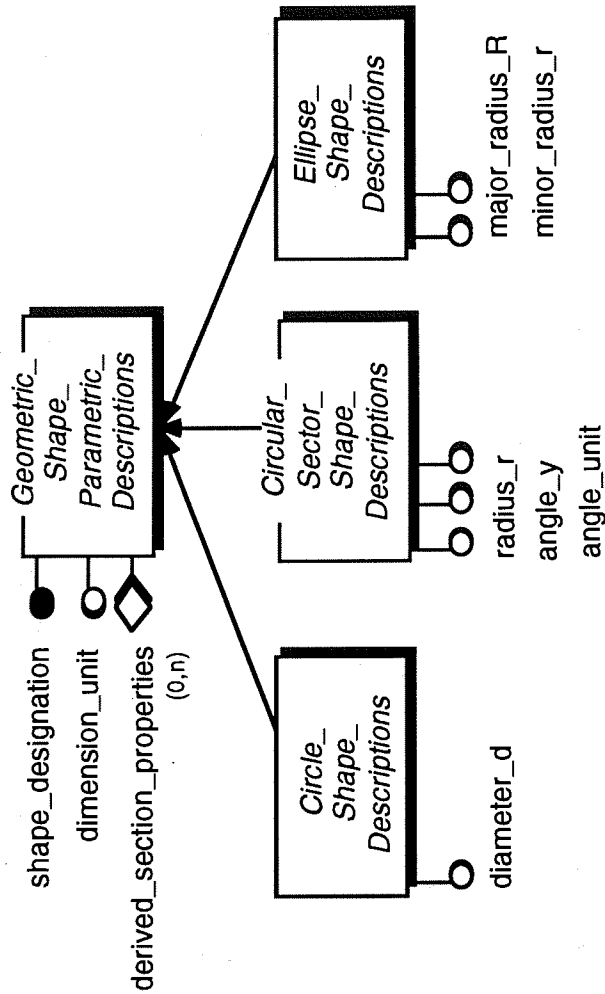




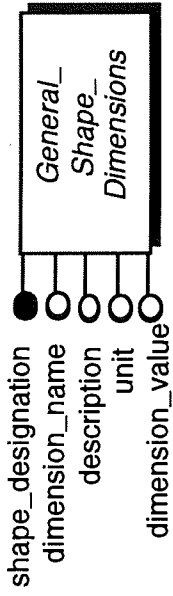
**FIGURE D.5: Primitive Characterization Hierarchies Describing  
SHAPE REPRESENTATION FORM.**

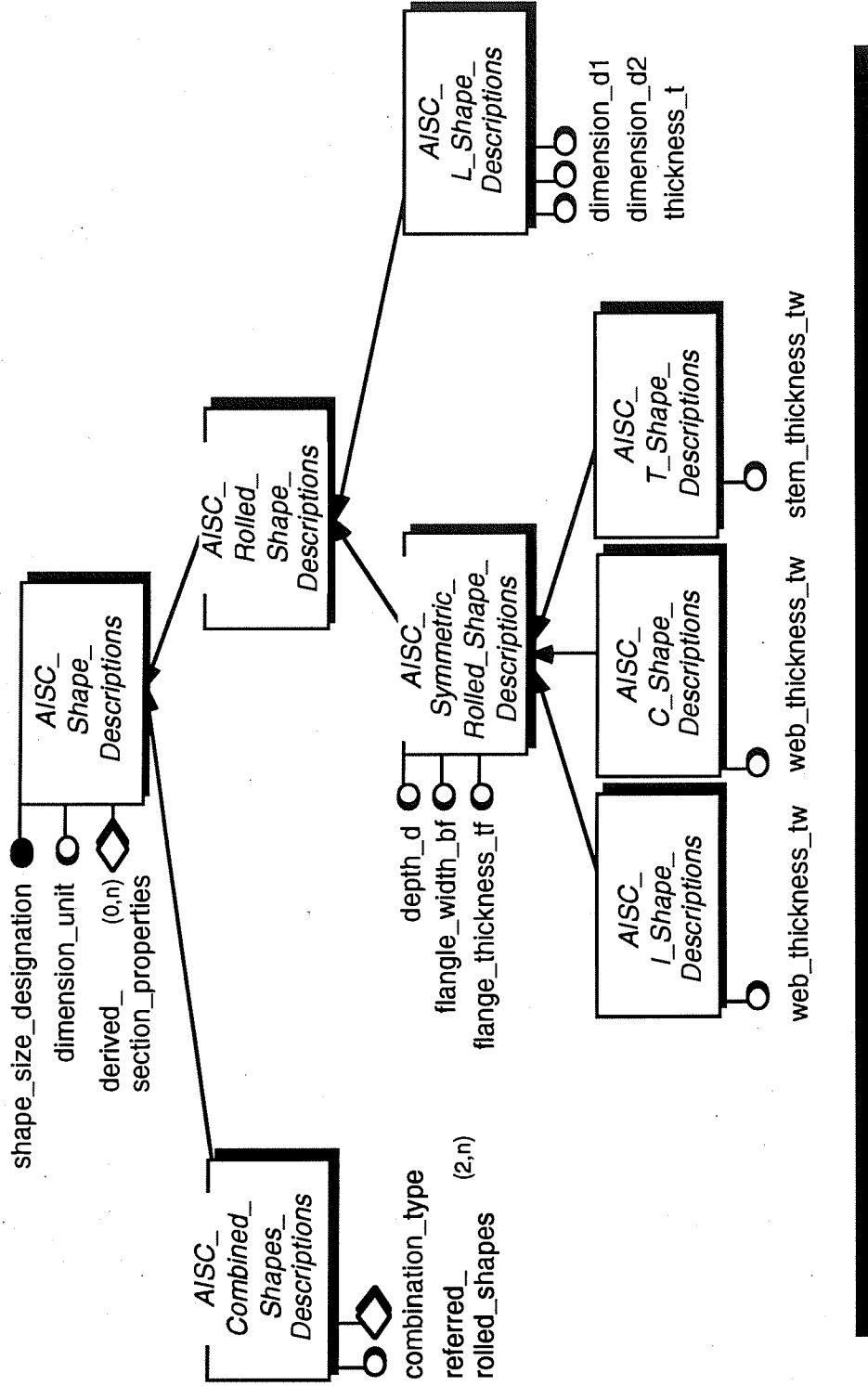


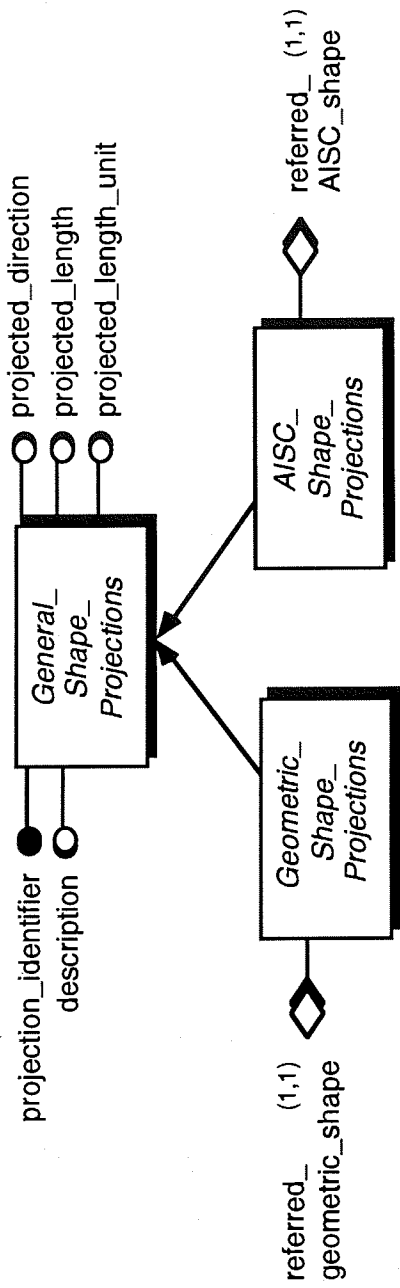
(Continued on next page)

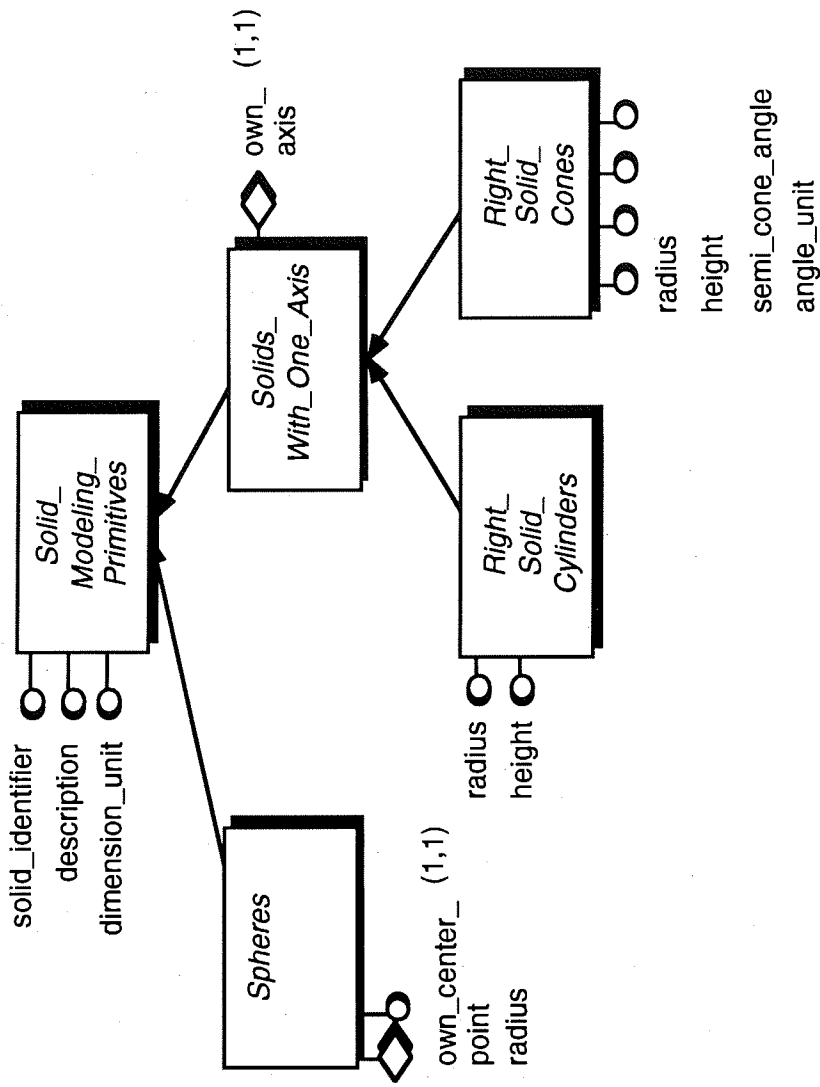




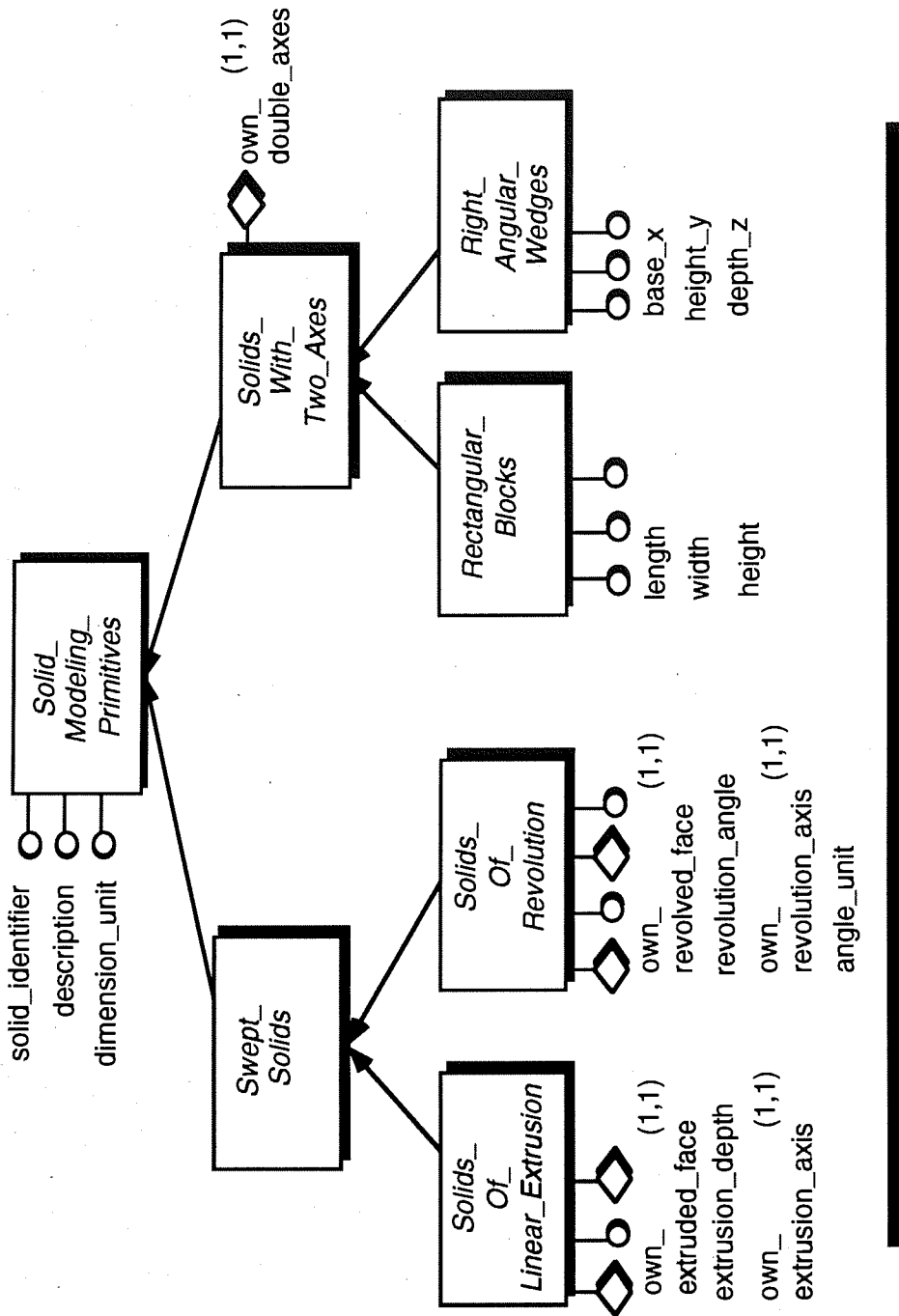


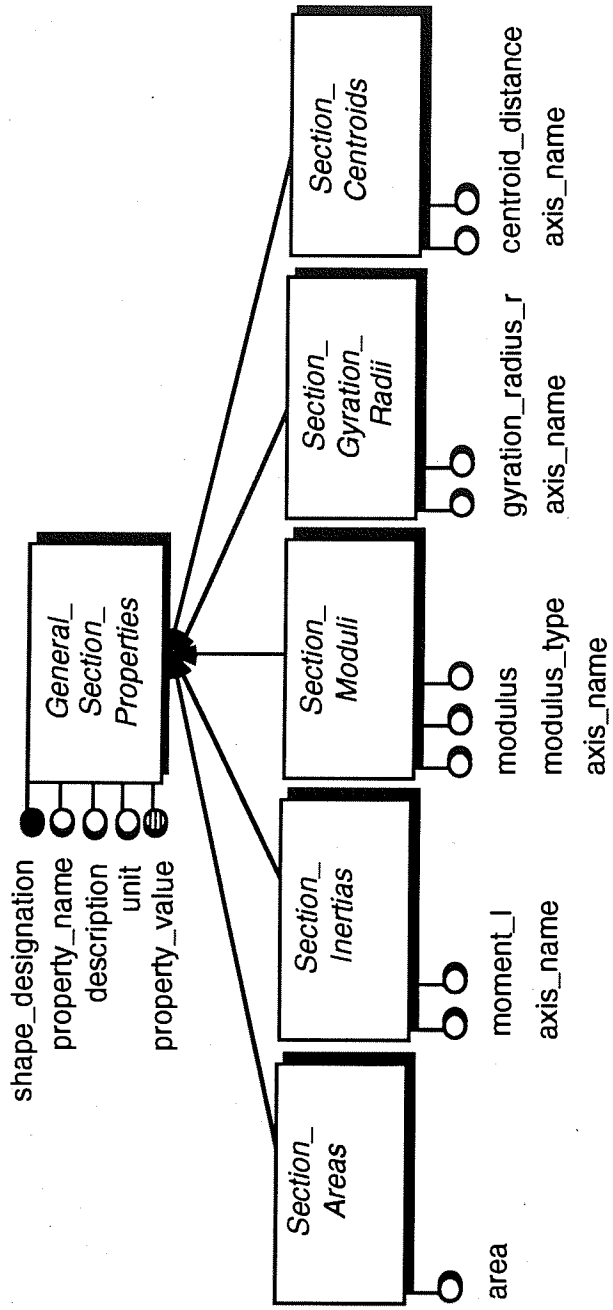




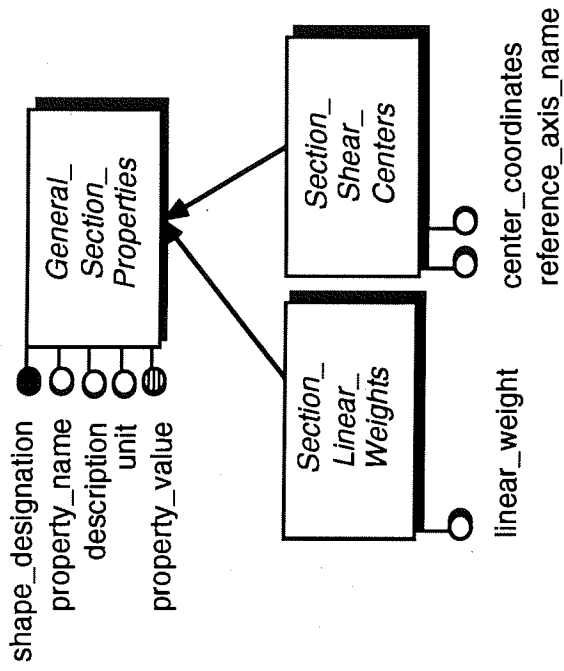


(Continued on next page)



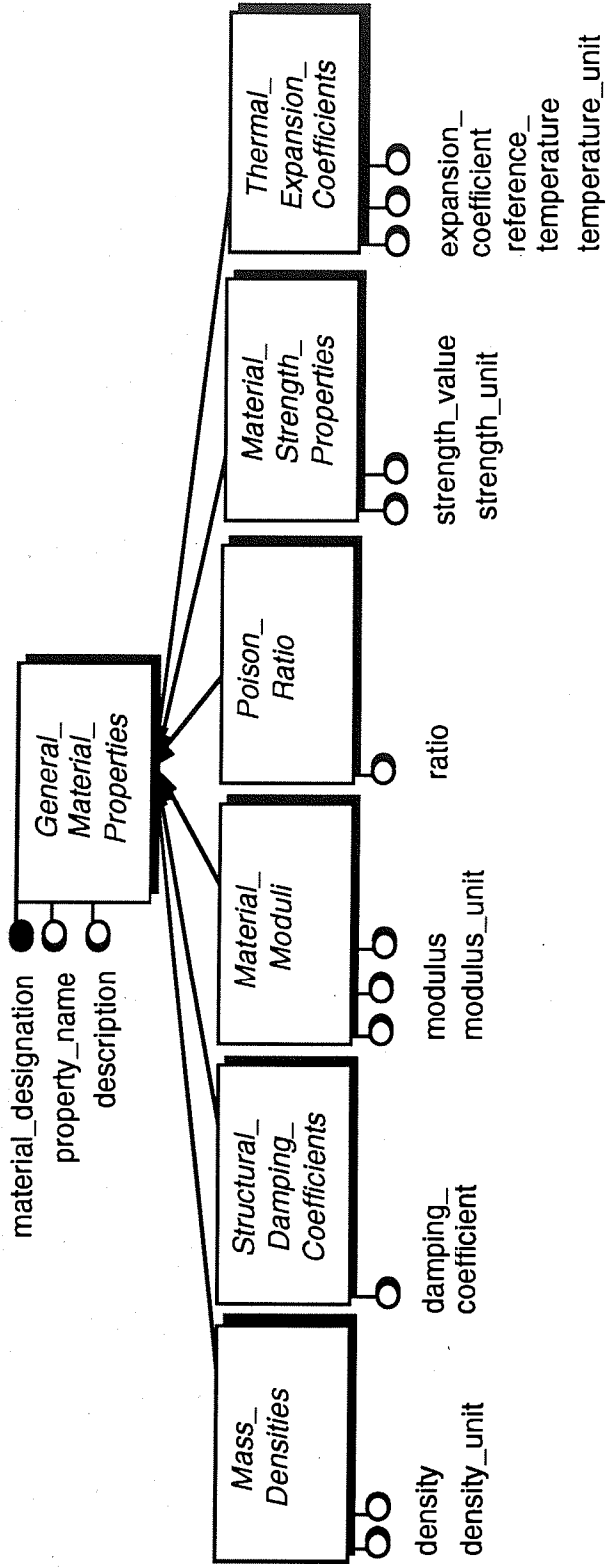


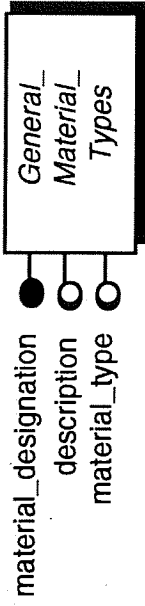
(Continued on next page)



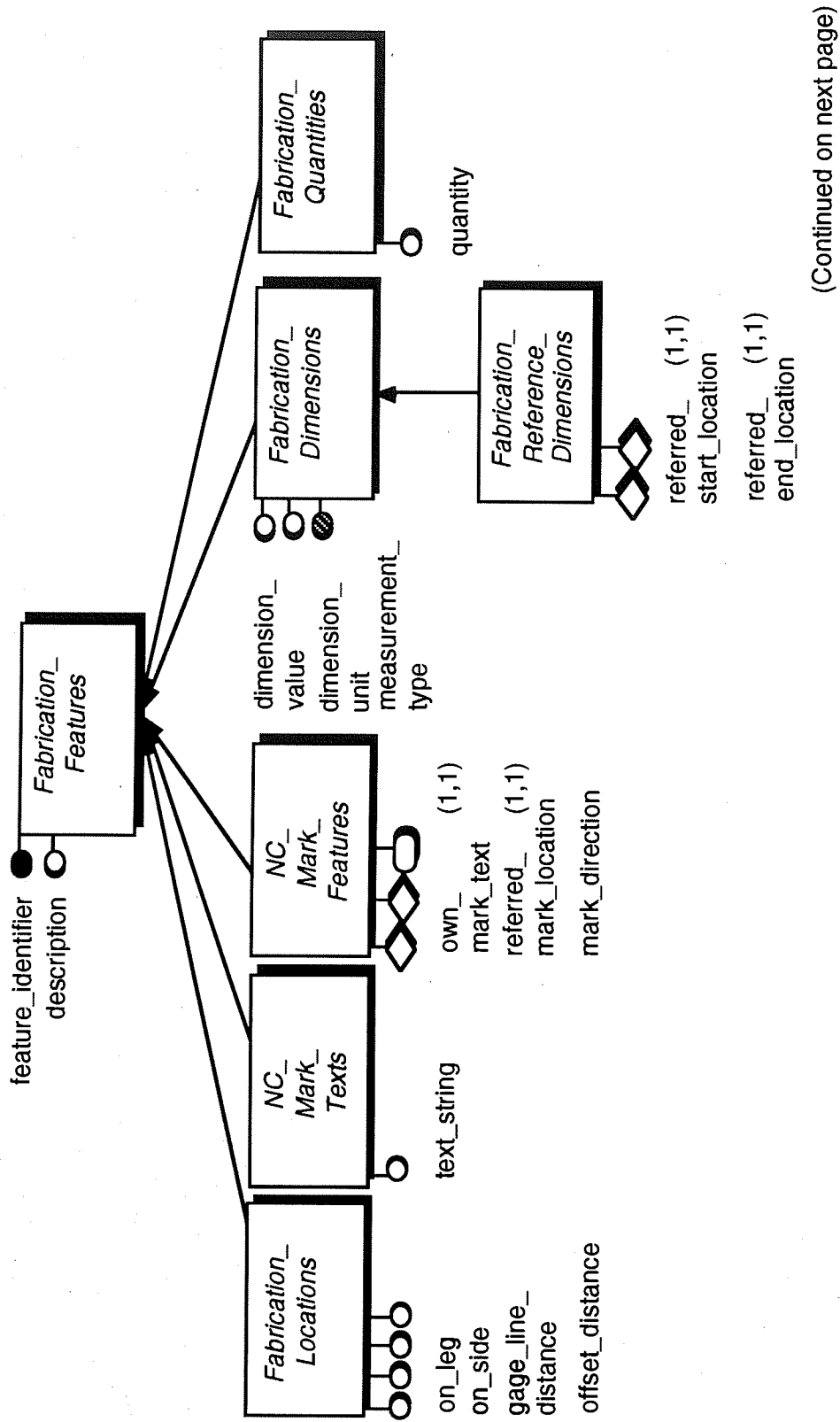
**FIGURE D.6: Primitive Characterization Hierarchies Describing MATERIAL FORM.**



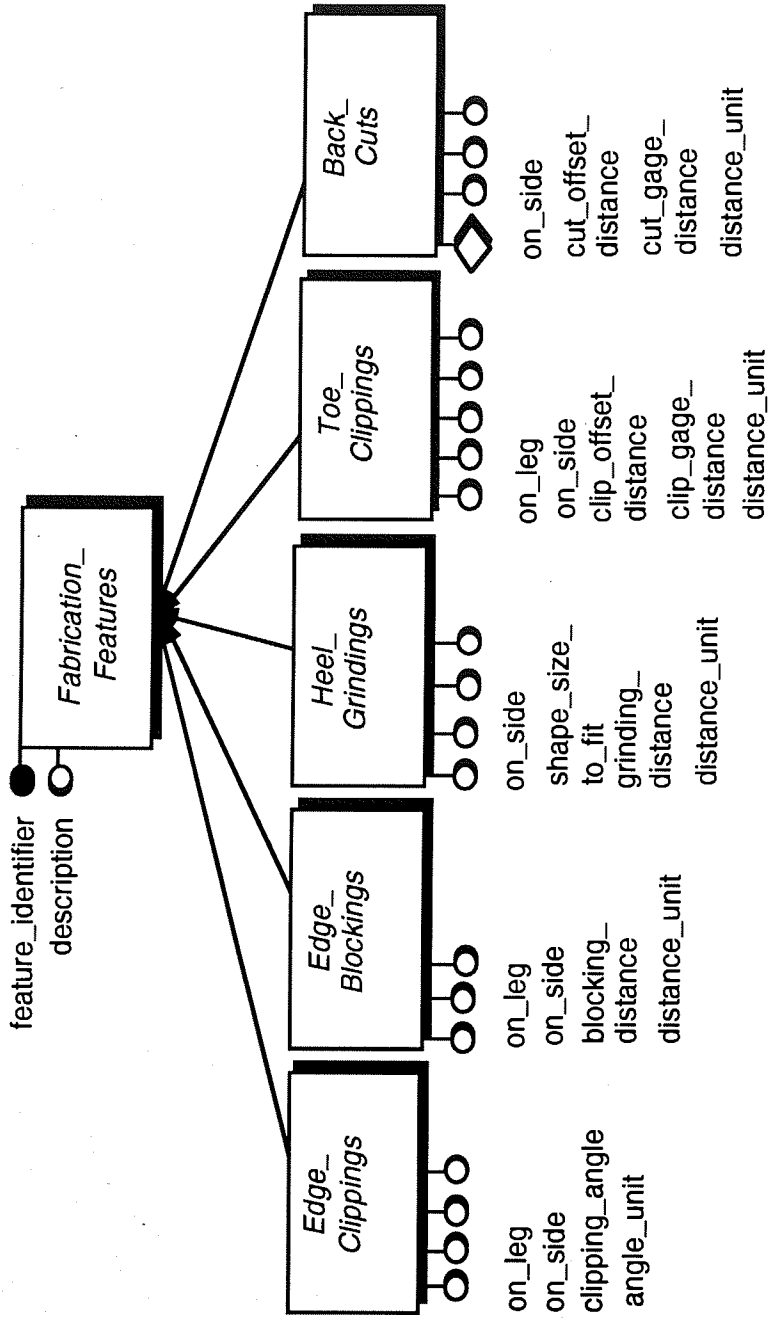




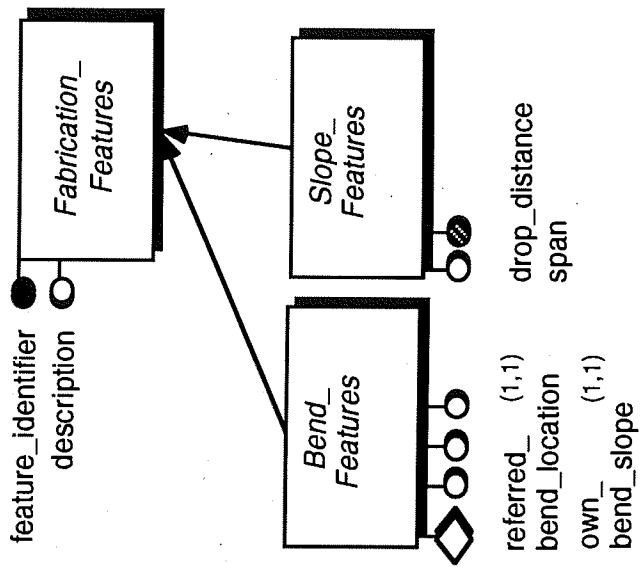
**FIGURE D.7: Primitive Characterization Hierarchies Describing  
PART DETAILING/FABRICATION FORM.**

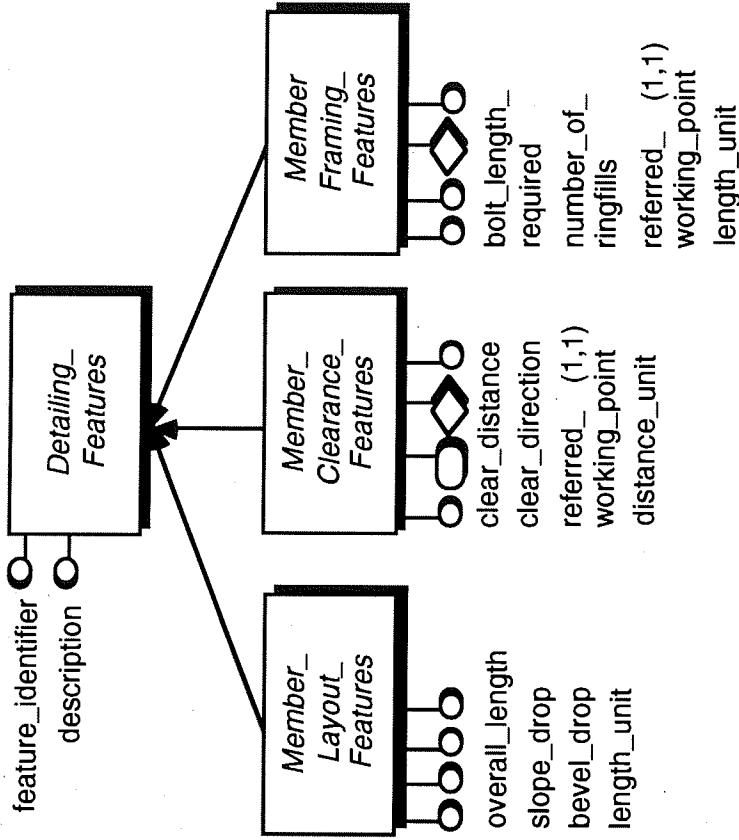


(Continued on next page)



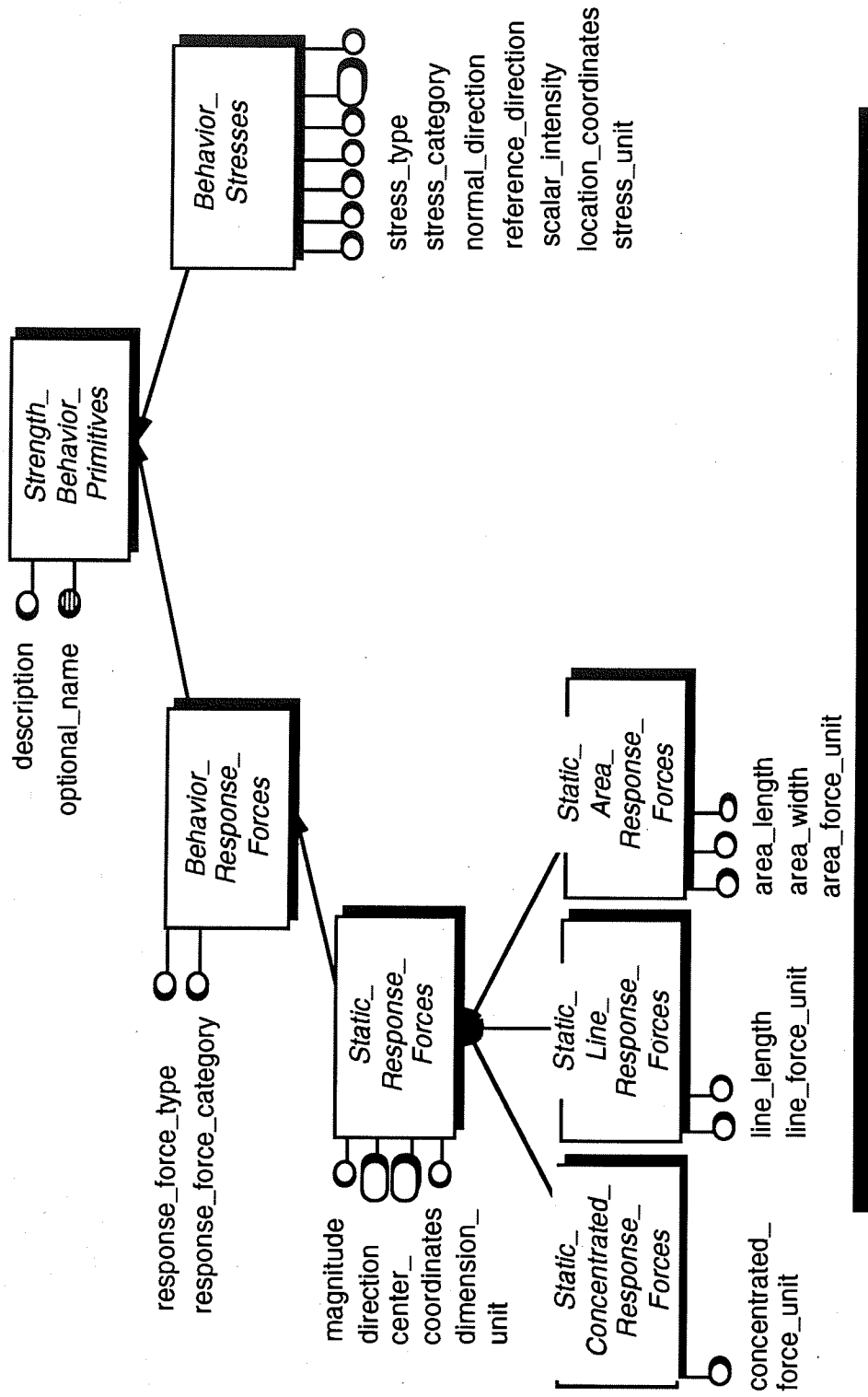
(Continued on next page)

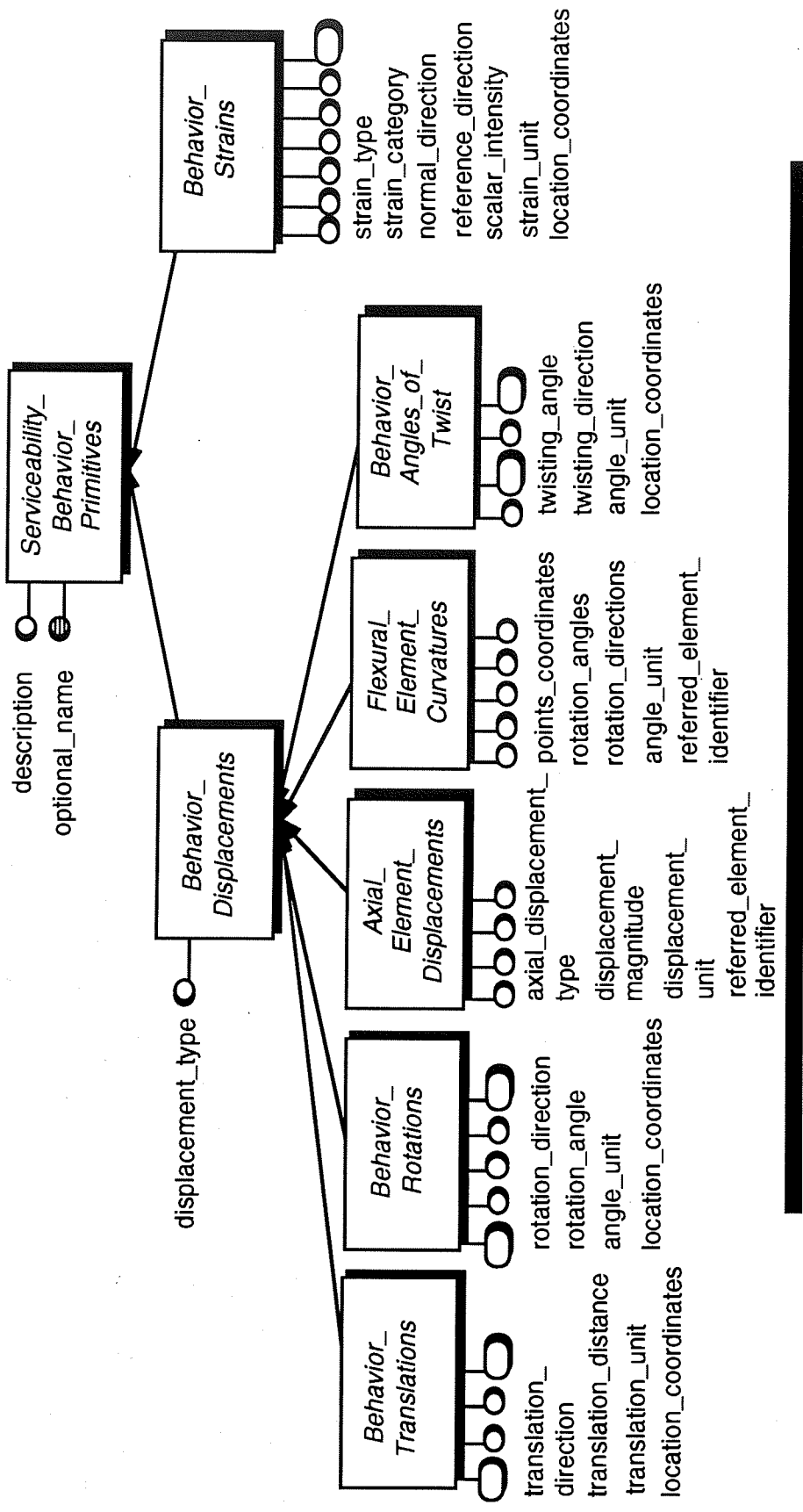


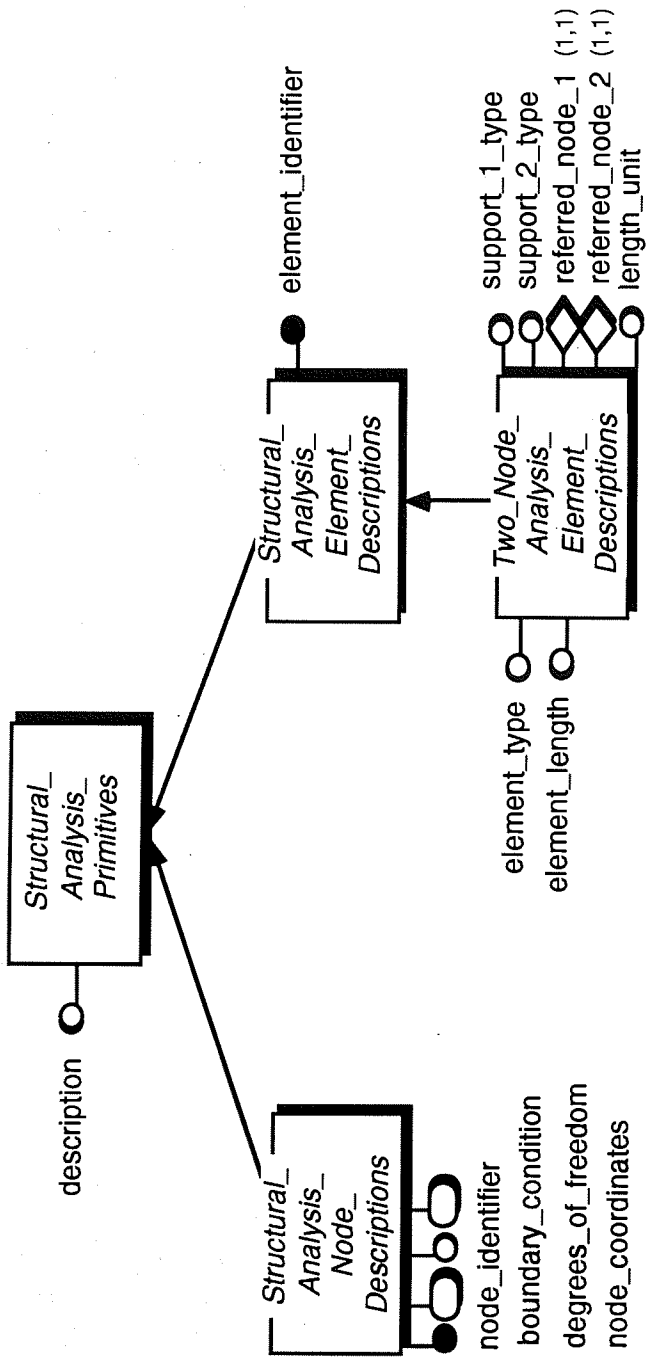


**FIGURE D.8: Primitive Characterization Hierarchies Describing  
STRUCTURAL ENGINEERING BEHAVIOR.**

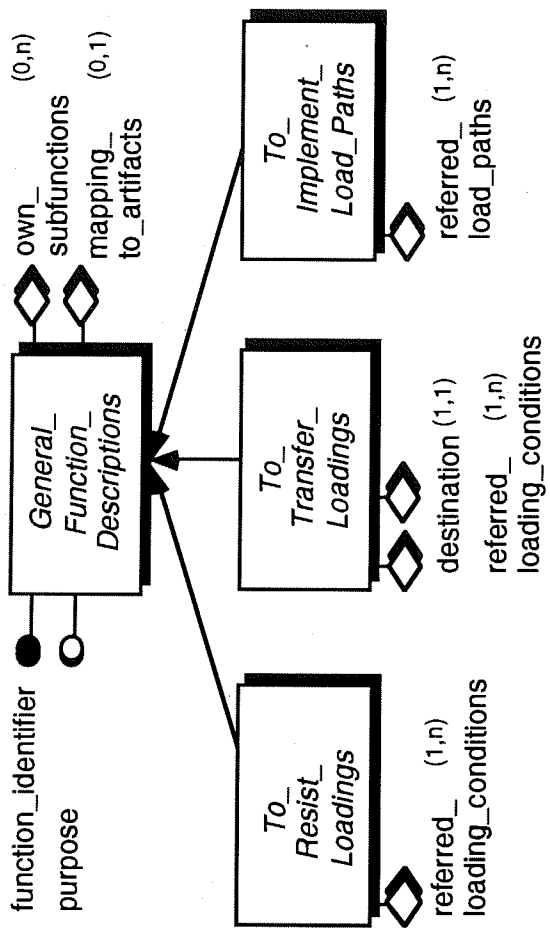




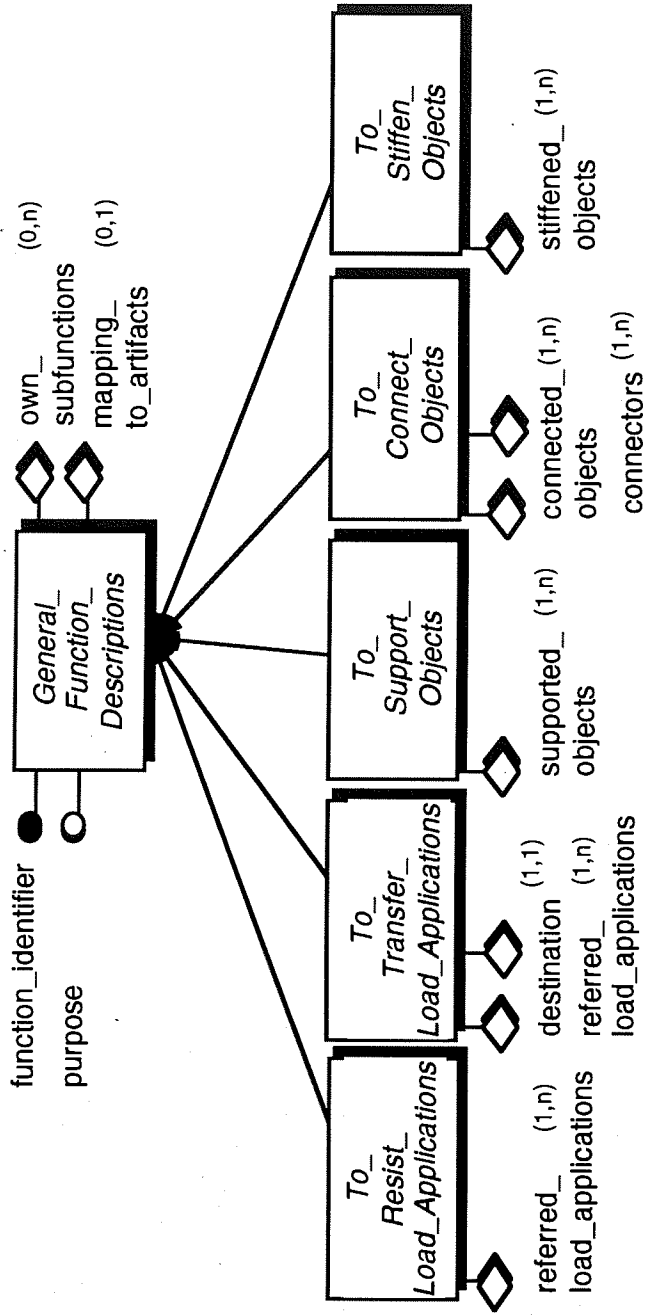




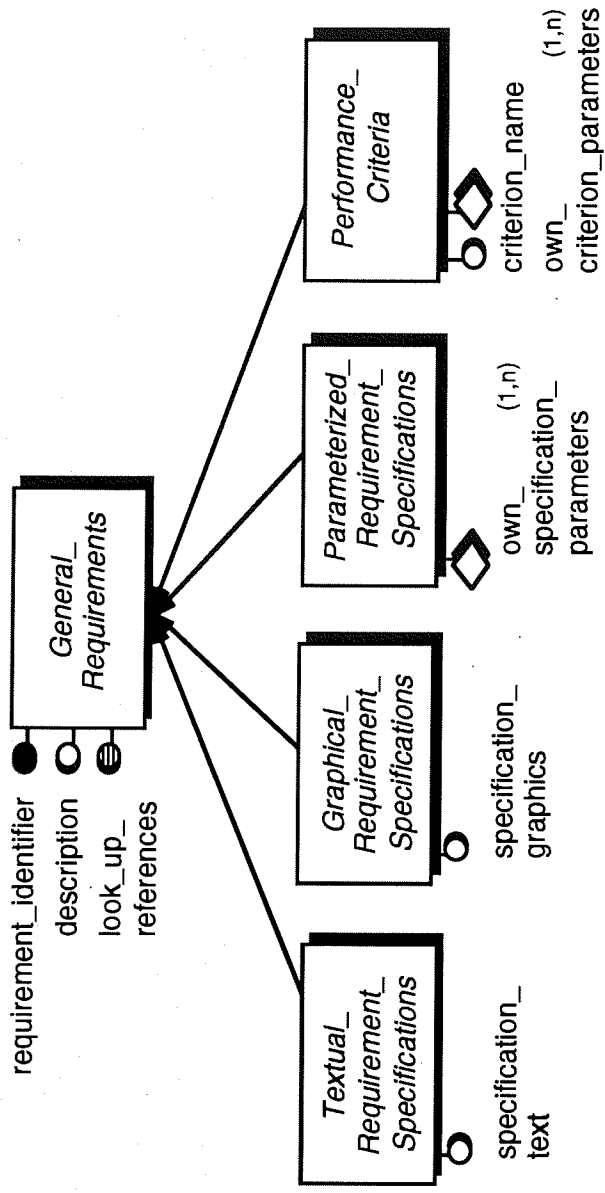
**FIGURE D.9: Primitive Characterization Hierarchies Describing  
STRUCTURAL ENGINEERING FUNCTIONS .**



(Continued on next page)

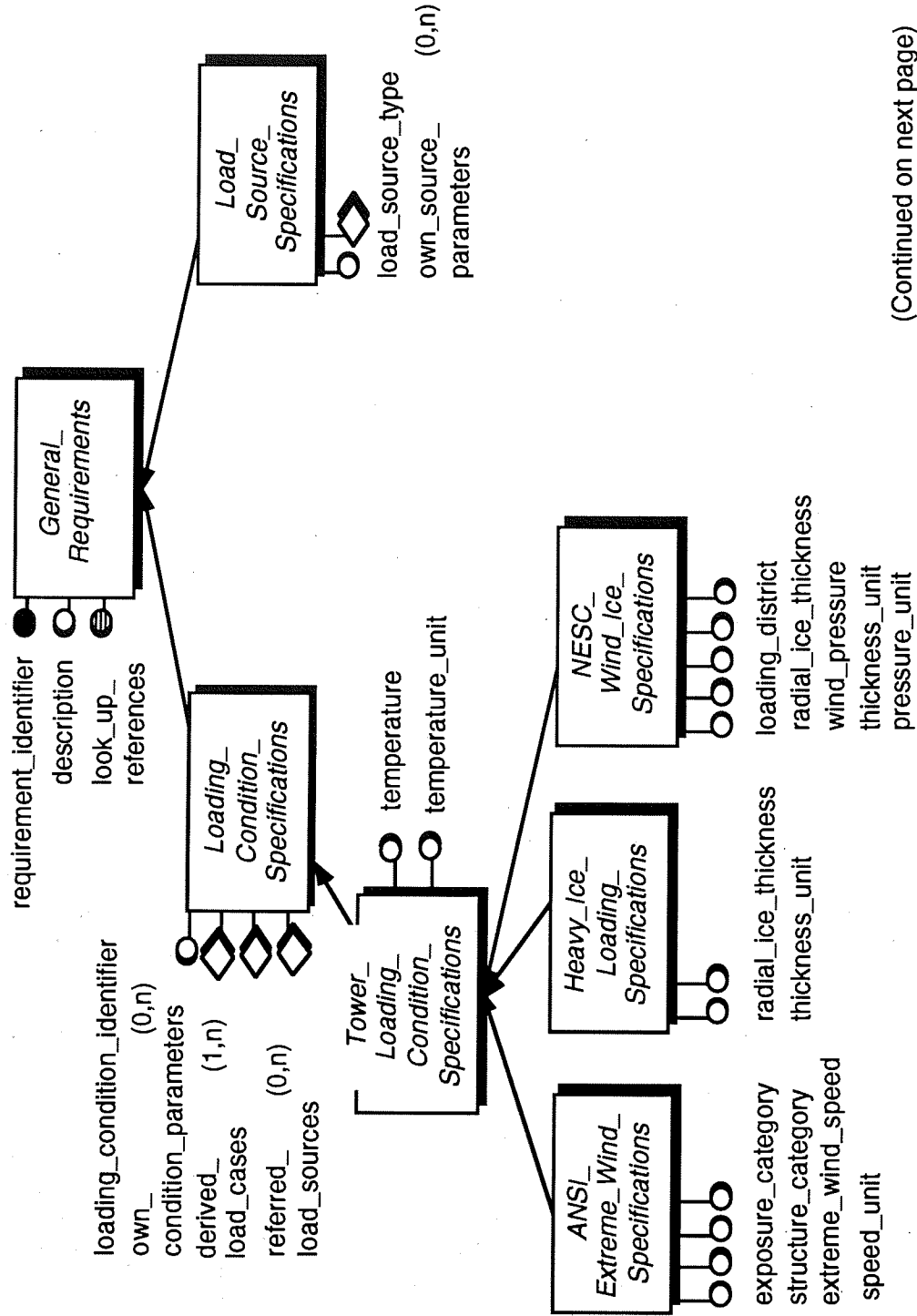


**FIGURE D.10: Primitive Characterization Hierarchies Describing REQUIREMENTS.**

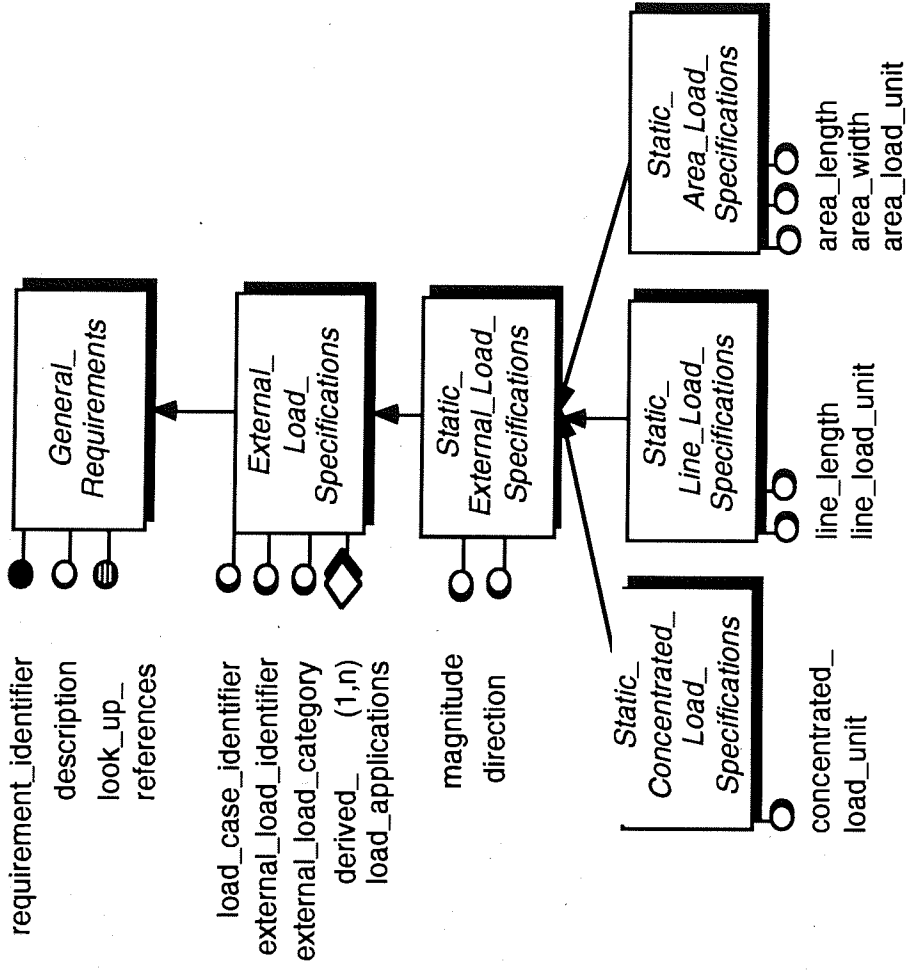


(Continued on next page)

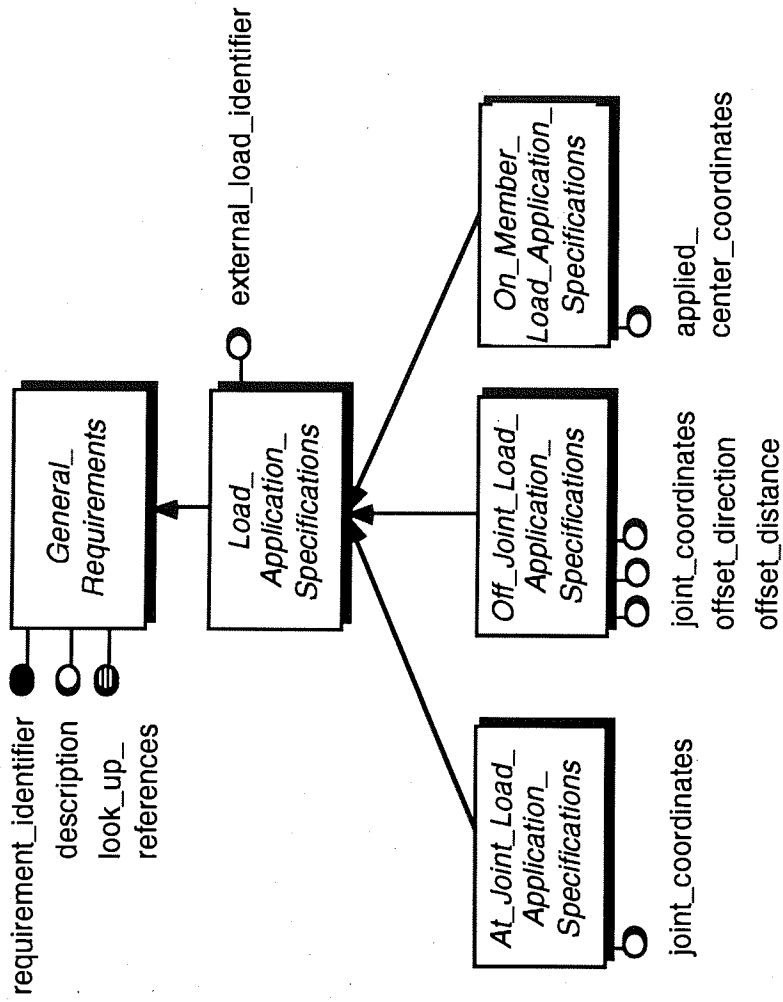




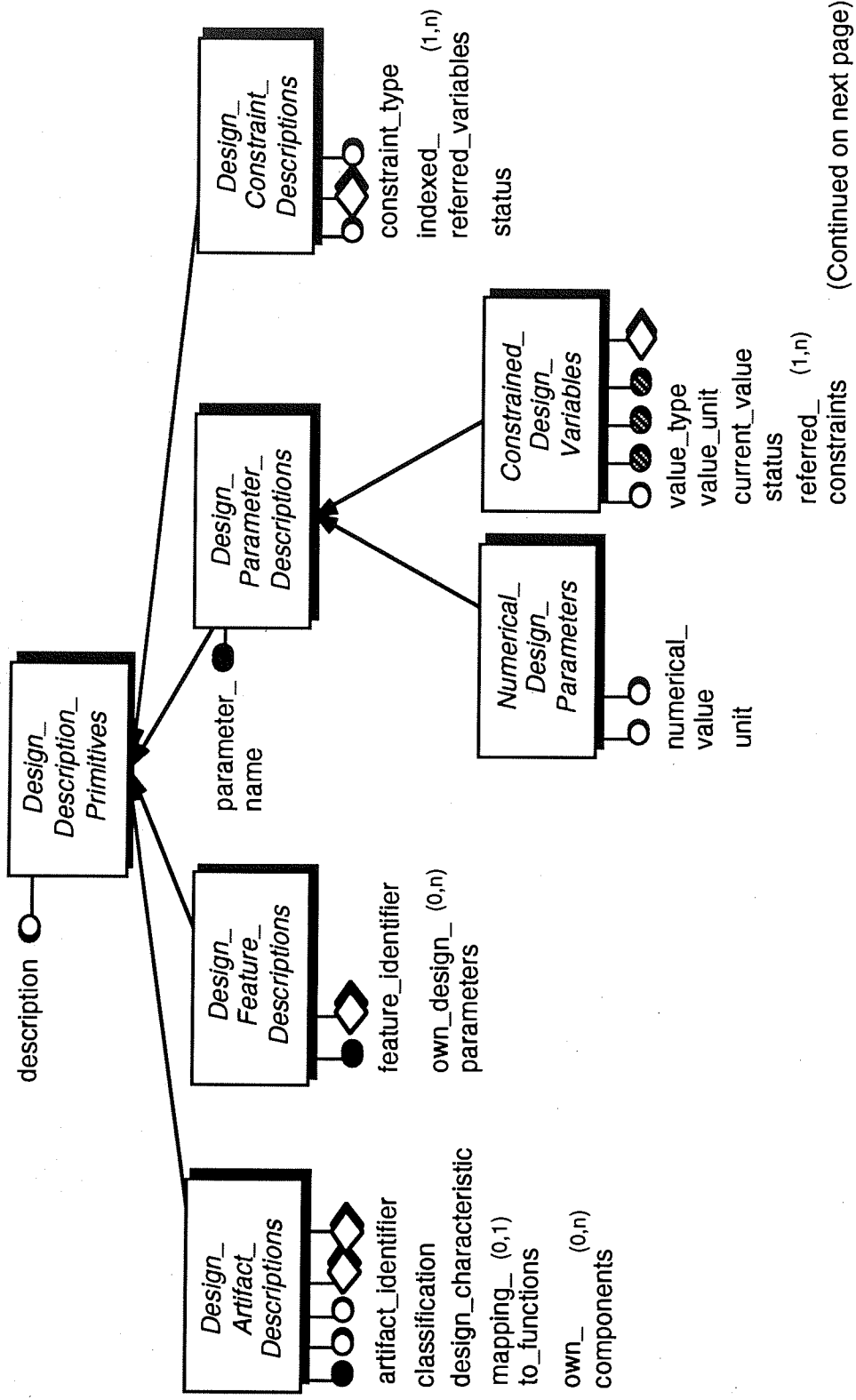
(Continued on next page)



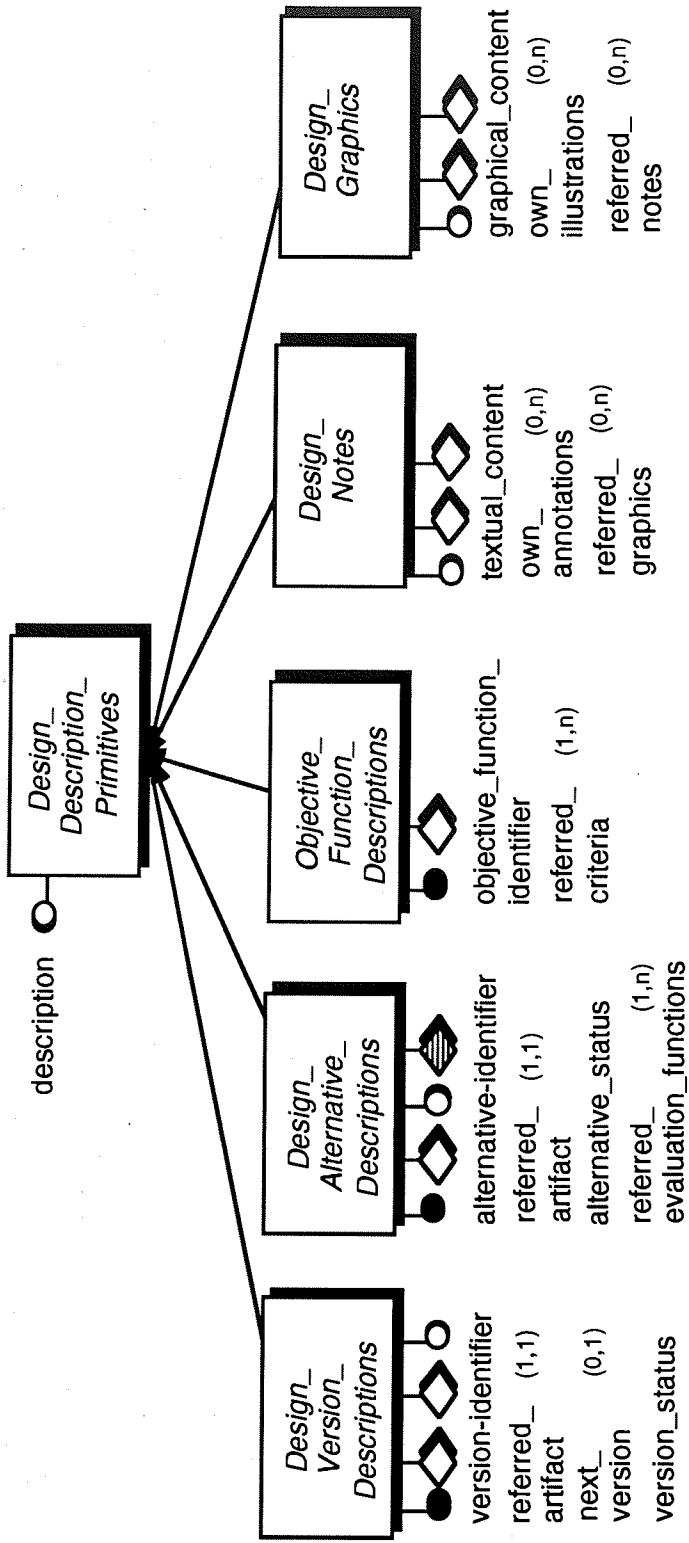
(Continued on next page)

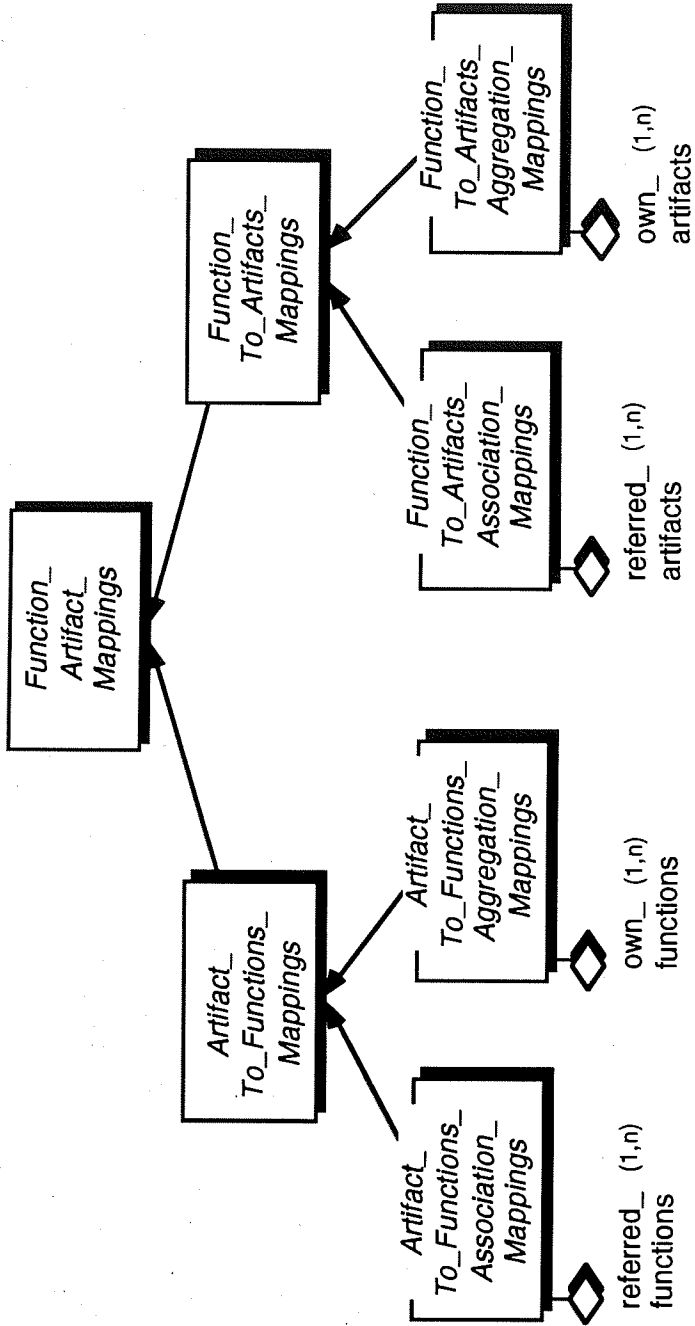


**FIGURE D.11: Primitive Characterization Hierarchies Describing DESIGN.**



(Continued on next page)





# ***Appendix E***

---

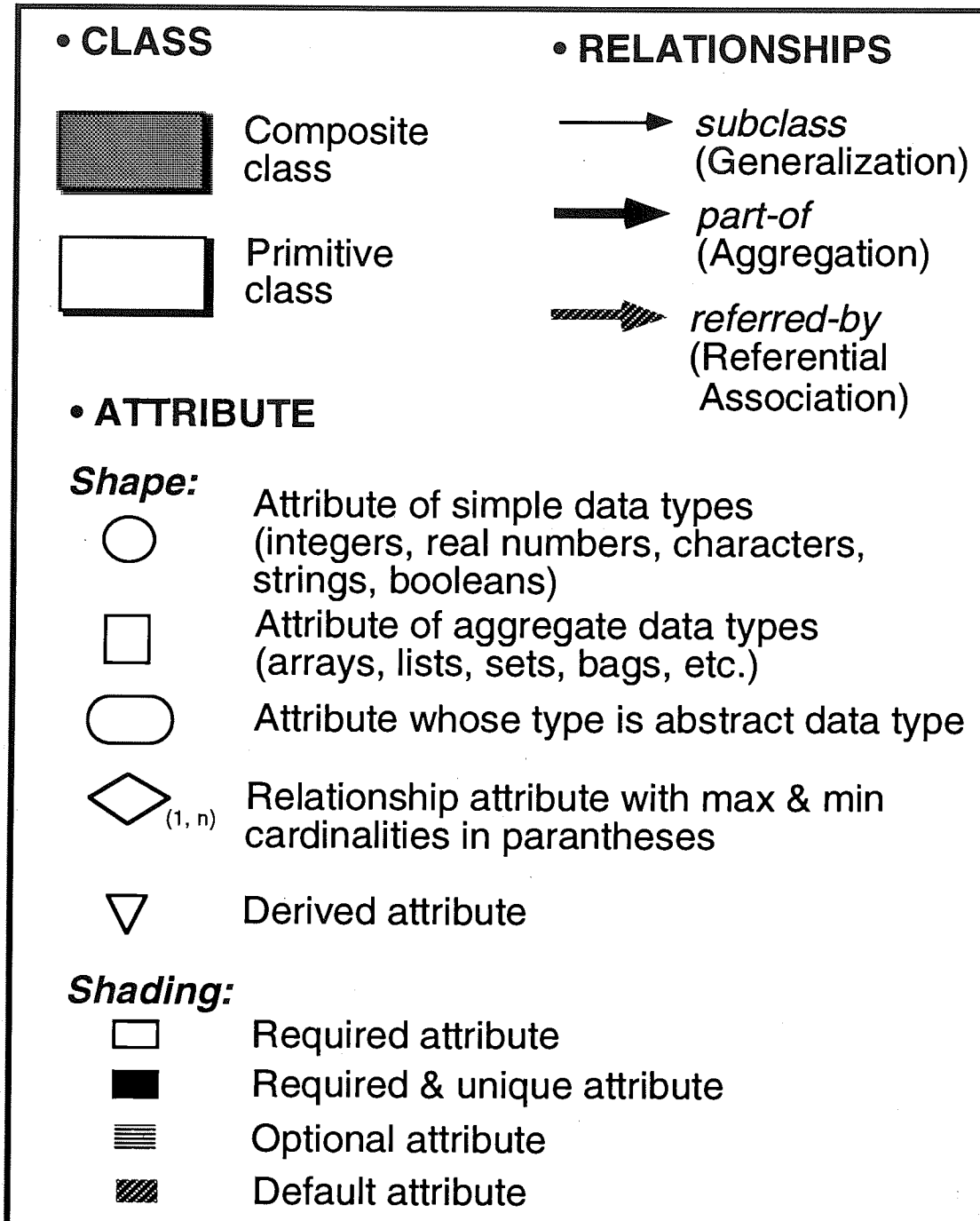
## ***Documentation of Composite Classes Defined in the Testing***

### ***Abstract:***

This appendix contains the documentation of the composite classes that represent different user views of the transmission tower leg members. These user views pertain to the data uses considered in the testing of the P-C Approach. Specifically, this appendix shows the graphical representations of those composite classes. These graphical representations are based on those presented in [Batini 92] for conceptual data modeling. The first figure is the legend for the graphical representations that follow.



## LEGEND



**FIGURE E.1:** Legend for the Graphical Representations of Composite Classes that follow.

***FIGURE E.2: Composite Classes Representing Different User Views  
in the Data Uses Considered in the P-C Approach's Testing.***

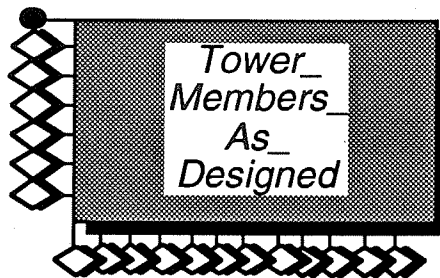


analyzed\_member\_identifier  
 own\_analysis\_elements (1,n)  
 own\_analysis\_nodes (2,n)  
 referred\_section\_area (1,1)  
 referred\_young\_modulus (1,1)  
 referred\_load\_cases (0,n)  
 referred\_external\_loads (0,n)  
 referred\_load\_applications (0,n)

#### DATA USE 1-A:

The structural engineer prepares the input file to the structural analysis program.

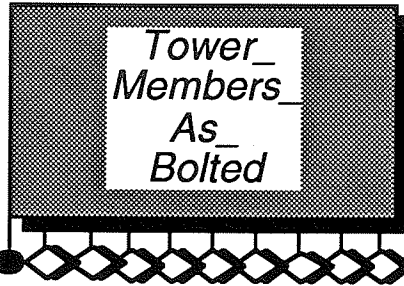
designed\_member\_identifier  
 referred\_shape\_description (1,1)  
 max\_tens\_T (0,1)  
 max\_comp\_C (0,1)  
 max\_shear\_V (0,1)  
 max\_bending\_M (0,1)  
 max\_torque\_To (0,1)



unbraced\_length\_x (1,1)  
 unbraced\_length\_y (1,1)  
 unbraced\_length\_z (1,1)  
 referred\_young\_modulus (1,1)  
 referred\_shear\_modulus (1,1)  
 referred\_yield\_Fy (1,1)  
 referred\_tensile\_Fu (1,1)  
 referred\_design\_artifact (1,1)  
 working\_normal\_stresses (0,5)  
 working\_shear\_stresses (0,5)  
 allowable\_axial\_stress (0,1)  
 allowable\_shear\_stress (0,1)

#### DATA USE 1-B:

The structural engineer designs the members.

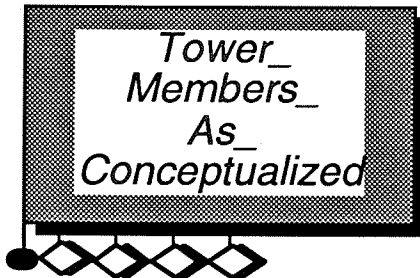


bolted\_member\_identifier  
 referred\_shape\_description (1,1)  
 max\_tens\_T (0,1)  
 max\_comp\_C (0,1)  
 max\_shear\_V (0,1)  
 referred\_analysis\_elements (1,n)  
 referred\_yield\_Fy (1,1)  
 referred\_tensile\_Fu (1,1)  
 bolt\_diameter (1,1)  
 number\_of\_holes\_out (1,1)

**DATA USE 1-C:**

The structural engineer designs the members' bolted end connectors.

---



conceptualized\_member\_identifier  
 own\_design\_artifact (1,1)  
 own\_schematic\_representation (1,1)  
 referred\_shape\_description (1,1)  
 referred\_coordinate\_system (1,1)

**DATA USE 2:**

The structural engineer assumes a preliminary geometry for the tower by arranging tower systems and configuring their members.



redesigned\_member\_identifier  
 member\_already\_designed (1,1)  
 referred\_design\_functions (1,n)  
 referred\_analysis\_elements (1,n)  
 referred\_coordinate\_system (1,1)  
 referred\_fabrication\_features (1,n)

### DATA USE 3:

The structural engineer redesigns the tower leg members to add new electrical equipment to the tower structure.

---



anchored\_member\_identifier  
 referred\_shape\_description (1,1)  
 max\_reactions\_P (1,n)  
 max\_reactions\_V (1,n)  
 max\_reactions\_M (1,n)  
 referred\_analysis\_element (1,1)  
 referred\_yield\_Fy (1,1)  
 bolt\_diameter (0,1)  
 number\_of\_holes (0,1)

### DATA USE 4:

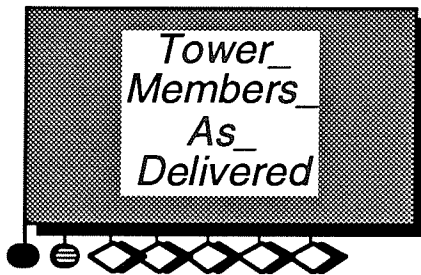
The foundation engineer uses the data of bottom leg members to design the tower anchoring devices.



detailed\_member\_identifier  
 bolt\_diameter (0,1)  
 number\_of\_holes (0,1)  
 referred\_shape\_description (1,1)  
 referred\_member\_layout (1,1)  
 referred\_member\_clearances (0,1)  
 own\_fab\_features (1,n)  
 own\_detail\_notes (0,1)  
 own\_detail\_illustrations (0,n)

**DATA USE 5-A:**

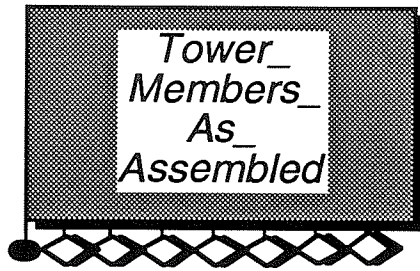
The detailer details the fabrication parts.



delived\_member\_identifier  
 optional\_remarks  
 referred\_NC\_mark (1,1)  
 referred\_fab\_quantity (1,1)  
 referred\_shape\_description (1,1)  
 referred\_fab\_length (1,1)  
 own\_approximate\_weight (0,1)

**DATA USE 5-B:**

The fabricator bundles and ships parts to the construction site.

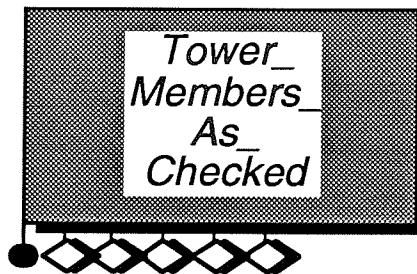


assembled\_member\_identifier  
 referred\_shape\_description (1,1)  
 referred\_NC\_mark (1,1)  
 referred\_fab\_quantity (1,1)  
 referred\_fab\_length (1,1)  
 referred\_bolt\_patterns (1,n)  
 referred\_approximate\_weight (0,1)  
 referred\_schematic\_representation (0,1)

#### DATA USE 5C:

The construction crews assemble the tower panels on grounds.

---



checked\_member\_identifier  
 referred\_shape\_description (1,1)  
 referred\_cartesian\_position (1,1)  
 referred\_cartesian\_orientation (1,1)  
 referred\_member\_layout (1,1)  
 own\_spatial\_envelope (1,1)

#### DATA USE 6:

The electrical engineer checks the leg members for electrical clearances.