

# Representing Systems through Object-Process Methodology and Axiomatic Design

by

**Nathan R. Soderborg**

Ph.D. Mathematics, University of Michigan, 1991  
B.S. Mathematics, Brigham Young University, 1986

Submitted to the System Design and Management Program  
in Partial Fulfillment of the Requirements for the Degree of

**Master of Science in Engineering and Management**

at the

Massachusetts Institute of Technology

February 2002

© 2002 Nathan R. Soderborg. All rights reserved

The author hereby grants to MIT permission to reproduce and to  
distribute publicly paper and electronic copies of this thesis document in whole or in part.

Signature of Author \_\_\_\_\_

Nathan R. Soderborg  
System Design and Management Fellow

Certified by \_\_\_\_\_

Dov Dori  
Thesis Supervisor  
Professor, Information Systems Engineering, Technion, Israel Institute of Technology  
MIT Research Affiliate

Certified by \_\_\_\_\_

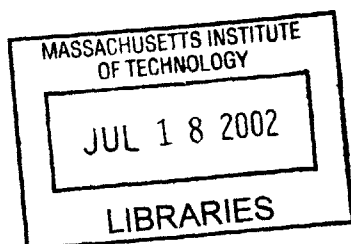
Edward F. Crawley  
Thesis Supervisor  
Mar/Vicar Faculty Fellow  
Professor and Department Head of Aeronautics & Astronautics

Accepted by \_\_\_\_\_

Steven D. Eppinger  
Co-Director, LFM/SDM  
QM LFM Professor of Management Science and Engineering Systems

Accepted by \_\_\_\_\_

Paul A. Lagace  
Co-Director, LFM/SDM  
Professor of Aeronautics & Astronautics and Engineering Systems



**BARKER**

# **Representing Systems through Object-Process Methodology and Axiomatic Design**

by

**Nathan R. Soderborg**

Submitted to the System Design and Management Program  
on January 11, 2001, in Partial Fulfillment of the  
Requirements for the Degree of Master of Science in  
Engineering and Management

## **ABSTRACT**

Object-Process Methodology and Axiomatic Design are presented as two fundamentally different methods for representing systems. Strengths of the two methods are discussed and synergies are identified. The methods are shown to be complementary. When applied together as an integrated framework, they provide a system architect descriptive and evaluative capability unavailable from either methodology alone.

The descriptive capabilities and definitional framework of Object-Process Methodology is used to improve formulation of Functional Requirements and Design Parameters in Axiomatic Design. Examples demonstrate that adequate descriptions of both function and architecture require a *combination* of objects and processes. Object-Process Methodology templates for describing function and architecture using such combinations are presented. Adherence to Axiomatic Design's Independence Axiom is evaluated through patterns identified in Object-Process Diagrams.

Thesis Supervisor: Edward F. Crawley

Title: Professor and Department Head of Aeronautics & Astronautics

Thesis Supervisor: Dov Dori

Title: MIT Research Affiliate, Professor, Information Systems Engineering, Technion,  
Israel Institute of Technology

# TABLE OF CONTENTS

<b>ACKNOWLEDGMENTS .....</b>	<b>4</b>
<b>1. Introduction.....</b>	<b>5</b>
1.1 Object-Process Methodology .....	5
1.2 Axiomatic Design .....	12
1.3 Connecting OPM and Axiomatic Design .....	15
<b>2. Representing Function and Architecture through OPM.....</b>	<b>17</b>
2.1 The WHATs and HOWs of Design .....	17
2.2 What is Function? .....	23
2.3 What is Architecture?.....	36
2.4 Function vs. Architecture.....	39
<b>3. Documenting Requirements, Design Parameters, and Intent in OPM.....</b>	<b>53</b>
3.1 Formulating Good Requirements .....	53
3.2 Functional Requirements .....	56
3.3 Constraints .....	66
3.4 Design Parameters .....	69
3.5 Preserving Intent.....	74
<b>4. Evaluating Independence via OPM.....</b>	<b>83</b>
4.1 Freezer Door Example.....	83
4.2 Water Faucet Example .....	88
4.3 Generalization of Coupling in OPDs.....	99
4.4 Conclusion.....	102
<b>BIBLIOGRAPHY .....</b>	<b>104</b>
<b>BIOGRAPHICAL NOTE .....</b>	<b>106</b>

## **ACKNOWLEDGMENTS**

*Dedicated to Sondra,  
for love, support, and vision.*

*Thanks to Professors Dov Dori and Ed Crawley,  
for teaching me through discussion and the exchange of ideas.*

## 1. Introduction

Representing systems well is an important task *and* tool for system architects. A good representation not only communicates what the system is and how it operates, it helps the architect develop the system, providing a means for organizing elements, understanding functional relationships, identifying critical interfaces, and guiding improvement. Many methods for representing systems are available, but each has its own particular strengths and weaknesses.

Two methods that have gained some prominence during the past decade are Object-Process Methodology and Axiomatic Design. Both provide useful representations of systems, but in fundamentally different ways. Object-Process Methodology is a *descriptive* method. It represents systems through visual diagrams and textual descriptions. Axiomatic Design is an *evaluative* method. It represents systems through matrices that depict the presence of important functional relationships. Its axioms provide the basis for judging whether or not a design is "good."

This thesis is the first work to extensively examine the relationship between Object-Process Methodology and Axiomatic Design. It demonstrates that they are complementary design methodologies. When applied together as an integrated framework, they provide a system architect descriptive and evaluative capability unavailable from either methodology alone.

### 1.1 Object-Process Methodology

Professor Dov Dori has described Object-Process Methodology (hereafter referred to as OPM) as "a system development methodology that integrates function,

structure and behavior in one model." [Do1] OPM provides methods for representing systems both graphically and textually. Graphical representations, known as Object-Process Diagrams (OPDs), convey complex interconnections and non-linear relationships according to established standards with brevity and clarity. Textual representations, composed in Object-Process Language (OPL), provide a corresponding English language script that expresses the contents of each OPD verbally. Together, a set of OPDs and the corresponding OPL script, specify a system. By allowing a system architect to articulate system function and contents as complementary visual and verbal expressions, OPM helps manage system complexity and reduce complicatedness for both designers and implementers.

OPM is a radical departure from the Object-Oriented approach that has been the prevailing paradigm in software system development for the past 10-20 years. OPM recognizes *processes* as stand-alone entities in addition to *objects*. Dori states: "The basic premise of OPM is that *objects and processes are two types of equally important classes of things*. Together, objects and processes faithfully describe the system's structure, function and behavior in a single, coherent model, in virtually any domain." [Do2, 1.2, p.7]

The elements of OPDs fall into three categories: entities, procedural links, and structural relations. Dori has summarized the elements of OPD in the following tables (reprinted here with permission).

Entities are objects (symbolized by rectangles), processes (ellipses), and states (rounded-corner rectangles within objects).

### OPM Entities: The Building Blocks

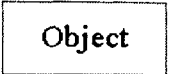
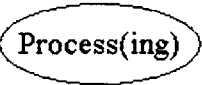
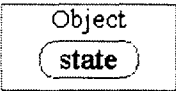
	Visual Representation	Textual Form	Definition	Description
Entities		Nouns; first letter in every word is capitalized	<i>An <b>object</b> is a thing that has the potential of stable, unconditional physical or mental existence.</i>	Objects are static things, which can be generated, changed, or consumed only by processes.
		Nouns in gerund form; first letter in every word is capitalized	<i>A <b>process</b> is a pattern of transformation that an object undergoes.</i>	Processes are dynamic things. They generate, change or consume objects.
		Nouns, adjectives or adverbs; non-capitalized	<i>A <b>state</b> is a situation an object can be at.</i>	An object is at some state. A process can change an object's state.

Table 1.1: OPM Entities [Do2]

Various directed lines that connect processes to objects represent procedural links. These links express transformations arising within or from the system and may be made possible by process enablers.

**Procedural Links: Connect Objects to Processes**


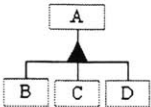

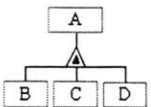

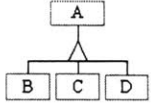

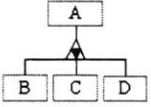
Name	Symbol	OPD	OPL	Description
<b>Consumption</b>	→		<b>Processing consumes Object.</b>	Process uses object up entirely during its occurrence.
<b>Result</b>			<b>Processing yields Object.</b>	Process creates an entirely new object during its occurrence.
<b>Input &amp; Output</b>			<b>Processing changes Object from input state to output state.</b>	The object is at input state prior to the process occurrence, and at output state as a result of its occurrence.
<b>Effect</b>	↔		<b>Processing affects Object.</b>	Process changes the state of the object in an unspecified manner.
<b>Agent</b>	—●		<b>Object handles Processing.</b>	Object is a human that is not changed by the process; process needs the agent object in order to occur.
<b>Instrument</b>	—○		<b>Processing requires Object.</b>	Object is a non-human that is not changed by the process; process needs the instrument object in order to occur.
<b>Invocation</b>	→▷		<b>X Processing invokes Y Processing.</b>	First process directly starts up a second process, without an intermediate object.

**Table 1.2: OPM Procedural Links [Do2]**



A set of triangular symbols represents fundamental structural relations.

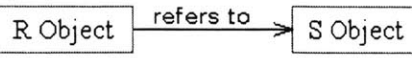
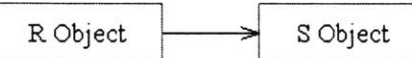


**Fundamental Structural Relations: Reveal Entity Structure**

<b>Full Name</b> (Shorthand Name in <b>bold</b> )	<b>Symbol</b>	<b>OPD</b>	<b>OPL</b>	<b>Description</b>
<b>Aggregation-Participation</b>			<b>A consists of B, C, and D.</b>	B, C and D are parts of the whole A.
<b>Exhibition-Characterization</b>			<b>A exhibits B, C, and D.</b>	B, C and D are attributes of A. (If B is a process, it is an operation of A.)
<b>Generalization-Specialization</b>			<b>B, C, and D are As.</b>	B, C and D are types of A.
<b>Classification-Instantiation</b>			<b>B, C, and D are instances of A.</b>	B, C and D are unique objects of the class A.

**Table 1.3: OPM Structural Relations [Do2]**

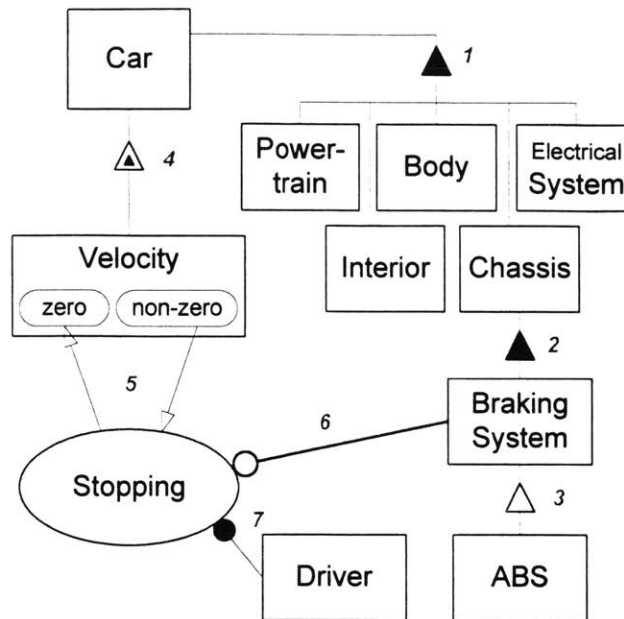
General structural relations are denoted by tagged structural links customizable to the specific system.

**Tagged Structural Links: Typically Link Objects, but May also Link Processes**

Name	Symbol/OPD	OPL	Description
<b>Tagged</b>		<b>R Object refers to S Object.</b>	Relation from source object to destination object; relation name is entered by architect, and is recorded along link.
<b>(Null)</b>		<b>R Object relates to S Object.</b>	Relation from source object to destination object with no tag.
<b>Bi-directional Tagged</b>		<b>R Object precedes S Object. S Object follows R Object.</b>	Relation between two objects; relation names are entered by architect, and are recorded along link.
<b>(Null) Bi-directional</b>		<b>R Object and S Object are equivalent.</b>	Relation between two objects with no tag.

**Table 1.4: OPM Tagged Structural Relations [Do2]**

The best way to learn OPM is through examples. The following diagram shows an example of a car stopping system. The details included in the diagram are selected to show a variety of OPM constructions.



**Figure 1.1: Example Object Process Diagram**

Each of the links in the diagram corresponds to an OPL sentence. Numeric annotations are included in this diagram to help identify the link with its corresponding sentence.

1. **Car** consists of **Powertrain**, **Body**, **Electrical System**, **Interior** and **Chassis**. (Aggregation Sentence)
2. **Chassis** consists of **Braking System**. (Aggregation Sentence)
3. **ABS** is a **Braking System**. (Specialization Sentence)
4. **Car** exhibits **Velocity**, which can be **non-zero** or **zero**. (Exhibition and State Enumeration sentence)
5. **Stopping** changes **Velocity** from **non-zero** to **zero**. (Change sentence)
6. **Stopping** requires **Braking System**. (Instrument Sentence)
7. **Driver** handles **Stopping**. (Agent Sentence)

## 1.2 Axiomatic Design

Axiomatic Design is a method for designing systems developed from an essentially different point of view than OPM. While OPM strives to provide a rich and flexible, yet standard, framework for describing systems, Axiomatic Design provides a decision-making process for constructing good designs according to basic principles or axioms. Professor Nam Suh, the developer of Axiomatic Design, states, "In order to obtain better performance, both engineering and management structures require fundamental, correct principles and methodologies to guide *decision making in design*; ... the fact that there are *good design solutions* and *unacceptable design solutions* indicates that there exist features or attributes that distinguish between good and bad designs... the *features* associated with good design may have *common elements*. These *common elements* may then form the basis for developing a unified theory for the synthesis process." [Su1, p.5] The synthesis process Professor Suh refers to is the design of any system. Axiomatic Design is the unified theory he developed by identifying the common elements of good designs.

The fundamental elements of an Axiomatic Design analysis are Functional Requirements (FRs) and Design Parameters (DPs). The goals of a system architect need to be translated into FRs, which define the problem to be solved in terms of desired function. In determining FRs for an original design it is important to define them in a *solution-neutral* form. DPs are the physical solutions selected to satisfy the FRs. Both FRs and DPs have hierarchies and can be decomposed. Professor Suh argues that "FRs at the  $n$ th level cannot be decomposed into the next level of the FR hierarchy without first going over to the physical domain and developing a solution that satisfies the  $n$ th level FRs with all the corresponding DPs. That is, we have to travel back and forth between the functional domain and the physical domain in developing the FR and

DP hierarchies." [Su1, p.36] Especially at high levels of the decomposition, DPs may be thought of as concepts selected for embodying function in form. At lower levels of the decomposition, DPs can be the actual parts used in the design, thus DPs describe form directly.

In a proper Axiomatic Design decomposition, each FR at each level of the decomposition has a corresponding DP intended to satisfy that FR. This relationship between the functions and physical design variables is key to defining a good design. It is the subject of the first axiom upon which Professor Suh bases his theory:

**Axiom 1, The Independence Axiom**  
 Maintain independence of FRs.

An alternate form of this axiom describes in more detail what is meant: "In an acceptable design, the DPs and FRs are related in such a way that a specific DP can be adjusted to satisfy its corresponding FR without affecting other functional requirements." [Su1, p.48]

Evaluating whether the Independence Axiom is satisfied is accomplished by constructing a design matrix that lists FRs down the left-hand side and DPs across the top. In a simplified form of the matrix, an "x" is entered in each square for which the corresponding DP (listed at the top of the column) affects the corresponding FR (listed at the left of the row). A simple example illustrates this.

	DP1	DP2
FR1	x	x
FR2	x	0

**Table 1.5: Example Design Matrix**

In this design matrix, FR1 is affected by adjustments in both DP1 and DP2. FR2 is affected only by adjustments in DP1. In general, *good* designs can be represented by lower-triangular matrices (all entries below the main diagonal are zero). Such designs are called "decoupled." The design represented in Table 1.5 is such a design because its elements can be rearranged to obtain a lower-triangular matrix. *Ideal* designs can be represented by diagonal matrices (all entries off the main diagonal are zero). Such designs are called "uncoupled."

	DP1	DP2	DP3	DP4
FR1	x	0	0	0
FR2	x	x	0	0
FR3	x	x	x	0
FR4	x	x	x	x

**Table 1.6: Decoupled Design**

	DP1	DP2	DP3	DP4
FR1	x	0	0	0
FR2	0	x	0	0
FR3	0	0	x	0
FR4	0	0	0	x

**Table 1.7: Uncoupled Design**

In uncoupled designs, each FR has one and only one DP whose adjustment affects it. This means that satisfying the FRs is a straightforward task of adjusting each DP to the proper setting. Work on each DP to achieve each FR can proceed in parallel without worry about interactions between various DP settings. For decoupled designs, the task of satisfying all FRs is more complicated but still achievable. Identifying DP setting adjustments must be done in order. For example, in Table 1.6, the setting for DP1 may be fixed first. Although this affects FR2, FR2 can still be achieved by fixing DP2 appropriately. Both DP1 and DP2 affect FR3, but FR3 has another degree of freedom, it can be achieved by fixing DP3. Proceeding in this order, all FRs can be satisfied.

In a precise application of Axiomatic Design, design matrices are actually specified by matrix equations. For a given vector {FR} of functional requirements, the

design process is defined as choosing a correct set (or vector) of design parameters {DP} that satisfactorily solves the equation {FR}=[A]{DP}. In this formulation, [A] is the design matrix:

$$[\mathbf{A}] = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & \vdots & & \vdots \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix}, \text{ with } A_{ij} = \frac{\partial FR_i}{\partial DP_j}.$$

**Equation 1.1: Design Matrix Equation**

### 1.3 Connecting OPM and Axiomatic Design

The purpose of this thesis is to advance both OPM and Axiomatic Design by applying the strength of each to enhance the other. This includes using the descriptive capabilities of OPM to develop better guidelines for systematically representing system architecture in terms of FRs and DPs and using the evaluative principles of Axiomatic Design to identify patterns in OPDs that indicate how well a design adheres to the Independence Axiom.

Chapter 2 outlines a standardized strategy for representing system function and architecture using OPM. It argues that adequate description of both function and architecture requires a *combination* of objects and processes. In the case of functions, the object corresponds to an operand; the process describes the intended service or use. In the case of architecture, the object corresponds to system structure; the process describes system behavior. These combinations of objects and processes conform to straightforward patterns in OPM; thus it is possible to construct OPM templates for representing function and architecture. The chapter concludes with a discussion of the "concept mapping" between function and form. Experts in system architecture have

described form being mapped to function via concept. This takes on an explicit meaning through the OPM notion of specialization.

Chapter 3 applies the conclusions of Chapter 2 regarding generic function and architecture to develop guidelines for developing good FRs and DPs. It includes a general discussion on writing good requirements along with a brief example of how OPM might be used to represent system constraints. It provides an analysis of FRs and DPs from several examples and illustrates how the FR-DP decomposition is expressed by specializing an FR-related process describing intent to a DP-related process describing behavior.

Chapter 4 explores how certain patterns in OPDs reveal a design's adherence to Suh's Independence Axiom. Objects and processes in OPDs are connected by links indicating various effects. Paths of links that form loops correspond to coupled designs that violate the axiom. Paths comprised of a single link appear in the OPDs of uncoupled designs that comply with the axiom. The chapter concludes with a general discussion of how the existence in OPDs of lengthy and looping effect paths indicates undesirable complexity in a design.



## 2. Representing Function and Architecture through OPM

### 2.1 The WHATs and HOWs of Design

The "WHAT-HOW" decomposition is a classic approach to system design problems. WHAT refers to *what* is desired—an objective or requirement. HOW refers to *how* the objective or requirement is fulfilled. An entire system can be detailed by successively identifying the WHATs and HOWs at each level of the design hierarchy. Among the system development methods that employ this paradigm are Quality Function Deployment and Axiomatic Design.

#### The QFD Framework for Design

Quality Function Deployment (QFD) was conceptualized in Japan in the 1960's, applied in Japanese industries in the 1970's, and applied in US industries in the 80's. It is a method for systematically identifying and implementing customer-desired functionality in designs.

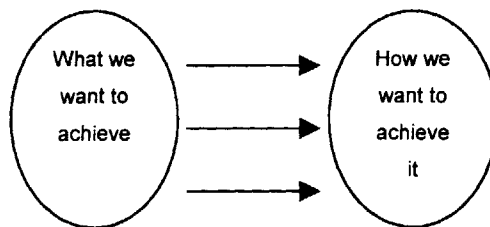
"QFD starts with a list of objectives, or the WHATs that we want to accomplish. In the context of developing a new product, this is a list of *customer requirements* and is often called the *Voice of the Customer*... Once the list of WHATs is developed, each will require further definition. We refine the list into the next level of detail by listing one or more HOWs for each WHAT..." [ASI, p. 3-5, 3-6]

QFD relates WHATs to HOWs through use of a series of interconnected matrices, often referred to as "Houses of Quality." WHATs are listed down the left-hand side of the matrix, HOWs are listed across the top. The extent to which each HOW is capable of influencing or satisfying each WHAT is recorded at their intersection cell in the matrix. In practice, there are many methods for doing QFD, championed by a variety of practitioners. There are also a variety of descriptions of WHATs and HOWs. WHATs

are designated by titles such as "Customer-desired Qualities," "Customer Wants", "Customer Needs," or "Requirements". "Think of them as *what* the customer wants—the individual characteristics of the product, service, or problem... Qualities, attributes, and requirements are all *Whats*." [GP, p. 48]. The corresponding HOWs are designated by titles such as "Technical System Expectations", "Functional Requirements", "Company Measures", or any of several others. One particularly expansive definition of HOWS reads, "*Hows* are ways of achieving *Whats*. Virtually any idea that can help solve a problem is a *How*. *Hows* consist of processes, facilities, and methods. They are also people, departments, and functions in organizations." [GP, p. 67] From the many interpretations and labels, it would be difficult to distill an exact, agreed definition of these terms and clear guidelines for what kind of items ought to be captured in these categories.

### Axiomatic Design Framework

One of the purposes of Suh's Axiomatic Design is to make the design process more rigorous. He states: "[a] rigorous design approach must begin with an explicit statement of '*what* we want to achieve' and end with a clear description of '*how* we want to achieve it.'"



**Figure 2.1 Suh's Mapping that Defines Design [Su2, p.3]**

Suh expands on the ideas in QFD to try to make clear the domains in which the WHATs and HOWs reside: "Design involves a continuous interplay between *what we want to achieve* and *how we want to achieve it*... the *objective* of design is always stated

in the functional domain, whereas the physical solution is always generated in the physical domain." [Su1, p.25, 26] It is corresponding to these two domains that Suh establishes his version of WHATs and HOWs: FRs and DPs. "Once we understand the customer's needs, this understanding must be transformed into a minimum set of specifications, which will be defined later as functional requirements (FRs) that adequately describe 'what we want to achieve' to satisfy the customer's needs. The descriptor of 'how we want to achieve it' may be in the form of design parameters (DPs)." [Su2, p. 4]

### **Existing Definitions of FRs and DPs**

Suh says that FRs must be established "from the needs the final product or process must satisfy." [Su1, p. 30] He defines FRs as "[a] minimum set of independent requirements that completely characterize the functional needs of the product (or software, organizations, systems, etc.) in the functional domain." In order to make the Independence Axiom operational, he adds "[b]y definition, each FR is independent of every other FR at the time the FRs are established." [Su2, p. 14] Thus, for original systems, the top-level FRs are defined to be independent automatically. The Independence Axiom then requires that architects maintain this independence as they select DPs to satisfy the FRs and further decompose the system. In practice, independence is demonstrated using the design matrix. In improving an existing system, the architect may find that independence was not maintained. In this case, an analysis of coupling in the system using the design matrix reveals areas in which the design may be improved.

Suh places DPs in the physical domain. In his first book, he describes DPs as the "physical embodiment...chosen to satisfy the FRs." [Su1, p. 26] He clarifies, "by the word *physical* we include all things that generate desired output." [Su1, p. 38] Suh's

definition in his second book is slightly more precise: "Design Parameters are the key physical variables (or other equivalent terms in the case of software design, etc.) in the physical domain that characterize the design that satisfies the specified FRs." [Su2, p. 14]

## **Constraints**

In addition to FRs, a system architect must satisfy constraints, which Suh describes as "Bounds on an acceptable solution." [Su1, p. 39] As Pahl and Beitz observe: "The fulfillment of the technical function alone does not complete the task of designers...the solution of technical tasks imposes certain constraints or requirements resulting from ergonomics, production methods, transport facilities, intended operation, etc." [PB, p. 45] Other constraints arise from considerations of economic feasibility, safety, and environmental concerns. Under the Axiomatic Design framework, constraints do not have to be independent of other constraints or FRs; thus the ability to clearly distinguish FRs from constraints is important for setting up the design problem. Chapter 3 discusses the distinctions between FRs and constraints in more detail.

## **WHATs, HOWs and OPM**

Suh's definitions of FR and DP help clarify the meaning of WHAT and HOW and add rigor to the design process, but there is an opportunity to introduce even more rigor. The definitions contain the terms *function*, *functional need*, and *physical embodiment*. These are terms that could be more precisely defined via OPM. OPM supplies the fundamental building blocks of objects and processes and formal rules for combining them that provide more precise and consistent definitions.

What is the correspondence between Objects and Processes and FRs and DPs? Our first impulse may be to think of simple answers to the questions WHAT and HOW? One might reasonably think that WHATs should be expressed as objects (typically

associated with nouns) and HOWs should be expressed as processes (typically associated with verbs). However, this association is exactly opposite of a recommendation by Suh: "It should be noted here that all FRs are stated starting out with verbs. This is a good way of distinguishing a FR from a DP, which should start with a noun, if possible." [SCL, p.3]

Many techniques for identifying system function employ the "verb-noun" rule, which specifies that a function be described by an active verb together with a noun. Examples might be "support weight," "control speed," "lift object," etc. The implication of this for representing WHATs in OPM is that a fully expressed FR ought to be portrayed by at least one object and one process together with an effect link. In fact, in OPM "no process exists unless it is associated with at least one object, for the transformation of which it is responsible," [Do2, p. 70].

In distinguishing FRs from constraints, Suh says "a specific range of design values must be maintained for each FR at all times," [Su1, p.29]. Maintaining the *level* of design values within the desired range is the requirement. In OPM, the level of the design value is an object that can be represented with various attribute values or states. Changing the level is accomplished by a process, for only a process can change the attribute values or states of objects.

While characterization of an FR requires both an object and process, it might seem that because a DP is a physical concept, it could be sufficiently characterized by objects alone. However, such a characterization would neglect a fundamental relationship between FRs and DPs. The Design Matrix equation (Equation 1.1) expresses elements  $A_{ij}$  of the design matrix as partial derivatives of FRs with respect to DPs.

$$A_{ij} = \frac{\partial FR_i}{\partial DP_j}$$

This formulation implies that some characteristic of a DP is changeable, and this change can affect the FR. Thus as an answer to the question HOW? the physical aspect of a DP, represented by objects, is inseparably connected to a dynamic aspect represented by processes.

Dori suggests a HOW is properly expressed as an architecture—a *structure/behavior combination* that attains the WHAT, i.e., the function. In OPM, structure is represented by objects connected by structural links; behavior is represented by processes that affect objects depicted through the connection of these processes and objects by transformation links.

The observations made so far suggest that both WHATs and HOWs ought to be defined as combinations of objects and processes. In order to arrive at this conclusion rigorously, it is necessary to now review and settle on clear definitions of function and architecture using the language and rules of OPM.

## 2.2 What is Function?

The word "function" has a variety of meanings depending on its context. It is useful to review these definitions in order to arrive at a working definition for this thesis.

### Standard English Language Definitions

The Oxford English Dictionary provides the following definitions of function:

1. In etymological sense: The action of performing; discharge or performance of (something).
2. Activity; action in general, whether physical or mental.
3. The special kind of activity proper to anything; the mode of action by which it fulfils its purpose. Also in generalized application, esp. (Phys.) as contrasted with structure.

These definitions contain three key elements that generally appear in more specialized definitions: action, performance, and fulfilling a purpose.

### Mathematics Definition

The mathematical definition of a function is perhaps the most specialized and restrictive: "An association of exactly one object from one set (the range) with each object from another set (the domain)." [JJ, p.153] The key idea here is that a function relates or associates entities with other entities and only one entity in the range may be associated with any item in the domain.

### Programming Definitions

Function used in the context of computer programming generalizes the mathematical definition: "Functions are among the most common kinds of relationship. In functional relationships, at least one direction of the relation associates a single element of one domain to those in the other." [De, p. 53] Sometimes function is used interchangeably with operation or subroutine. For example, an operation can be defined as, "[t]he action of an operator or function, which takes one or more pieces of data and

produces a new piece of data." [CST, p. 372, 629] This definition suggests the idea of input and corresponding output, which is represented in OPM by processes that transform objects.

### **Axiomatic Design Definition**

In the realms of system and product design, definitions of function become less precise. Suh defines the word function rather generally: "By the word *function* we mean the desired output." [Su1, p. 38] Again, this definition emphasizes output, but it also includes the element of desire (i.e., intent). In creating a system, the architect intends to provide a service for the system users and beneficiaries. Of course, what is a service to one may be a disservice to another. From this point of view, a system's function is subjective—perhaps identified differently by the architect and individual users.

### **System and Product Development Definitions**

In the context of explaining system architecture, Crawley describes function as "The activities, operations and transformations that cause, create or contribute to performance (i.e., meeting goals), or the actions for which a thing exists or is employed." [Cr2, 1/19, p. 6] This definition preserves the key ideas of action, performance, and fulfilling a purpose and is generally applicable to all kinds of systems.

As people have devised specific approaches to developing systems, more formal definitions and rules for stating functions have been developed. This is especially true in the area of product development. For example, identifying function is a key step in the beginning of Failure Mode and Effects Analysis (FMEA) applied to the development of new products. The Ford Motor Company FMEA handbook gives the following rules for identifying functions:



A description of the Function should answer the question: "What is this item supposed to do?" Functions are design intent or engineering requirements. Functions are:

- Written in Verb/Noun/Measurable format.
- Measurable...
- Design intent or engineering requirement
- Representation of all wants, needs and requirements, both spoken and unspoken for all customers and systems

[Fo1, p. 4-21]

These rules maintain key elements seen in the preceding definitions, but they extend too broadly to be considered rigorous. For example, an engineering requirement should stem from a function but should not be confused with the function itself. Furthermore, describing functions as representations of all "wants, needs and requirements..." is simply too expansive to be useful.

A better set of rules for describing function is provided by Otto and Wood who follow the guidelines of Pahl and Beitz:

"A function of a product is a statement of a clear, reproducible relationship between the available input and the desired output of a product, independent of any particular form... The *product function* is the overall intended function of the product—what it is to do; [it] is the simplest representation of the product, usually just a noun and an active verb."  
[OW, p. 151]

This description captures two important ideas. First, function is independent of form.

This is important because it distinguishes function from the behavior of a particular design solution. It means more than one design solution can fulfill the same function.

Second, functions can be stated using a noun-active verb combination. This corresponds nicely to an object-process classification of functions that will be developed rigorously using OPL and OPDs.

Pahl and Beitz, who have developed an extensive methodology for engineering design, apply the term function specifically to the conversion of energy, material, or signals in engineering applications. They capture the flow of these items within a system in diagrams known as *function structures*. Examples of functions recorded in these

diagrams might be "increase pressure," "transfer torque," or "reduce speed." While OPM is capable of depicting a much broader range of function, the discipline of this approach can be helpful to any architect developing an engineered system.

### **OPM Definition: Function vs. Behavior and Structure**

The key elements of action, performance, and fulfilling a purpose seem to appear in some form in each of the design-related definitions of function. Dori captures these elements in his OPM-based definition: "Function is an object attribute that describes what the object does, what phenomenon it exhibits, what service it supports, or what it is used for. ...[Its definition] emphasizes the 'what' aspect and is not concerned with the 'how.' This distinguishes function from dynamics, as dynamics is about *how* the object operates, while function is about *what* it does." [Do2, p. 97, 4.1.2]

Dori goes on to distinguish function further from structure and behavior. "...the system's function dictates the structure of the system and the way this structure operates—its behavior, or dynamics. A unique combination of structure and behavior enables the system to function—to achieve the goal for which it is designed." [Do2, p.110, 4.4.2]

These explanations are in line with Otto and Wood's observation: "[Function] is what a system does as opposed to what it is." [OW, p. 165] Dori would elaborate even further using OPM:

- What the system *does* is its dynamics (or equivalently, its behavior)
- What the system *is*, is its structure.
- What *purpose* the system serves for the beneficiary is its function.

This perspective allows function to be defined subjectively—relative to intent. So, depending on the purpose they wish to achieve, a system architect or system user may state function differently for the same system. Dori contends that the term "system" itself

is subjective. "Whether or not an object is a system is in the eye of the beholder." [Do2, p. 101] Formally he defines a system as "an object that carries out or supports a significant function." [Do2, p. 99]

Although this subjectivity might be confusing when trying to represent a system via OPDs, it is necessary to address these different perspectives unless portions of the system's purpose or use are to be ignored. For example, Chapter 3 includes a discussion on how to incorporate intent and perspective in OPDs in order to preserve this information for future architects or users to whom this information may be unclear.

### **The Dynamic and Static Aspects of Function**

The groundwork has been laid for discussing how function should be represented in OPM. Certainly the "action" element of function relates to processes. Otto and Wood make this connection explicit: "A function is defined in terms of a description of a process." [OW, p. 165] Thus, functions have a dynamic aspect, represented by processes. These processes describe the architect's intended *service* to be provided by the system or the user's intended *use* of the system. Consider a freezer, one of Nam Suh's basic examples: the intended service to be provided by the freezer is **Food Preserving**, which is a process in OPM. The process is accomplished through subprocesses that freeze the food and ensure air temperature is kept within desired limits.

Crawley acknowledges the "process element" of function, but also identifies the other critical element that has appeared frequently in our discussion: intent. "Function is process with intent." [Cr1, 9/9, p. 24] However, intent expressed as a process is incomplete without an associated object. This follows a basic tenet of OPM: "no process exists unless it is associated with at least one object, for the transformation of which it is responsible." [Do2, p. 70] Thus functions have a static aspect, represented by an object,

which is the function operand. In OPM an operand is called a "transformee" in order to emphasize that some process transforms the object. Related attributes and parts of the object are also static aspects of the function. In the freezer example, food is the operand; the freezing process transforms it so that its spoilage rate is significantly slowed.

### A Generic OPM Template for Function

In general, OPM can be used to portray functions, capturing both their dynamic and static aspects. Figure 2.2 portrays an OPM template for a generic function. (Note that in OPDs throughout this thesis, elements drawn with dotted lines and words in bold italic font are annotations that are not part of the actual diagram.)

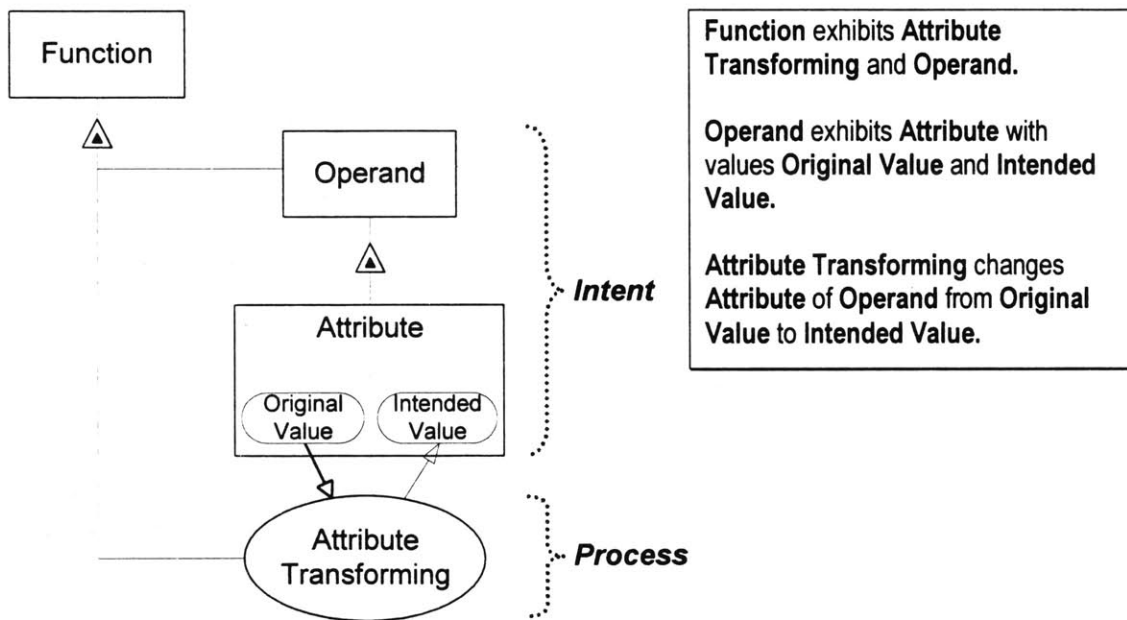


Figure 2.2 OPD and OPL Script for a Generic Function

The dynamic aspect of function appears in the OPD in **Attribute Transforming**, a process. The static aspect appears in the **Operand** and **Attribute**, objects. Intent appears in these objects through values of **Attribute**. It is possible to simplify the OPD

by omitting **Attribute** and drawing an effect link directly between **Attribute Transforming and Function Operand**. However, **Attribute** is included explicitly in this general case because it often corresponds to a metric that is important to the system architect or user.

Based on definitions in the system engineering and product development literature, this template should lead to valid representations of function in OPM. Certainly there is a correspondence between the "noun-verb" concept of a function and the **Operand Attribute/Attribute Transforming** structure in the OPD. The description of what a system *does* as opposed to what it *is* is also clear: **Attribute Transforming** describes what the system does, while the diagram includes no description of any system. Finally, Suh's notion of "desired output" is represented by the intended value of **Attribute**.

However, the only way to really validate the template is to develop examples. Many authors have classified various types of function. Pahl and Beitz summarize some of these classifications and make use of one based on the work of Krumhauer. These "generally valid functions" [PB, p. 34] are formulated with specific input-output relationships, useful for their ability to be represented in computer applications during the conceptual design phase. The following table lists these functions:

Input(I)/Output(O) Characteristic	Generally Valid Function	Explanation
Type	Change	Type and outward form of I and O differ
Magnitude	Vary	$I < O$ $I > O$
Number	Connect	Number of I $> O$ Number of I $< O$
Place	Channel	Place of I $\neq O$ Place of I = O
Time	Store	Time of I $\neq O$

**Table 2.1: Pahl and Beitz List of Generally Valid Functions [PB, p.36]**

The correspondence between each of these functions and the generic OPM template is straightforward. The Input/Output Characteristic listed in the table corresponds to an attribute of an operand; the Generally Valid Function corresponds to a process; the Explanation describes the change in states of the attribute. The "Store" function might be considered problematic because Time is listed as a characteristic, and in OPM time is typically associated with processes and not used to characterize objects. In this case, Time may be considered to characterize the "time spent by an object in a certain state." The Storing process belongs to a class of processes Dori calls "State Maintaining Processes." These processes are associated with verbs whose meaning is to maintain an object as it is for some more time, e.g., maintaining, containing, prolonging, etc. Representation of these processes in OPM includes the use of a special "state-maintaining" link that will appear in some upcoming examples.

Little, et al, have prepared a similar "Limited Syntax" classification of function that includes several subcategories for each of the classes: Channel, Support, Connect, Branch, Provision, Control Magnitude, Convert, and Signal. [Cr2, 1/19/01]

Correspondence between these functions and the generic OPM template would be similar to that outlined above. OPM is flexible enough to model these functions as well as more general functions that convey even high-level intent of system architects.

Presented here are some OPD renderings of function based on simple systems mentioned by Suh and Dori. These examples capture the initial idea in a system's development, displaying intent with an associated process. Design solutions to achieve the intent are not included in the diagrams, but will be added in future extensions of the examples. Each example identifies the perspective from which the intent is being modeled (system architect or user), the function operand, the service or use intended, the attribute being affected, and the target value of the attribute.

### Example 2.1: Food Preserving

A person desires a system to preserve food. Is this a function? Can this desire be described as a "process with intent?" One may argue whether "preserving" properly conveys a "pattern of transformation" [Do2, p. 70] that defines a process. However, it is not hard to formulate the desire in terms of the change intended. Process-oriented descriptions for preserving food include *slowing* spoilage rate or *extending* shelf life. For this example, **Spoilage Slowing** is selected as the process. The intent is to change the **Spoilage Rate** from fast to slow as illustrated in the following OPD.

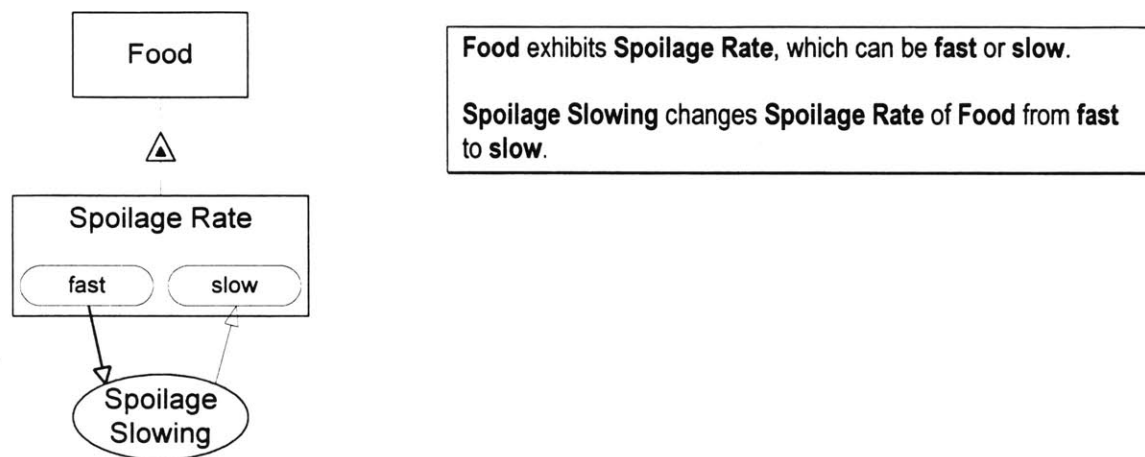


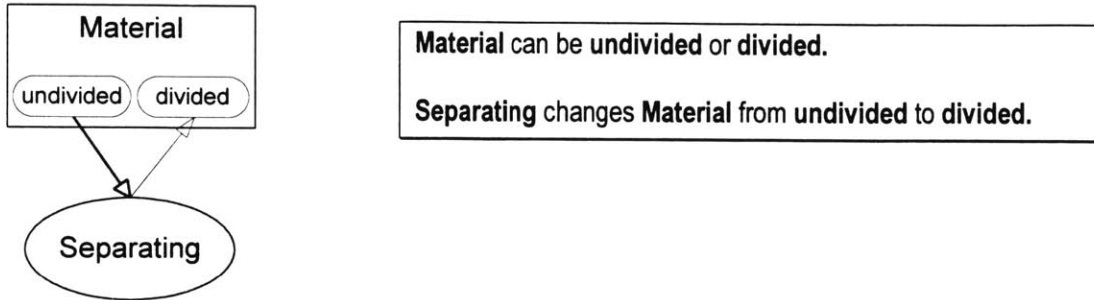
Figure 2.3 OPD and OPL Script for Food Preserving

Several alternatives exist for fulfilling this function, including dehydration, freezing, sealing in an airtight container, etc.

Summary	
Modeling Perspective	System Architect or User
Operand	Food
Service or Use	Slow Food Spoilage (Preserve Food)
Attribute (Metric)	Food's Spoilage Rate
Target Value	Slow

### Example 2.2: Material Separating

A person desires to separate material. Although continuity could be identified explicitly as the attribute the person would like to change, it is implicit in the states assigned to Material in this OPD:



**Figure 2.4: OPD and OPL Script for Material Separating**

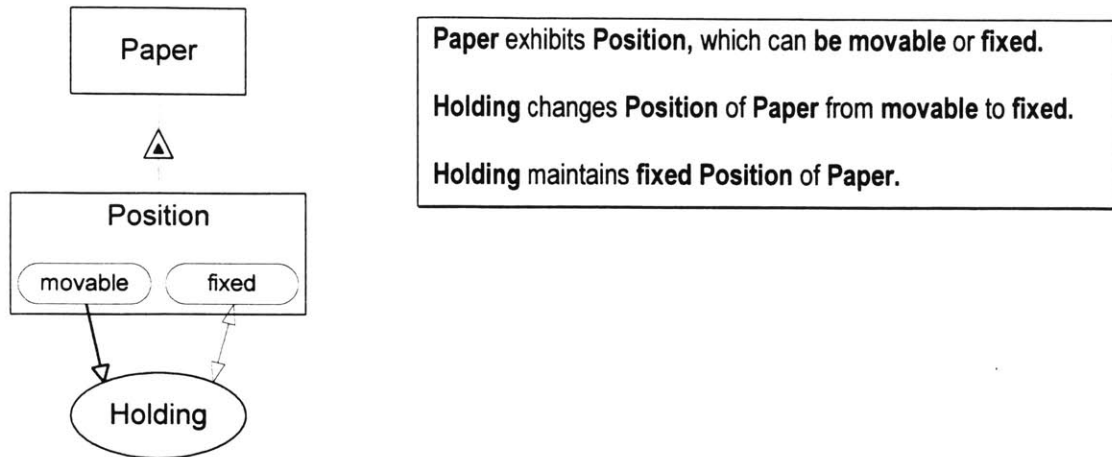
Possible alternatives for fulfilling this function include using an existing tool to such as a pair of scissor to cut the material or a knife to slice the material. If there is no tool available, one might choose to use oneself as a material separating system and tear the material.

<b>Summary</b>	
Modeling Perspective	System Architect or User
Operand	Material
Service or Use	Separate Material
Attribute (Metric)	Material's "Dividedness" or Continuity
Target Value	Divided



### Example 2.3. Paper Holding

A person desires to hold paper in a fixed location so it doesn't blow away or become separated from its pile.



**Figure 2.5: OPD and OPL Script for Paper Holding**

Possible alternatives for fulfilling this function include applying weight with a paperweight, fastening the paper to a surface with a pin or staple, gluing the paper to a surface with adhesive, etc. Other requirements of the beneficiary will determine which alternative to select. For example, if the paper needs to be easily removed from its fixed position, then the paperweight may be most desirable.

<b>Summary</b>	
Modeling Perspective	System Architect or User
Operand	Paper
Service or Use	Hold Paper
Attribute (Metric)	Position
Target Value	Fixed

### Example 2.4: River Crossing, Step 1

A person desires to cross a river, but no system for crossing the river is available. In order to fulfill the desire to cross the river, the person must fulfill a preliminary step. He/she decides to become a system architect with the desire to provide a means of crossing the river. "Provide" is a nondescript word to describe a function. Keeping the definition of process in mind, is it possible to select a better word? What transformation is taking place? The river is changing states: from uncrossable to crossable.

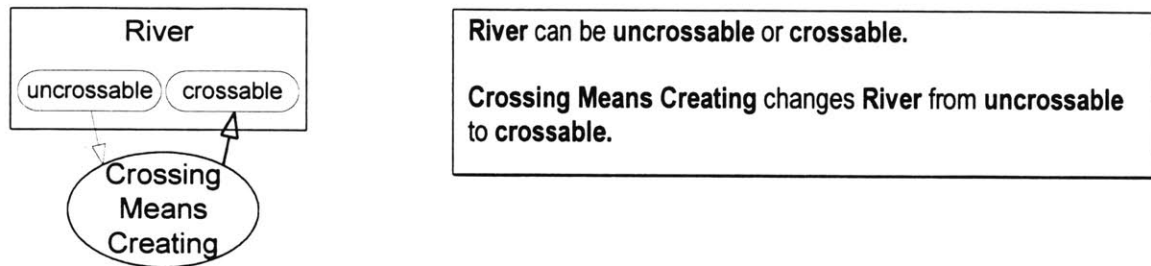


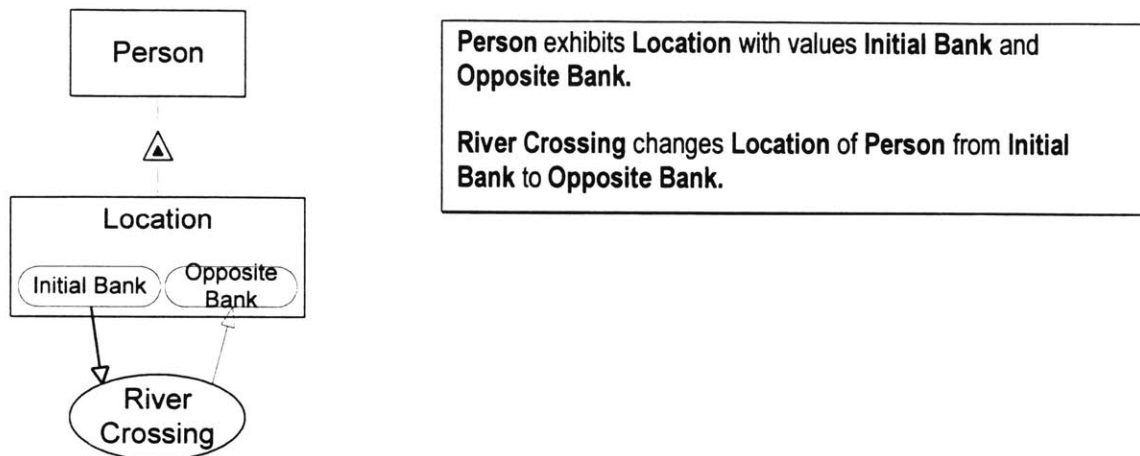
Figure 2.6: OPD and OPL Script for River Crossing Enabling

Possible alternatives for fulfilling this function include building a bridge, instituting a ferry service, providing helicopter service, etc. These kinds of system creation activities precede the fulfillment of system user desires. In contrast to Examples 2.1-2.3, this example includes both the system creation step and the system operating step in order to explicitly show that each step may generate different OPDs from different perspectives. The OPD model for Step 1 will represent the domain of *system design*, while the OPD for Step 2 will represent the domain of *system operation*.

Summary	
Modeling Perspective	System Architect
Operand	River
Service or Use	Create a Means of Crossing
Attribute (Metric)	River's "Crossability"
Target Value	Crossable

### Example 2.4: River Crossing, Step 2

A person desires to cross a river. A means for crossing the river is available. Instead of capturing the intent to change states of the river, the OPD captures the intent to change states of the river crosser, i.e., changing their location from one side of the river to the other.



**Figure 2.7: OPD and OPL Script for River Crossing**

Possible alternatives for fulfilling this function include swimming, or using a bridge, ferry, or helicopter.

<b>Summary</b>	
Modeling Perspective	User (Person)
Operand	User
Service or Use	Cross River
Attribute (Metric)	Location
Target Value	River's Opposite Bank

## 2.3 What is Architecture?

### Standard English Language Definitions

The Oxford English Dictionary provides the following definitions:

#### ARCHITECTURE:

1. The art or science of building or constructing edifices of any kind for human use.
5. Construction or structure generally;

#### ARCHITECT:

1. A master-builder.
2. One who designs and frames any complex structure; esp. the Creator; one who arranges elementary materials on a comprehensive plan.
3. One who so plans, devises, contrives, or constructs, as to achieve a desired result (especially when the result may be viewed figuratively as an edifice); a builder-up.

These definitions emphasize the building of *edifices*. The OED says an edifice is "[a] building, usually a large and stately building, as a church, palace, temple or fortress; a fabric or structure." Another dictionary extends the definition to include "elaborate conceptual structures." [AHD, 4<sup>th</sup> Ed.] It would be fair to say that a standard English description of "an architecture" is "a complex structure," and the work of the architect involves managing complexity of the structure.

### System and Product Development Definitions

The most basic definitions of architecture in the context of system design emphasize structure. For example, Rechtin and Maier describe architecture as "[t]he structure—in terms of components, connections, and constraints—of a product, process or element. [RM, p. 251] But other definitions go beyond mere structure. Crawley defines architecture as "[t]he embodiment of concept and the allocation of functionality

and definition of interfaces among the elements. [Cr1, 9/8, p. 11] Otto & Wood describe architecture as the mapping of function to form:

"The challenge in [the Concept Development stage of product design] is to translate the customer needs and business case into a realizable product concept(s). This translation is what we define as the product architecture, which is the mapping from the product function to the product form. It is the division into parts and assemblies of a product and how the functional network matches or cuts across these physical divisions and interfaces. [OW, p. 358]

The term mapping is somewhat abstract in this definition. Ulrich and Eppinger capture a similar idea, but more concisely. "The architecture of a product is the scheme by which the functional elements are arranged into physical chunks and *by which the chunks interact.*" [UE, p. 183, italics added]

This last definition identifies both static (physical chunks) and dynamic (interaction) elements. The recognition by Dori that accurate representation of systems requires equal status of objects and processes leads to a definition of system architecture that clearly recognizes both the static and dynamic aspects of an architecture: "System architecture is the overall system's structure/behavior combination, which enables it to attain its function while embodying the architect's concept. ... In fact, the system is no more and no less than its structure/behavior combination." [Do2, p. 110, 4.4.2]

Returning to the traditional view of architecture for a moment, it is interesting to note that Frank Lloyd Wright, the most celebrated civil architect in twentieth century America, agreed with this combination. "Form follows function—that has been misunderstood. Form and function should be one, joined in a spiritual union." [W] Translating this quote into the language of OPM, one might reasonably conclude, "architecture is the embodiment of structure and behavior."

## **The Static and Dynamic aspects of Architecture**

The typical description of architecture is structural. Naturally, the static aspect of architecture is structure, represented in OPM by the objects that comprise it. These objects are essentially the physical elements of a system: "the parts, components, and subassemblies that ultimately implement the [system's] functions." [UE, p. 183] For example, in the freezer, a sensor and compressor operate to maintain air temperature. This sensor/compressor combination is part of the refrigerator structure.

The dynamic aspect of Architecture includes the operations and transformations that comprise system behavior, represented in OPM by processes. These process are the operational elements of a system: "the individual operations and transformations that contribute to the overall performance of the [system]." [UE, p. 182] For example, in the freezer, the sensor senses air temperature and signals the compressor if the temperature gets too high; the compressor operates to cool the air.

Terms such as "operations and transformations" were also used to describe function. Is it proper to re-use them to describe architecture? Such duplication is likely unavoidable given that function and architecture are both dualistic, i.e., each has static and dynamic aspects. An architect's intent is stated as a system function. That function is achieved through an architecture. Boundaries between the two become a matter of perspective. However, at each stage of system development, boundaries can be made clearer using a WHAT-HOW framework and applying the concepts of OPM.

## 2.4 Function vs. Architecture

Dori makes a distinction between Function and Behavior that aligns Function with the question "What is the system supposed to do?" and Behavior with "How does the system do it?" He has described architecture as a "structure/behavior combination." Our discussion of the dualistic nature of function leads to an analogous description of function as an "operand/service combination." The service is what the system is supposed to do; the operand is what is affected. A user may not use the system for the intended service, so from his/her perspective a function may be an "operand/use combination." In fact, the concept of use is broader than that of service. The intended service of a system is usually just one of the system's possible uses. So it is appropriate in general to describe function as an operand/use combination, which corresponds well to the OPM description of function established previously.

Table 2.2 shows a summary of the ideas presented so far for expressing the WHATs and HOWs of system development. WHATs are functions and HOWs are architectures:

<b>WHAT?</b> What result do you desire?		<b>HOW?</b> How does the system achieve it?	
<b>Function:</b> Operand/Use Combination		<b>Architecture:</b> Behavior/Structure Combination	
<b>Static Aspect</b> (Object-related)	<b>Dynamic Aspect</b> (Process-related)	<b>Dynamic Aspect</b> (Process-related)	<b>Static Aspect</b> (Object-related)
What should the system affect?	What effect should the system cause?	How does the system behave?	How is the system structured?
Operand-State, Transformee	Service, Use	Behavior, Operation	Structure, Form

**Table 2.2: Organizing HOWs and WHATS by OPM Concepts**

The answers to each of the questions in the table may differ depending on who is answering. In practice these differences determine whether the system is effective and safe. For example, to the question, "How will you achieve your desired result?" the architect may answer: "By devising an architecture that will fulfill the perceived, desired function. Among the possible architectures, the one that best meets the constraints related to creation, distribution, and application will be selected for implementation." The user may answer: "By selecting and applying a system that will fulfill my purpose. Among the possible alternatives, the one that best meets my constraints for application will be selected."

The table subdivides each of the two high-level WHAT and HOW questions into two sub-questions related to static and dynamic aspects. Together, these four sub-questions elicit the basic information required for a good WHAT-HOW decomposition of a system:

- 1. What should the system affect?**
- 2. What effect should the system cause?**
- 3. How does the system behave?**
- 4. How is the system structured?**

The order of the questions is important. Question 3 follows Question 2 because the dynamic aspect of the HOW is a specific solution to dynamic aspect of the WHAT. Once Question 2 is asked, it is natural and most productive to answer Question 3 before moving on to addressing structure in Question 4. In OPM terms, the answer to Question 3 is a "specialization," of Question 2. This concept will become clear in the example OPDs that follow.

Suh's books provide many examples of FR-DP decompositions, but neither his examples nor his definitions provide a completely clear delineation of essential versus



non-essential elements of FRs and DPs. Axiomatic Design practitioners know that properly formulating FRs and DPs is one of the most challenging aspects of applying the methodology. Fortunately the framework and questions captured in Table 2.2 provide a tool for improved formulation of FRs and DPs based on templates easily constructed in OPM.

Many frameworks for system decomposition have been developed, but few incorporate the formalism found in OPM. In general however, there will be similarities between the questions generated by various frameworks. For example, the four questions highlighted here correspond reasonably to three of Crawley's questions for system architects: What, How, and Where [Cr1]. This correspondence is outlined in Table 2.3.

<b>WHAT?</b> What result do you desire?		<b>HOW?</b> How does the system achieve it?	
What should the system affect?	What effect should the system cause?	How does the system behave?	How is the system structured?
<b>What?</b>		<b>How?</b>	<b>Where?</b>
Goal-Operand, Attribute-Metric	Solution-Neutral Function	Solution-Specific Process	Form, Structure

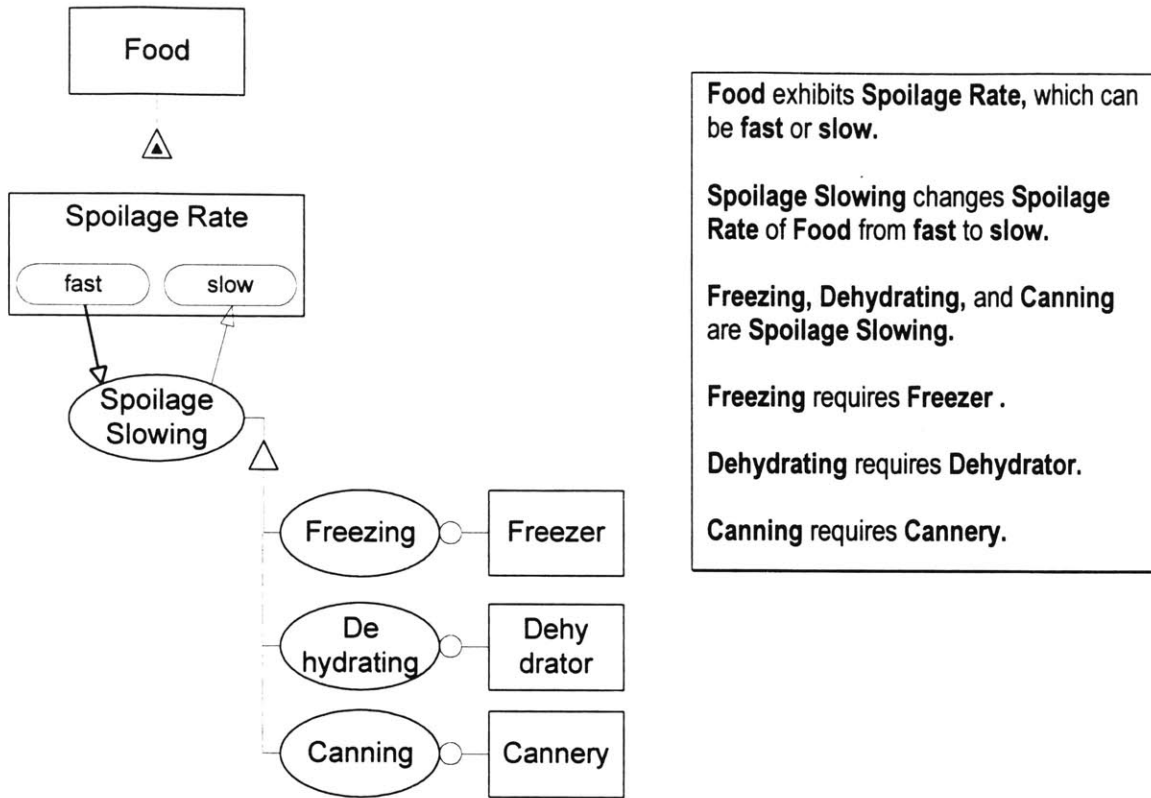
**Table 2.3 Alignment of OPM-based Questions for Architects with Crawley's Questions for Architects**

Crawley's two other questions for architects, When and Who, don't show up in the WHAT-HOW decomposition, but they are addressed in OPM. "When" corresponds to the flow of time represented by the vertical placement of processes in OPDs. "Who" corresponds to operators, users, and "affectees" represented via objects and agent and effect links in OPDs.

## Examples

It is now possible to expand on the examples presented at the end of Section 2.2. Each example of function described a WHAT—an answer to the question, "What result do you desire?" For each example, an architecture is selected that fulfills the function and describes a HOW—an answer to the question, "How does the system achieve it?" In the OPM representation of these architectures, objects that exhibit behavior or act as instruments describe system structure. A process attached to those object describes system behavior. In each case this process is a specialization of the more general process represented in the function. In the case of the OPD for Example 2.1, a variety of architectures is also shown, each representing a different alternative for fulfilling the function.

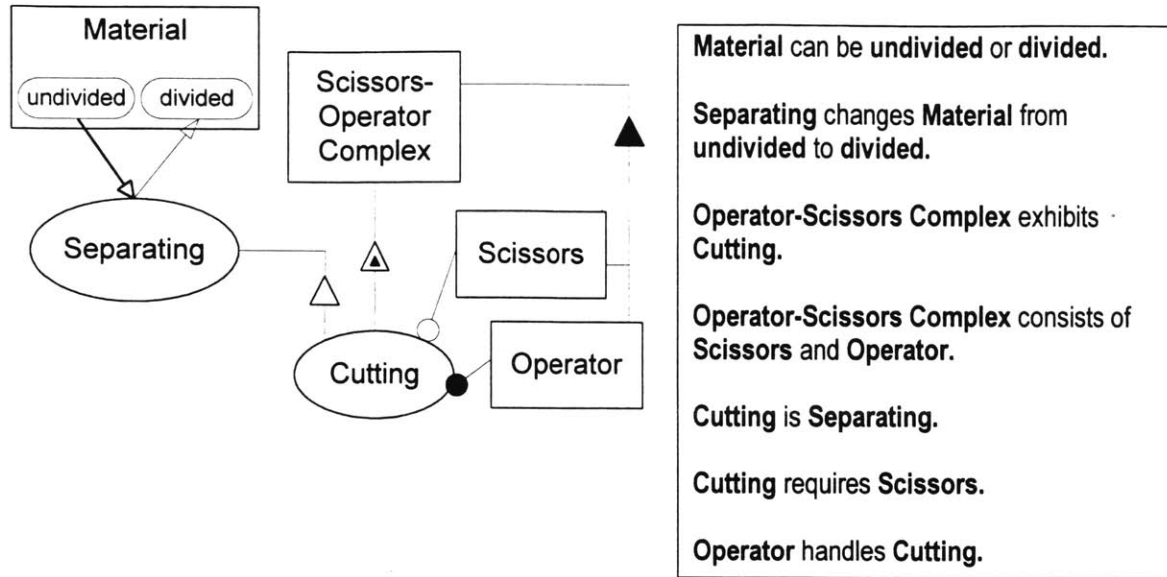
**Example 2.1, continued: Food Preserving via Various Systems**



**Figure 2.8: OPD and OPL Script for Food Preserving via Various Systems**

<b>Summary (for the freezer system)</b>		
<b>What result do you desire?</b>	What should the system affect?	Spoilage Rate of Food
	What effect should the system cause?	Slowing
<b>Elements of Function</b>	Spoilage Rate/Slowing Combination	
<b>How does the system achieve it?</b>	How does the system behave?	By freezing
	How is the system structured?	As a freezer
<b>Elements of Architecture</b>	Freezing/Freezer Combination	

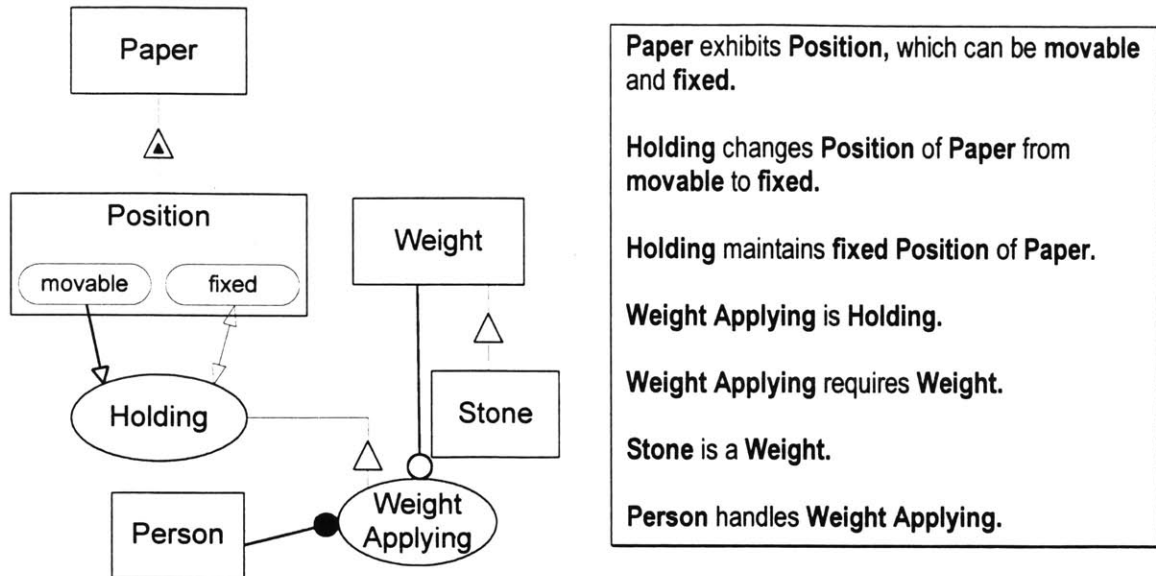
**Example 2.2, continued: Material Separating via Scissors**



**Figure 2.9: OPD and OPL Script for Material Separating via Scissors**

<b>Summary</b>		
<b>What result do you desire?</b>	What should the system affect?	"Dividedness" of Material
	What effect should the system cause?	Separating
<b>Elements of Function</b>	Material/Separating Combination	
<b>How does the system achieve it?</b>	How does the system behave?	By an operator using scissors to cut
	How is the system structured?	As an operator and pair of scissors
<b>Elements of Architecture</b>	Cutting/Scissors/Operator Combination	

**Example 2.3 continued: Paper Holding via a (Stone) Paper Weight**

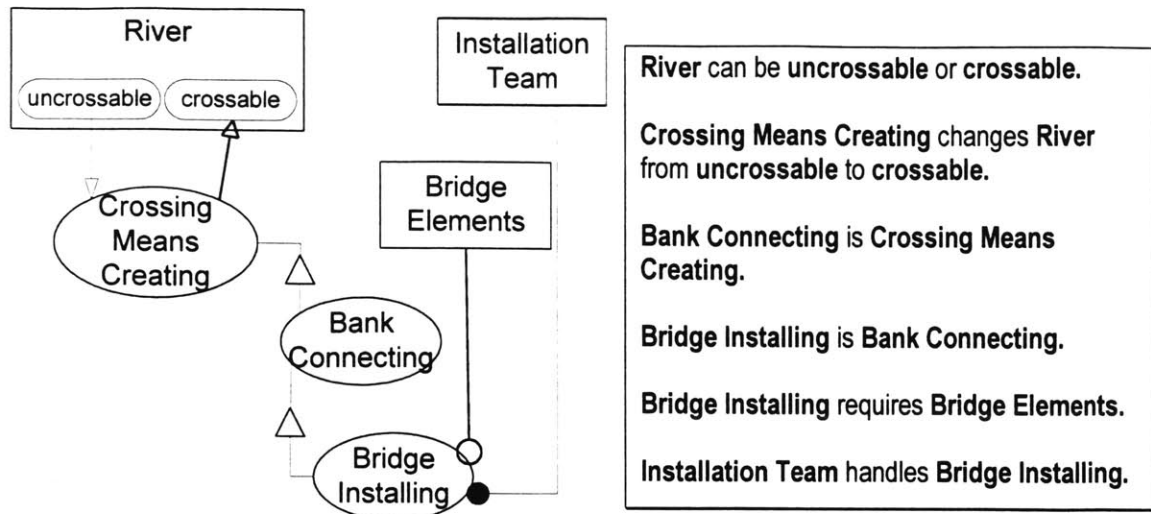


**Figure 2.10: OPD and OPL Script for Paper Holding via a Paper Weight**

Summary		
<b>What result do you desire?</b>	What should the system affect?	Position of paper
	What effect should the system cause?	Holding: keeping it fixed
<b>Elements of Function</b>	Paper Position/Holding Combination	
<b>How does the system achieve it?</b>	How does the system behave?	By applying weight
	How is the system structured?	As a stone
<b>Elements of Architecture</b>	Weight Applying/Stone Combination	

In this system, the architect and user are likely the same person and part of the system. The user places and removes the stone and once it is in place, the system exhibits no dynamics.

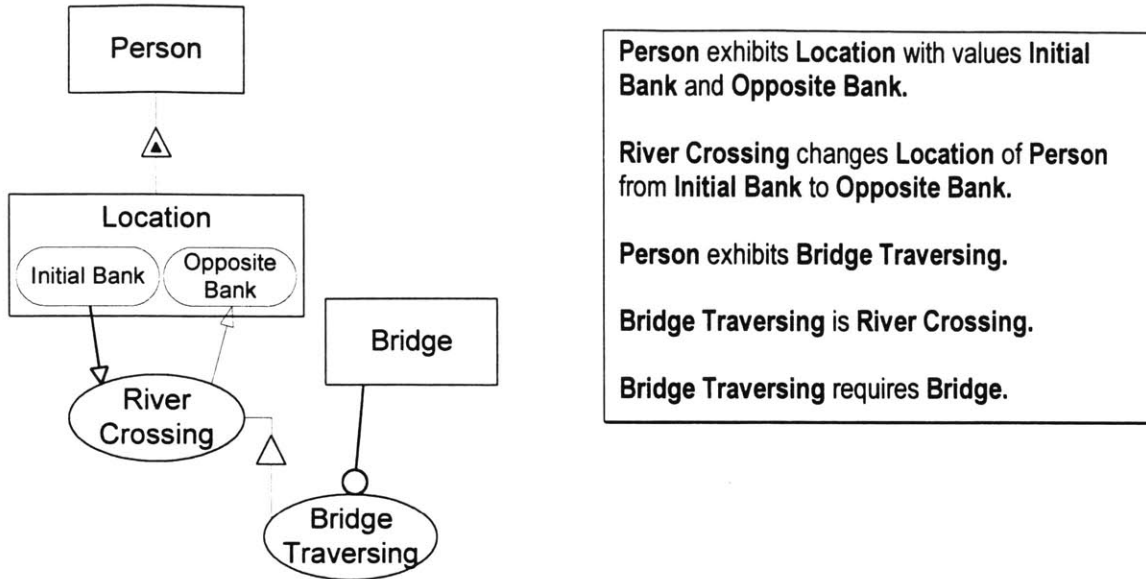
**Example 2.4 continued: River Crossing via a Bridge, Step 1**



**Figure 2.11: River Crossing Enabling via Bank Connecting by a Bridge**

Summary		
<b>What result do you (bridge builder) desire?</b>	What should the system affect?	Crossability of River
	What effect should the system cause?	Creating a means of river crossing
<b>Elements of Function</b>	River/Crossing Means Combination	
<b>How does the system achieve it?</b>	How does the system behave?	By connecting banks by installing a bridge
	How is the system structured?	As a bridge-installation team combination
<b>Elements of Architecture</b>	Bridge Installing/Team/Bridge Elements Combination	

**Example 2.4 continued: River Crossing via a Bridge, Step 2**



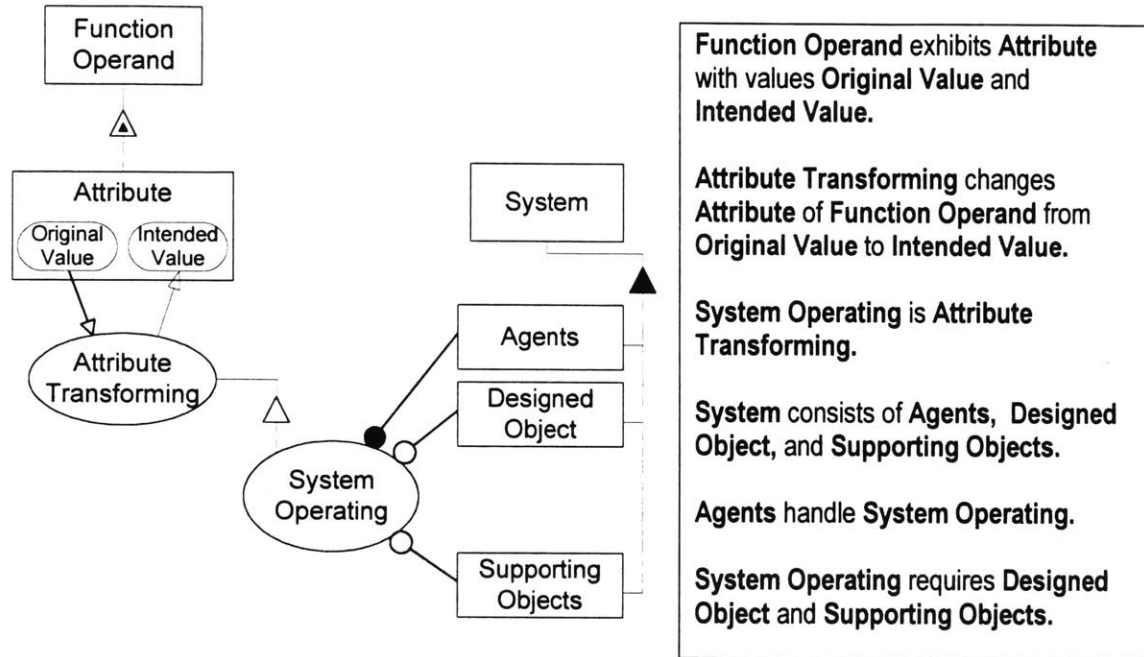
**Figure 2.12: OPD and OPL Script for River Crossing via a Bridge**

Summary		
<b>What result do you (river crosser) desire?</b>	What should the system affect?	Location of Person
	What effect should the system cause?	Crossing River
<b>Elements of Function</b>	Location/River Crossing Combination	
<b>How does the system achieve it?</b>	How does the system behave?	By a person traversing the bridge
	How is the system structured?	As a bridge-person complex
<b>Elements of Architecture</b>	Traversing/Bridge & Person Combination	

This system includes not only the bridge, but the beneficiary users as well. The system is operating when users traverse the bridge.

## Generic OPM Template for Functional WHATs and HOWs

Based on the examples shown, the generic OPM template for a function can be expanded to a generic template for a function with an associated architecture. As with functions, the template captures both the static and dynamic aspects of architecture in the objects and processes depicted. The template appears in Figure 2.3:



**Figure 2.3 OPD and OPL Script for a Generic Function and Architecture**

In the template, a HOW consists of a system that fulfills the function. The system represented is a *general* system that may include

- **Agents:** humans such as operators or controllers that enable the system
- **Designed Object:** the object designed by the system architect as a solution to fulfill the function within the system
- **Supporting Objects:** additional objects—not designed by the architect—required for the system to fulfill the function.



In a typical WHAT-HOW decomposition it is the designed object that gets most attention as the HOW. It is what the architect is focused on producing. However, the architect should always be mindful of the other elements of the system that affect successful fulfillment of the function. It may not be necessary to include all these elements in a particular OPD. For example, not all HOWs will include agents, and supporting objects may or may not be relevant in a particular model. On the other hand, it will often be important to model the broader supersystem in which the system operates. For example, this would include modeling the environment—those external surroundings and situations that may affect and be affected by the system. The choice of how to apply the template depends on the modeler's need and the context of the model.

A key idea represented in the template is that a HOW ought to include both processes and objects. In this representation, the "HOW process" is **System Operating**, which is a specialization of **Attribute Transforming** identified in the WHAT. Furthermore, the template provides a way for making a clear distinction between WHATs and HOWs in an OPM model. This distinction is made explicit by the dotted line annotations that separate the different sections of Figure 2.4.

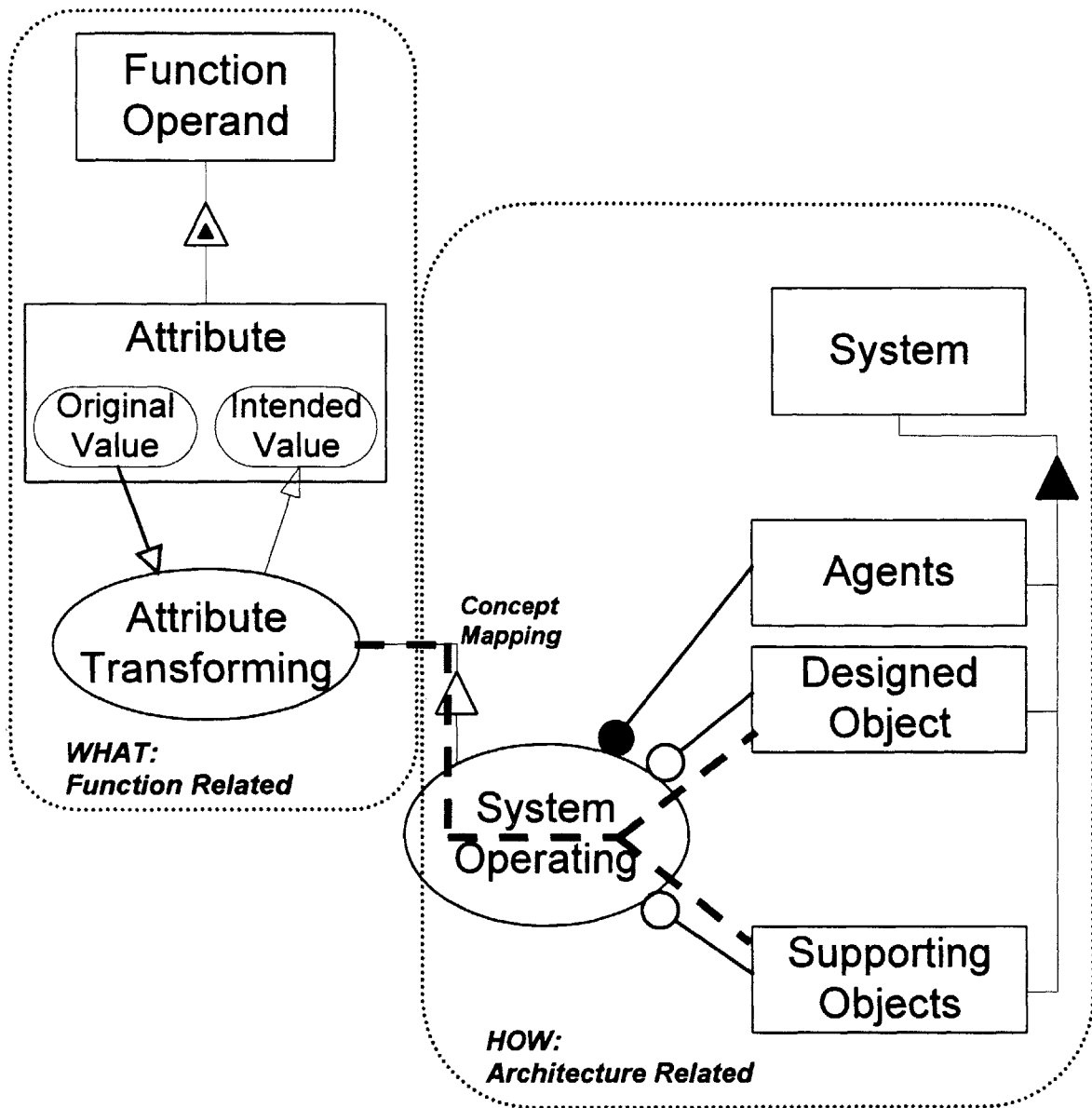


Figure 2.4: Template for WHATs and HOWs in OPM, Concept Mapping Highlighted

## Concept Mapping

What is the connection between the WHAT and the HOW? Crawley has used the term "concept" to describe this mapping. Concept is "a product or system vision, idea, notion or mental image which maps form to function and embodies working principles.

[A concept]...

- Is created [or selected] by the architect.
- Must allow for the execution of all functions.
- Establishes the solution vocabulary
- Implicitly represents a level of technology." [Cr2, 1/23, p.16-17]

In the template, concept emerges through specialization. In Figure 2.4, a bold dashed line marks the specialization and enabler links associated with the concept mapping. **System Operating** portrays a process that fulfills the desired function. **Designed Object** and **Supporting Object** comprise the form that "embodies working principles." In Example 2.1 the architect has three concepts from which to select for slowing the spoilage of food. Each is described as a process with an associated form, expressed in OPL as follows:

<p><b>Freezing is Spoilage Slowing.</b> <b>Freezing requires Freezer.</b></p> <p><b>Dehydrating is Spoilage Slowing.</b> <b>Dehydrating requires Dehydrator.</b></p> <p><b>Canning is Spoilage Slowing.</b> <b>Canning requires Cannery.</b></p>
--

By checking Crawley's list it is possible to verify that each of these cases are examples of concepts. Each allows for execution of the desired function, establishes the solution vocabulary, and determines an implicit level of technology for the solution.

### 3. Documenting Requirements, Design Parameters, and Intent in OPM

#### 3.1 Formulating Good Requirements

One of the system architect's most important tasks is to define requirements based on the desires of intended beneficiaries. The distillation of desires into unambiguous and useful requirements is an exercise in communication. It involves the assimilation of knowledge through observation, reading, and listening (as well as, perhaps, tasting, smelling, and touching). It requires the conversion of this "sensed" knowledge into language, and thus is a process inherently susceptible to ambiguity. In fact, the book *Exploring Requirements: Quality Before Design* contends that, "The fundamental problem of requirements definition is ambiguity." [GW, p. 92] However, the definitional discipline of OPM provides a structure for limiting ambiguity in both individual requirement statements and overall system specifications. In the realm of Axiomatic Design it provides assistance in formulating good FRs and DPs.

The task of formulating good requirements is a difficult one, as described in this excerpt from a paper published by the American Institute of Aeronautics and Astronautics:

Any contractor who has tried to respond to a set of requirements knows the difficulty of the task. Many requirement documents contain statements that are not requirements, but are unverifiable goals or objectives. For example, "minimize costs" or "provide adequate margins" are statements of design goals or objectives and should be stated as such. Other statements found in requirements documents are actually statement of work items, such as, "perform trade studies". Some requirements are so grammatically incorrect that they defy interpretation. Conflict and inconsistency between requirements are not unusual. Sets of requirements are often incomplete.

The problem is that most engineers who are assigned the tasks of writing requirements do not know how to do the job. Colleges do not normally

provide training in this aspect of engineering. Everyone gets "on-the-job" training; but often this is without guidance. Most engineers assigned such a task simply obtain an existing requirements document and use it as an example. Often this example is flawed, so the engineer starts with bad information. Over several generations of documents, the problem will compound to the point that any resemblance to valid requirements is purely coincidental. [Ho, p. 2]

The article goes on to list many reasons why engineers typically can't write good requirements including not knowing "what to do," not "understanding why" it is important, desiring to be "doing something else," and seeing "no reward" in it. Although all four of these reasons can be addressed through education, the first can be partially overcome if the engineer has a simple and consistent framework in which to compose requirements. Many advanced software systems are available for developing requirements, but if OPM is to fulfill its promise as a "comprehensive modeling tool" it must be extended to better manage this task. Its framework of formal rules applied in its combination graphic-language environment give it advantages over other systems, but methods for converting OPDs and OPL into practical and useful system requirements lists remain relatively undeveloped.

Providing a framework for formulating good requirements first requires an understanding of the characteristics of good requirements. Ford Motor Company uses the following descriptions for good product requirements in its System Engineering Fundamentals course:

The purpose of requirements is to focus on WHAT the system has to do, not HOW to do it...

Well-written requirements are statements that describe

- Function—what the system must do
- Performance—how well the system must do it
- Constraints—considerations outside of your control, such as carryover plants, parts, methods, restraints on that function, operating environment or usage
- Things surrounding it
- Things that keep it from performing well or to its optimum
- Interfaces—interaction with other systems
- Product Quality (quality of execution)—how well are we going to do it? (i.e., what variation is allowed?)

A well-written requirement is

- Unambiguous—understood by everybody the same way
- Valid—accurately represents true customer, regulatory, corporate wants / musts
- Measurable—it is possible to measure the requirement
- Objective—it can be measured by a technical means not requiring subjective judgment
- Verifiable—we can prove that we meet the requirement
- Repeatable—many measurements produce the same answer
- Correlated—it has got to correlate back to the objective of the original requirement
- As implementation free as possible—do not name the technology used to implement the function, just the function itself [Fo2, p. 67, 68 notes, 69 notes]

These lists provide a description of what well-written requirements *are*, but they don't sufficiently explain *how* to develop them. In *Requirements Engineering: A Good Practice Guide*, Sommerville and Sawyer recommend five guidelines for the process of defining requirements:

- Define standard templates for describing requirements
- Use language simply, consistently and concisely
- Use diagrams appropriately
- Supplement natural language with other descriptions of requirements
- Specify requirements quantitatively [SS, p.141]

These guidelines align extremely well with the principles of OPM, so it is reasonable to believe that OPM can enhance the requirement formulation process. The next section examines this process in the context of formulating FRs, although it should be similar for formulating other types of requirements as well.

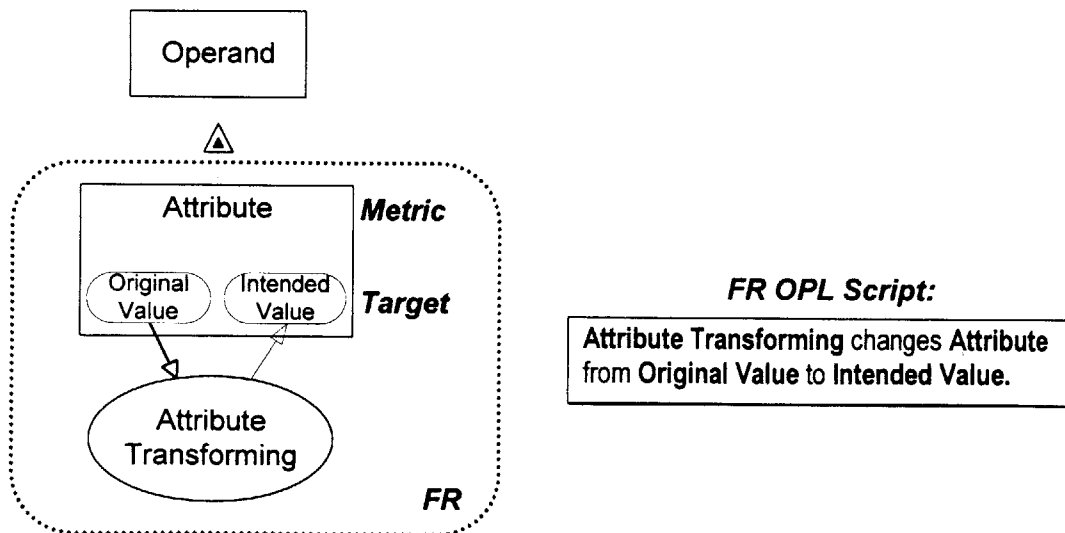
## 3.2 Functional Requirements

According to Suh, FRs should characterize the "functional needs" of the system. This has been clarified to mean the intended use or service to be provided. However, even this description is ambiguous. To be less ambiguous, it is useful to ensure that FRs always answer the two basic questions associated with functional WHATs:

**What should the system affect?**

**What effect should the system cause?**

Based on the OPM description of a WHAT, this means an FR should be associated with some operand object (transformee) and be expressed as a process that changes attribute values of the operand. An OPD output link should identify the intended attribute value. This template for a generic FR is exhibited in Figure 3.1.



**Figure 3.1: A Generic FR Expressed in OPM**

This construct addresses the first four of Sommerville's and Sawyer's guidelines specifically: a template, consistent use of language, a diagram, and a supplement to natural language. The fifth, quantitative specification, is a rule that can be easily adopted for expressing the values in the Attribute object. The template not only helps



clarify which requirements should be considered "functional," it also aligns well with the Ulrich and Eppinger definition of specification: "A specification (singular) consists of a *metric* and a *value*... Note that the value may take on several forms, including a particular number, a range, or an inequality. Values are always labeled with the appropriate units (e.g., seconds, kilograms, joules). Together, the metric and value form a specification." [UE, p.82]

The OPL script generated from FRs in each of the examples in Chapter 2 is compiled in Table 3.1. Note that the OPL sentences do not use the typical language of requirements. This, however, should not pose a great difficulty. OPM users could simply choose to adopt the OPL versions as requirement statements, or a straightforward translator could be developed to formulate these sentences as "requirements." For example the OPL sentence

**Food Preserving changes Edible Duration of Food from short to long.**

could be translated as:

**The Food Preserving process shall change the Edible Duration of Food from short to long.**

In addition, words such as short and long can easily be made more explicit using the concept of specialization, e.g., four days is short, six months is long.

<b>Example</b>	<b>Function</b>	<b>FR Expressed in OPL</b>
2.1	Food Preserving	Food Preserving changes Edible Duration of Food from short to long.
2.2	Material Separating	Separating changes Material from undivided to divided.
2.3	Paper Holding	Holding changes Position of Paper from movable to fixed.  Holding maintains fixed Position of Paper.
2.4A	Crossing Means Creating	Crossing Means Creating changes River from uncrossable to crossable.
2.4B	River Crossing	River Crossing changes Location of Person from Initial Bank to Opposite Bank.

**Table 3.1: FRs of Chapter 2 Examples Expressed in OPL**

In addition to these examples, it is instructive to examine a list of FRs taken from Suh's work. Table 3.2 details a list of systems studied in *Axiomatic Design: Advances and Applications* and the top-level FRs associated with them.

No.	System	FRs
3.1	Freezer Door [Su2, p. 20]	<ol style="list-style-type: none"> <li>1. Provide access to the items stored</li> <li>2. Minimize energy consumption</li> </ol>
3.2	Newcomen Steam Engine [p. 24]	<ol style="list-style-type: none"> <li>1. Extend the piston</li> <li>2. Contract the piston by creating a vacuum in the cylinder</li> </ol>
3.3	Hydraulic Tube Shaping [p. 27]	<ol style="list-style-type: none"> <li>1. Bend a titanium tube to prescribed curvatures</li> <li>2. Maintain the circular cross section of the bent tube</li> </ol>
3.4	Refrigerator [p. 32]	<ol style="list-style-type: none"> <li>1. Freeze food for long term preservation</li> <li>2. Maintain food at cold temperature for short-term preservation</li> </ol>
3.5	Water Faucet [p. 119]	<ol style="list-style-type: none"> <li>1. Control the water flow rate without affecting the water temperature</li> <li>2. Control the temperature of the water without affecting flow rate</li> </ol>

**Table 3.2: FRs from Systems Studied by Suh [Su2]**

Suh provides the following general guidance for stating FRs and DPs: "[A]ll FRs are stated starting out with verbs. This is a good way of distinguishing an FR from a DP, which should start with a noun, if possible." [SCL, p.3] Suh's rule for starting FRs with verbs is observed in these examples, however the variety in formulation of the FR statements is noteworthy. FR1 in Example 3.1 and FR2 in Example 3.3 use verbs that are not very descriptive of function (provide, maintain). FR2 in Example 3.2 includes a HOW. Example 3.4 includes statements of intent that elaborate beyond actual function. Example 3.5 includes limitations on the functions. The systems considered in these examples are all relatively simple, yet no two sets of FRs are stated in quite the same way. If this kind of variety is exhibited in requirement statements of single author, it is easy to imagine the significant dilemmas arising from ambiguity in requirements statements composed by a team of authors for a complex system. OPM's definitional

discipline can help eliminate this ambiguity in FRs. The following discussion examines in more detail the FRs for the examples listed and determine OPM-based versions that reduce the requirements to their essential, unambiguous elements.

### **Example 3.1: Freezer Door**

In Example 3.1 it is important to first establish the context of the requirements. The architect is designing a freezer system. The intentions related to Food Preserving expressed in Example 2.1 have already been considered. The architecture has already been chosen: a freezer—an enclosure in which temperature is maintained below freezing. This architecture requires a source of energy that is consumed to keep the interior of the enclosure cold; it also requires a method for accessing the contents inside.

Suh's FRs for the freezer system capture these requirements generally, but some ambiguity remains. FR1 begins with the phrase "Provide access..." which raises a question of perspective. Is this a requirement posed from the perspective of the architect's development process or from the user's operating process? For instance, in Example 2.4A the intent is to provide a means for crossing, i.e., "Crossing Means Creating"—a development task for the architect. On the other hand, in Example 2.4B the intent is "River Crossing." Is this requirement about providing a means for access or simply accessing? The importance of making this distinction should not be downplayed. Besides the reduction in ambiguity that results when these distinctions are clarified, these choices determine how to set up the Axiomatic Design Matrix and how the system is modeled in OPM (as evidenced by the differences in Examples 2.4A and 2.4B).

From the user's perspective, the two questions are posed:

*What should the system affect?*

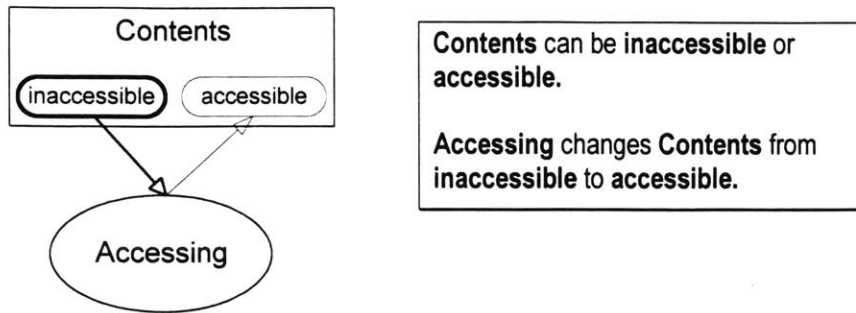
*What effect should the system cause?*

For FR1, the answers to these questions (without specifying design solutions) are

*The accessibility of the contents should change.*

*The contents should be made accessible.*

Expressed in OPL, the FR becomes

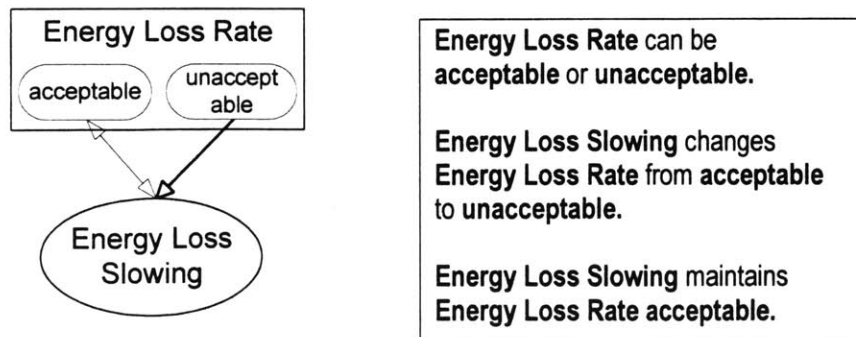


**Figure 3.2: FR1 from Example 3.1 Formulated in OPM**

FR2 requires minimizing the energy consumption of the freezer. The concern is about energy consumed due to heat transfer into the freezer, designated as *energy loss* in this example. To formulate the FR consistently, it is again useful to ask the questions:

*What should the system affect?* It should affect energy loss. Energy loss occurs over time, so the system should affect the *rate* of energy loss.

*What effect should the system cause?* The desire is to minimize energy loss. Minimizing results from reducing or slowing, so the system should *slow* the rate of energy loss (e.g., from unacceptable to acceptable levels).



**Figure 3.3: FR2 from Example 3.1 Formulated in OPM**

By asking the questions and formulating OPM-based FRs, each of the other examples can be developed in a similar way.

### Example 3.2: Newcomen Engine

FR1: "Extend the piston."

*What should the system affect?* It should affect the position of the piston.

*What effect should the system cause?* The position of the piston should change from the bottom of the cylinder to the top of the cylinder.

FR2: "Contract the piston by creating a vacuum in the cylinder."

*What should the system affect?* It should affect the position of the piston.

*What effect should the system cause?* The position of the piston should change from the top of the cylinder to the bottom of the cylinder.

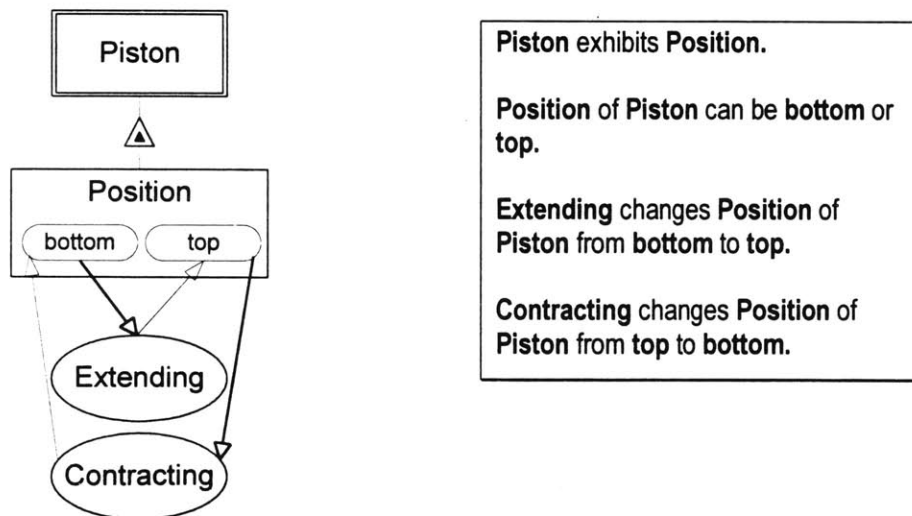


Figure 3.4: FRs from Example 3.2 Formulated in OPM

### Example 3.3: Hydraulic Tube Shaping

A challenge in this example is that FR2 uses the verb "maintain." This word does not indicate an effect, i.e., a change that is inherent in a process. Trying to understand exactly what effect should occur will help the architect do a better job of determining architecture and communicating requirements.

FR1: "Bend a titanium tube to prescribed curvatures."

*What should the system affect?* It should affect the curvature of the tube.

*What effect should the system cause?* It should bend the tube, changing its curvature from its initial to its prescribed position.

FR2: "Maintain the circular cross section of the bent tube."

*What should the system affect?* The bending process in FR1 creates a force that distorts the cross sectional shape of the tube. The system should affect force that distorts the cross sectional shape.

*What effect should the system cause?* It should prevent distortion, i.e., change the distorting force from an unacceptable to an acceptable level.

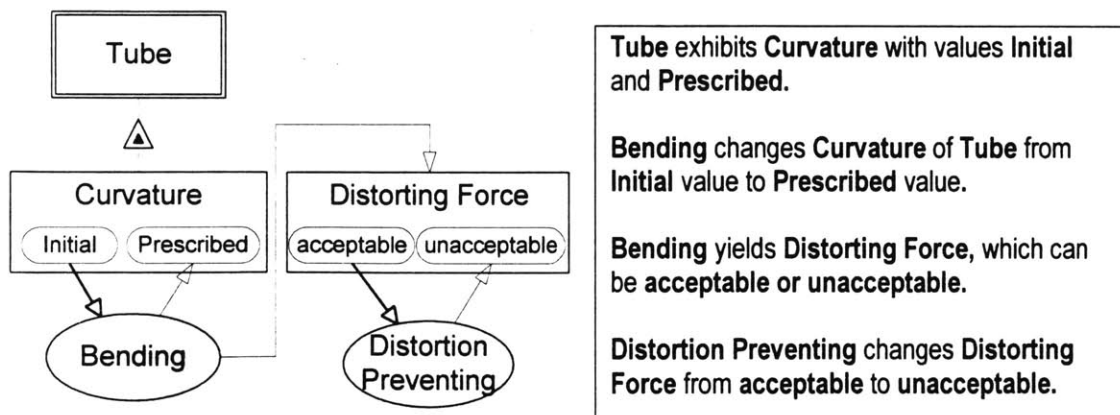


Figure 3.5: FRs from Example 3.3 Formulated in OPM

### Example 3.4: Refrigerator

FR1: "Freeze food for long term preservation."

*What should the system affect?* It should affect the food, in particular the rate at which it spoils.

*What effect should the system cause?* It should change the spoilage rate from fast to very slow (several months).

FR2: "Maintain food at cold temperature for short-term preservation."

*What should the system affect?* It should affect the spoilage rate of food.

*What effect should the system cause?* It should change the spoilage rate from fast to slow (several days).

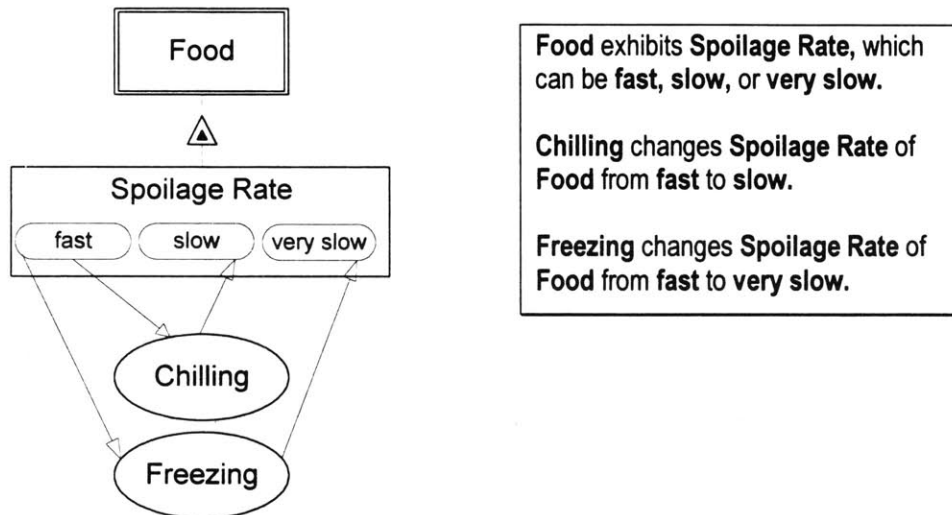


Figure 3.6: FRs from Example 3.4 Formulated in OPM



### Example 3.5: Water Faucet

FR1: "Control the water flow rate without affecting the water temperature."

*What should the system affect?* It should affect the water flow rate.

*What effect should the system cause?* It should change the water flow rate from "maintained" to "changed."

FR2: "Control the temperature of the water without affecting flow rate."

*What should the system affect?* It should affect the water temperature.

*What effect should the system cause?* It should change the water temperature from "maintained" to "changed."

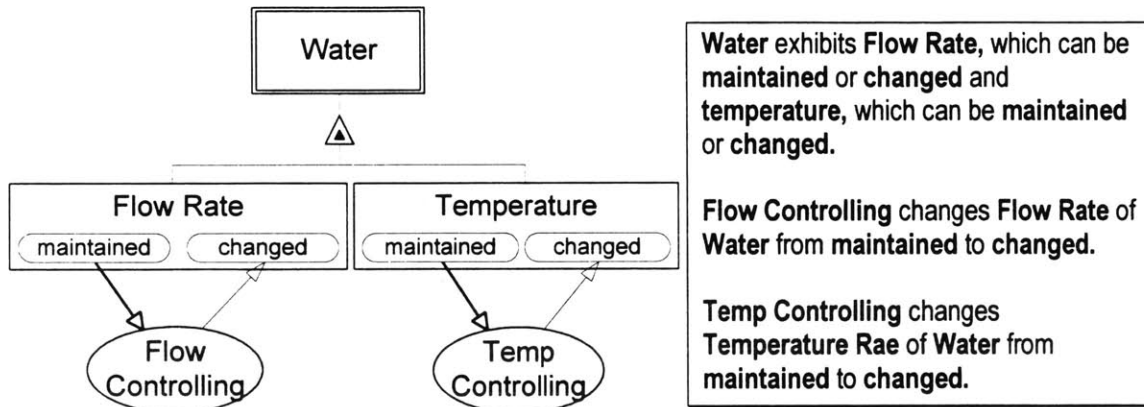


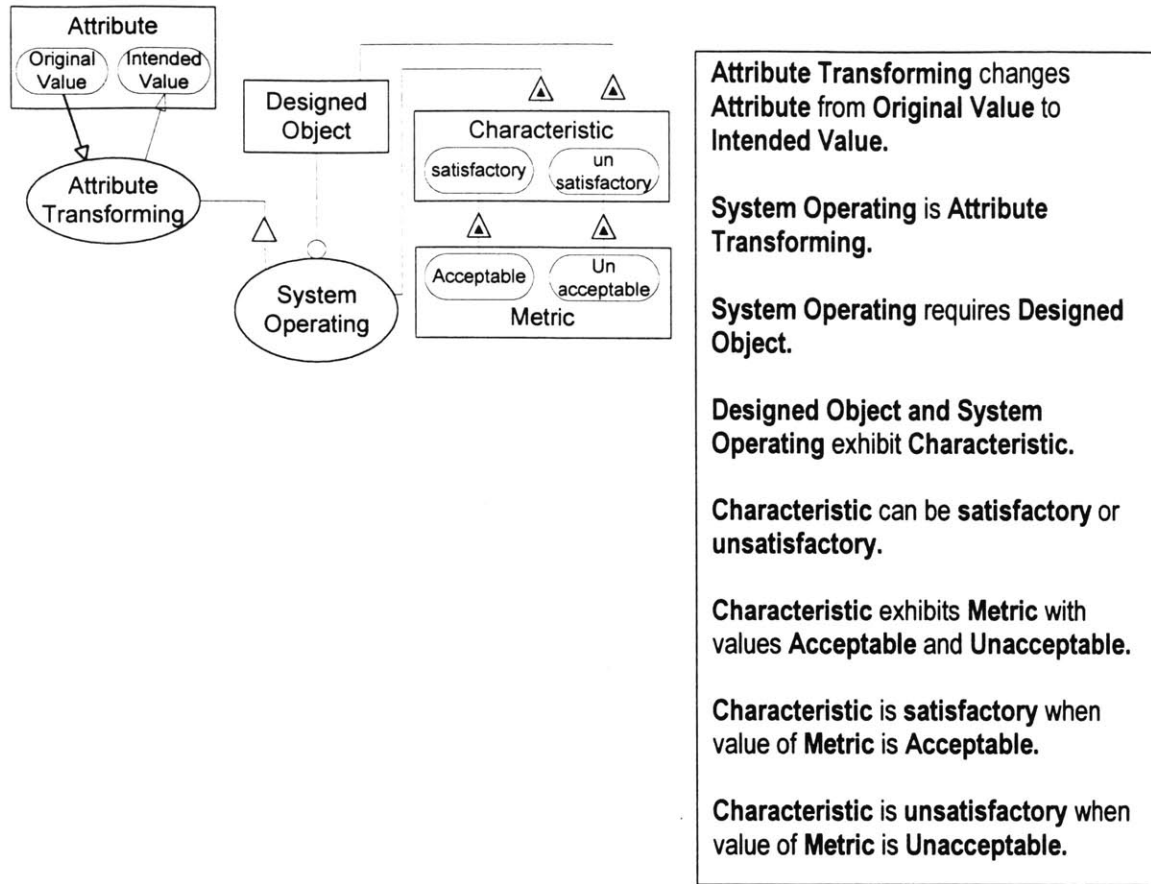
Figure 3.7: FRs from Example 3.5 Formulated in OPM

### 3.3 Constraints

Not all system requirements are FRs. In Axiomatic Design Suh categorizes system requirements as either FRs or Constraints. Other methodologies do not necessarily make the same distinctions. For example, QFD does not distinguish "functional" WHATs from "non-functional" WHATs. On the other hand, some authors divide requirements into several categories, with FRs being one among many. In any case, the utility of OPM for formulating standard, unambiguous requirements is not limited to formulating FRs. It can be useful for representing constraints and other requirement types as well as distinguishing between functional and non-functional requirements.

Otto and Wood provide a useful rule of thumb for distinguishing constraints from functions. "A constraint is a statement of a clear criterion that must be satisfied by a product and requires consideration of the entire product to determine the criterion value...Functions are satisfied by subsets of the product through their operation; constraints are satisfied by properties of the entire product." [OW, p. 152] Thus, unlike a function, a constraint is not something a system "does." Also, unlike the ability to add functionality to a system by adding a new assembly or part, one typically can't meet a constraint by adding a new part to a system.

This description helps clarify how a constraint should be represented in OPM. Since a constraint is not something satisfied by what a system does, it will not necessarily include a *process* in its OPM representation. Instead, the representation will be typically be structural in nature, expressed through OPM's exhibition-characterization relation. Furthermore, the constraint may be attached to a process or an object:

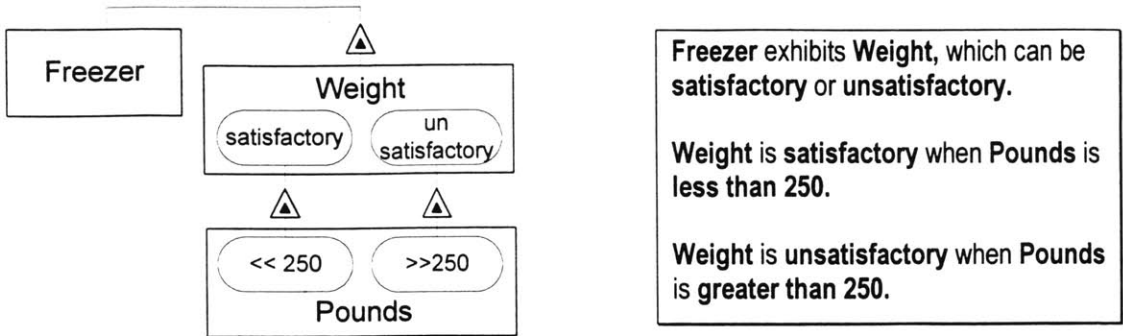


**Figure 3.8: A Generic Constraint Expressed in OPM**

Suh considers two types of constraints

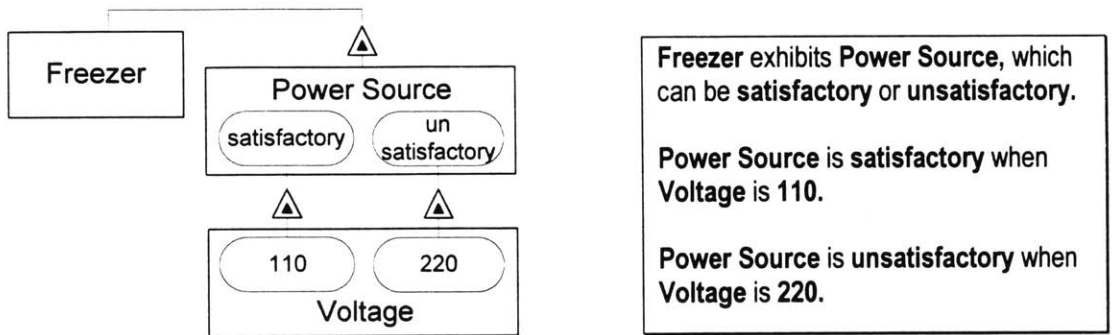
- *Input Constraints:* constraints in design specifications, e.g., bounds on size, weight, material, cost.
- *System Constraints:* imposed by the system in which the design solution must function, e.g., interfacial bounds, geometric shape, machine capacities, laws of nature. [Su1, p. 39]

Suh also notes that constraints do not normally have tolerances associated with them, although they may have an associated upper or lower bound. An example of the first kind of constraint for the weight of the freezer system is shown below.



**Figure 3.9: Weight Constraint for a Freezer**

An example of the second type of constraint, relating to the kind of power the freezer must use is shown below.



**Figure 3.10: Power Source Constraint for a Freezer**

### 3.4 Design Parameters

The actual design and development of a complex system requires decomposition of the WHATs and HOWs into a useful hierarchy. In the QFD framework, HOWs at an upper level become the WHATs at the next lower level. Suh calls this process "zigzagging" because one must move repeatedly back and forth between answering the WHATs and answering the HOWs. He describes the process in the following way:

To decompose FR and DP characteristic vectors, we must zigzag between the domains. That is, we start out in the 'what' domain and go to the 'how' domain... From an FR in the functional domain, we go to the physical domain to conceptualize a design and determine its corresponding DP. Then, we come back to the functional domain to create FR1 and FR2 at the next level that collectively satisfies the highest-level FR. FR1 and FR2 are the FRs for the highest level DP. Then we go to the physical domain to find DP1 and DP2 by conceptualizing a design at this level, which satisfies FR1 and FR2, respectively. This process of decomposition is pursued until the FR can be satisfied without further decomposition... [Su2, p.30]

What exactly is a DP? Suh defines DPs as "key physical variables (or other equivalent terms in the case of software design, etc.) in the physical domain that characterize the design that satisfies the specified FRs." [Su2, p. 14] Even though Suh describes DPs as HOWs, this definition actually makes DPs a subset of HOWs, because properly formulated HOWs describe both behavior and structure. Nevertheless, in describing the zigzagging decomposition process, Suh speaks of finding DPs by "conceptualizing a design." For this conceptualization to be useful, it will need to provide answers to the two basic questions associated with functional HOWs:

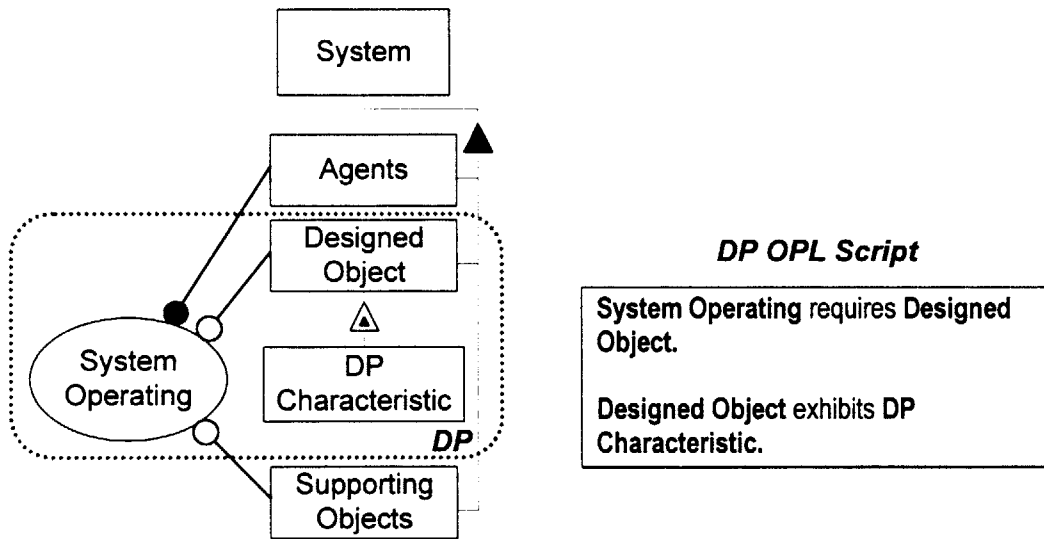
**How does the system behave?**

**How is the system structured?**

In casual descriptions it is fair to use the term DP for the system solution arrived at through this conceptualization. However, Axiomatic Design still requires the identification of a DP as a particular characteristic of the system that can be evaluated in

the design matrix (Equation 1.1). In fact, in their most limited sense, FRs and DPs are both characteristics that can be quantified and expressed as variables. This is how they are represented in the design matrix in which elements are partial derivatives of FRs with respect to DPs.

The implication of this for system decomposition in OPM is that good models will include all elements of properly formulated WHATs and HOWs as well as identification of the characteristics and effects that should be represented in the design matrix. Thus, a template for a DP in OPM should be embodied in the template for a HOW. In addition, the particular characteristic that affects the FR metric should be represented through an exhibition link. A template for a generic DP is exhibited in Figure 3.11.



**Figure 3.11: A Generic DP Expressed in OPM**

The template includes the key elements of a HOW and its associated DP, but does not prescribe the manner in which they must be combined in a particular model. This template is not as general as the FR template. **Agents** and **Supporting Objects** may or may not need to be indicated in a particular OPD. In practice, DPs might be considered to be a characteristic these other system elements. However, the name

"Design Parameter" implies it should be a characteristic of the object designed by the architect. Failing to adhere to this practice can cause confusion. For example, one of the difficulties in modeling DPs in OPM is distinguishing between processes that occur during system design and processes that occur during system operation. For example, Example 3.1 presented a freezer door system for which Suh states the FRs as

- FR1: Provide access to the items stored
- FR2: Minimize energy consumption.

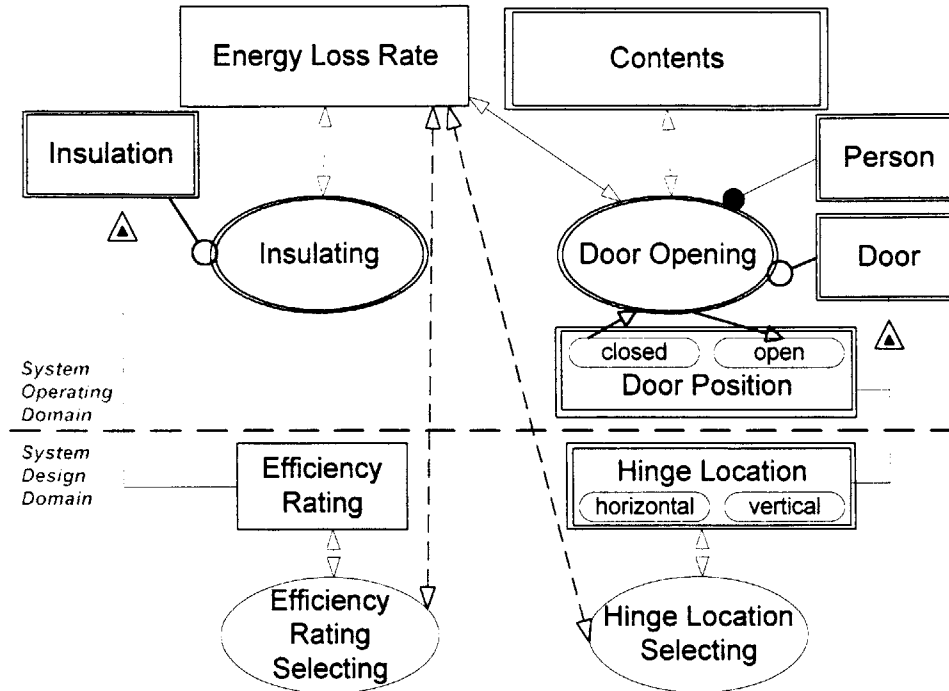
He identifies the following, corresponding DPs for a particular design:

- DP1: Vertically hung door
- DP2: Thermal insulation material in the door. [Su2, p.20]

The door itself fulfills FR1, but its hinge location (vertical) has an effect on FR2.

The actual parameter for which an architect must select a value is hinge location.

Similarly, although DP2 is stated as "insulation material," it should be understood that the architect selects the material based on its insulating properties. The actual variable is an efficiency rating that corresponds to the material's insulating capability. The process of selecting settings for these parameters occurs during design. One way to represent this appears in Figure 3.12. Note that the FR-related processes **Energy Loss Slowing** and **Accessing** introduced in example 3.1 are specialized in this diagram to **Insulating** and **Door Opening** based on the DPs selected.



**Figure 3.12: OPD Combining System Design and Operating Domains**

**Contents, Insulation, Door, Person, Hinge Location, and Door Position are physical objects.**

**Insulating and Door Opening are physical processes.**

**Insulating affects Energy Loss Rate.**

**Insulating requires Insulation.**

**Insulation exhibits Efficiency Rating.**

**Door exhibits Door Position and Hinge Location.**

**Door Opening affects Contents and Energy Loss Rate.**

**Door Opening changes Door Position from closed to open.**

**Door Opening requires Door.**

**Person handles Door Opening.**

**Efficiency Rating Selecting affects Efficiency Rating and Energy Loss Rate.**

**Hinge Location Selecting affects Hinge Location and Energy Loss Rate.**



Processes carried out by the architect during the design phase appear in the lower portion of the diagram labeled "System Design Domain." The diagram highlights the flexibility of OPM. Both the system design domain and operating domain can be represented in a single model. This flexibility, however, can also be a source of confusion if the domain boundaries are not clearly understood. Processes that occur during design may be mistaken for processes that occur during system operation; and objects that can vary between states during design may be mistakenly thought to vary during system operation.

This situation is related to the imprecision in the use of the term "DP." During design, the DP Characteristic has various states that can be assigned by the architect. It is truly a parameter. In the design matrix, the partial derivative  $\partial FR / \partial DP$  is expressed as a formula or parametric relationship. Once the design is frozen and a state is assigned, it is common to continue referring to the characteristic and its assigned state as a DP. The column heading in the design matrix is still called a "DP" even though the partial derivative entry is a fixed value.

OPM models of system operation will be different depending on which states are selected for DPs because each state essentially represents a different design. Figure 3.12 shows that **Hinge Location Selecting**, which occurs in the system design domain, can ultimately affect **Energy Loss Rate** in the system operating domain (the effect link is dashed because it does not have to be identified in the OPD; it occurs implicitly through OPM inheritance). Whether or not the effect actually occurs depends on the value of **Hinge Location** selected by the architect. Section 4.1 will present OPDs for both hinge locations. Though the elements of the OPDs are similar, the number and placement of effects links are different. This allows a comparison of the two designs and an evaluation based on the Independence Axiom.

### 3.5 Preserving Intent

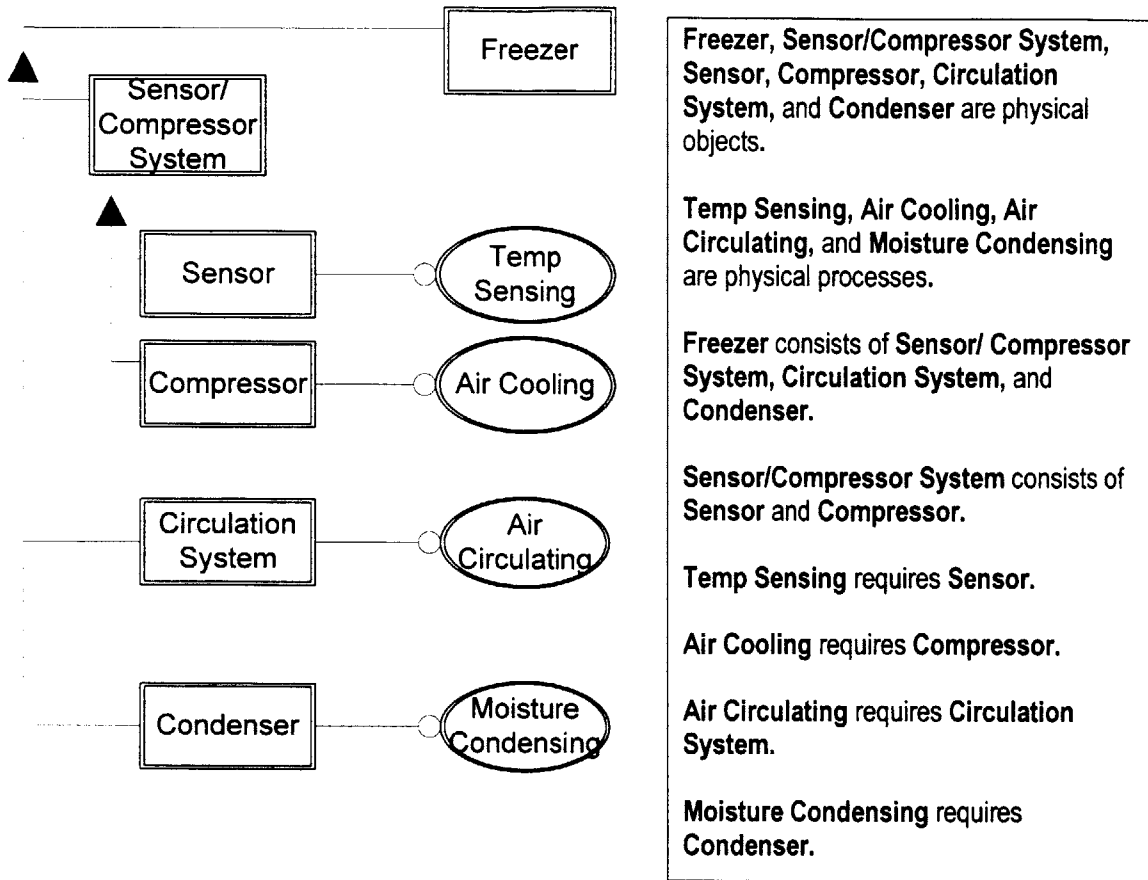
One of the strengths of Axiomatic design is that intent is explicitly captured in FRs at every step of the system decomposition and success is based on achieving this intent according to defined criteria. This helps overcome a major design problem, described by Suh: "One of the major problems in design is that designers do not state explicitly the FRs that their design must satisfy. They try to design intuitively. They also do not recognize the probable need to reiterate the establishment of FRs until a satisfactory design results." [Su1, p. 32]

Because there is such great flexibility in the ways systems can be modeled in OPM, there is danger that the architect who uses OPM for system development will experience this problem. Based on intuition it is possible to represent a very elaborate system in OPM without explicitly recording requirements and their associated intent. In the engineering world, an architect is more often than not given the task of "re-engineering" a design. In this situation it is common to omit from system documentation items that describe intent. Typically this is not a purposeful omission. Intent often is often simply obvious to the engineer in the context of the engineering task and its documentation seems extraneous. However, good system documentation not only describes elements of the system, it also explains their purpose.

Sommerville and Sawyer recommend recording requirement rationale for all requirements: "The rationale associated with a requirement is a link between the problem and the requirements for the proposed solution. The rationale makes it easier for readers to understand the requirement and to assess the impact of changes to the requirement. Problem experts can use the rationale to check if the requirement is consistent with the problem being solved." [SS, p. 87]

Using the four questions for system decomposition and OPM templates for WHATs and HOWs introduced in this thesis can provide the documentation of rationale described by Sommerville and Sawyer. However, repeated use of the characterization link between FR-related processes and DP-related processes may indeed become tedious and may not always be necessary. A shorthand mechanism for capturing the flow of intent in OPM is the function box that allows identification of functional intent, without having to show both the more general functional process and the specialized system behavior process every step of the way. This idea can be illustrated through a deeper examination of Suh's Refrigerator Example (Example 3.4). [Su2, p. 32]

Suppose that an engineer was not provided the system decomposition of FRs and DPs listed by Suh. By examining the freezer section of an existing refrigerator, the engineer would be able to model the freezing system based on reverse engineering. The resulting OPD might be similar to Figure 3.13.



**Figure 3.13: OPD of a Freezing System based on Reverse Engineering**

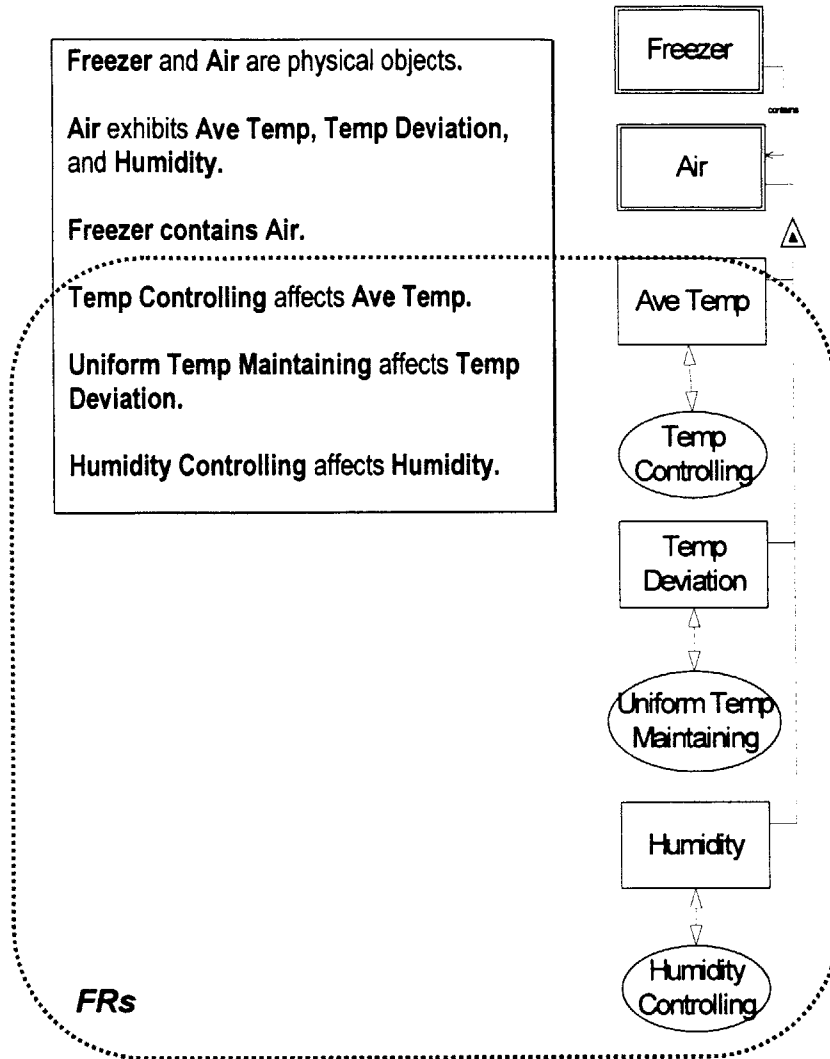
The diagram and script show how the system works. The existence of a sensor and compressor indicates temperature sensing and air cooling processes. A circulation system indicates air circulating, and a condenser indicates moisture condensing. The intent of these systems may be obvious to those familiar with freezers, but the diagram does not capture it. How would the diagram look if it were developed along with the system, in conjunction with Axiomatic Design decomposition?

Suh provides the following decomposition of the freezing system:

- FR11: Control temperature of the freezer section in the range of  $-18\text{C} \pm 2\text{C}$
- FR12: Maintain a uniform temperature throughout the freezer section at the preset temperature
- FR13: Control humidity of the freezer section to relative humidity of 50%

- DP11: Sensor/compressor system that turns the compressor on (off) when the air temperature is higher (lower) than the set temperature in the freezer section
- DP12: Air circulation system that blows air into the freezer section and circulates it uniformly throughout the freezer section at all times
- DP13: Condenser that condenses the moisture in the returned air when its dew point is exceeded.

The OPD in Figure 3.14 of the freezer system shows how the initial definition of FRs 11, 12, and 13 could be captured. Freezer Air exhibits three characteristic attributes that correspond to important system metrics. Effect links connect each of these characteristics to processes that will fulfill the FRs. The effect links could be drawn to connect directly to the parent object, Freezer Air, but this would not clearly indicate which characteristics are affected by which processes.



**Figure 3.14: OPD of Freezing System Functional Requirements**

The next step is to expand this diagram to include DPs. In this step, the original and somewhat general functional processes that express intent are specialized to system-specific processes. This is illustrated in Figure 3.15.

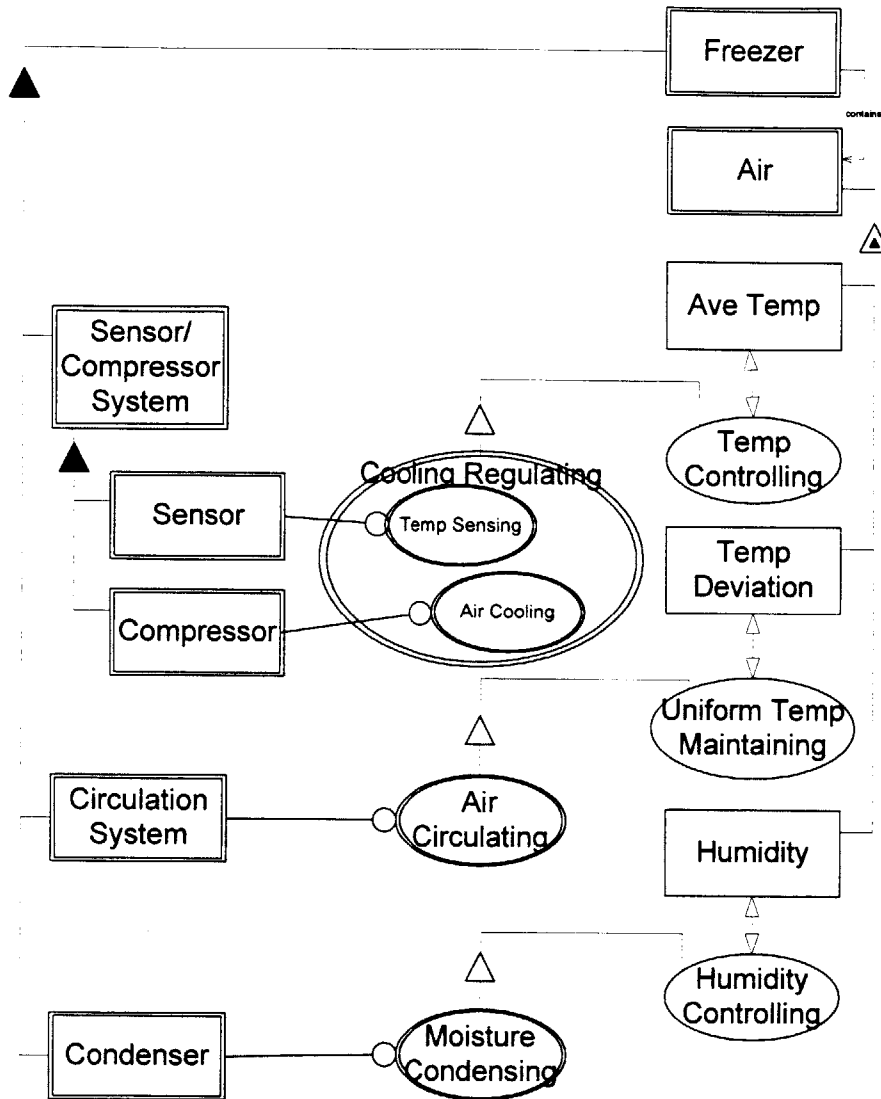


Figure 3.15: OPD of a Freezing System including Intent

**Freezer, Sensor/Compressor System, Air, Sensor, Compressor, Circulation System, and Condenser** are physical objects.

**Temp Sensing, Cooling Regulating, Air Cooling, Air Circulating, and Moisture Condensing** are physical processes.

**Freezer** consists of **Sensor/Compressor System, Circulation System, and Condenser**.

**Sensor/Compressor System** consists of **Sensor and Compressor**.

**Air** exhibits **Ave Temp, Temp Deviation, and Humidity**.

**Freezer** contains **Air**.

**Temp Controlling** affects **Ave Temp**.

**Cooling Regulating** is **Temp Controlling**.

**Cooling Regulating** zooms into **Air Cooling and Temp Sensing**.

**Temp Sensing** requires **Sensor**.

**Air Cooling** requires **Compressor**.

**Uniform Temp Maintaining** affects **Temp Deviation**.

**Air Circulating** is **Uniform Temp Maintaining**.

**Air Circulating** requires **Circulation System**.

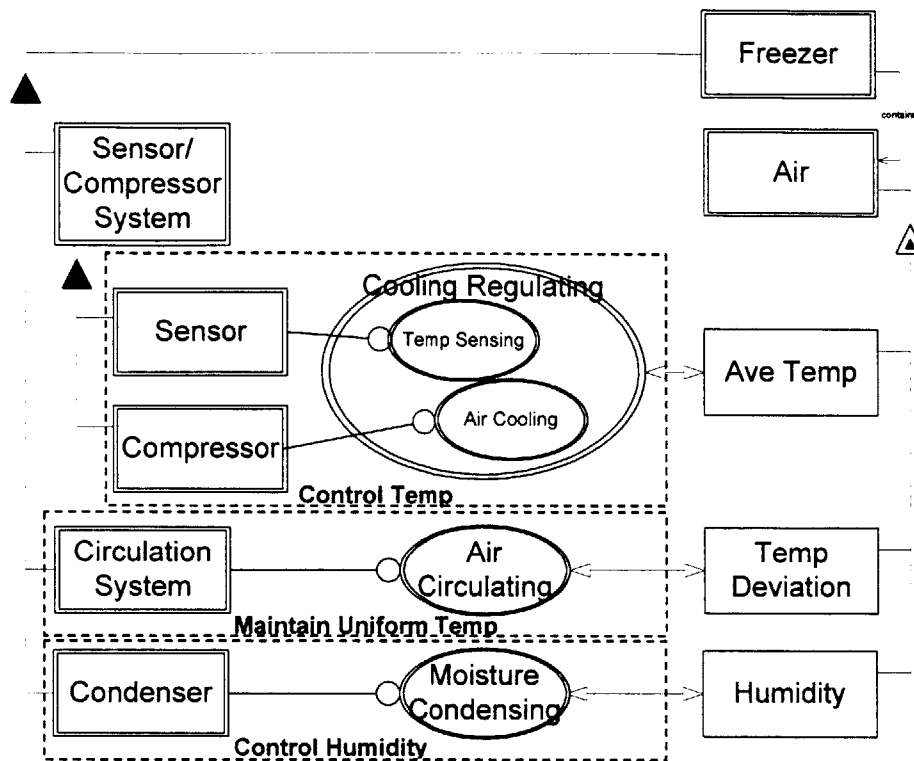
**Humidity Controlling** affects **Humidity**.

**Moisture Condensing** is **Humidity Controlling**.

**Moisture Condensing** requires **Condenser**.

In the design depicted, **Air Circulating** *is how* **Uniform Temp Maintaining** is to be accomplished. Once this is decided, **Air Circulating** could replace **Uniform Temp Maintaining** altogether in the OPD, *but this would omit the original intent*. An alternative is to preserve intent using Dori's function boxes, [Do2] as illustrated in Figure 3.16.





**Figure 3.16: OPD of a Freezing System with Intent Capture using Function Boxes**

The addition of these function boxes to the OPD corresponds to the following additions in the OPL Script:

Function **Control Temp** is achieved by **Cooling Regulating** as well as **Sensor** and **Compressor**.

Function **Maintain Uniform Temp** is achieved by **Air Circulating** as well as **Circulation System**.

Function **Control Humidity** is achieved by **Moisture Condensing** as well as **Condenser**.

This kind of documentation preserves the intent of system architecture that may become obscured through time or transfer to new audiences. However, the use of a function box may become cumbersome for complex systems. Other methods for capturing intent have been proposed. For example, Crawley has created a specialized

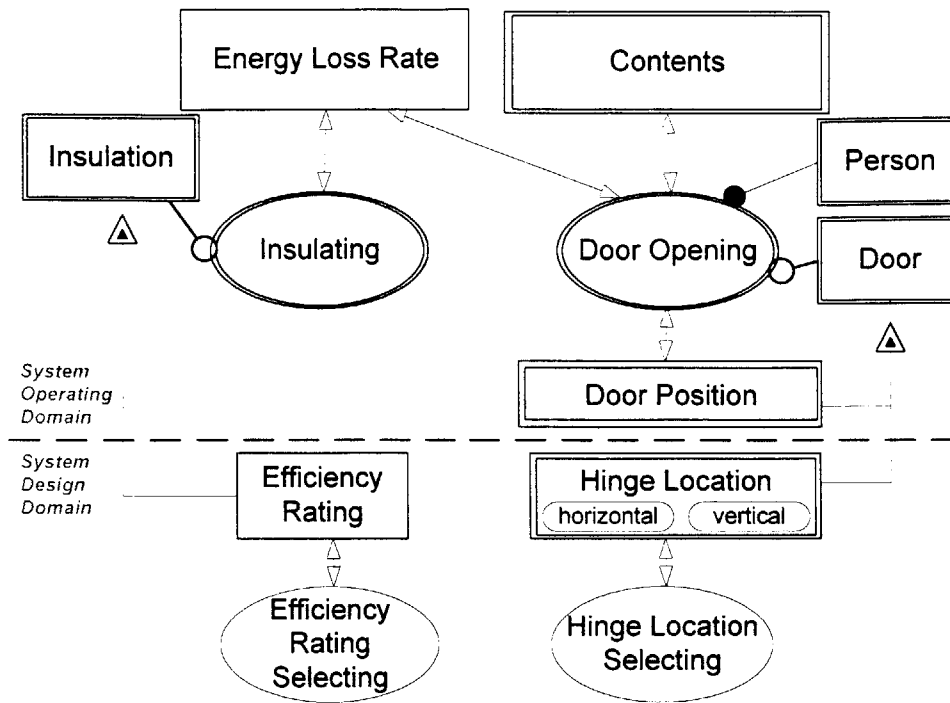
"intent object" for representing system intent. This has the advantage of a compact representation that avoids issues of nesting in OPDs.

## **4. Evaluating Independence via OPM**

Suh's Independence and Information Axioms provide a systematic basis for evaluating designs. Designs chosen based on the axioms are considered good designs in the Axiomatic Design framework. For designs represented in OPM, is it possible to recognize adherence to the axioms? Can patterns be identified that will help OPM modelers recognize potential design problems? This section answers these questions affirmatively for the Independence Axiom. The best way to begin the explanation is through examples.

### **4. 1 Freezer Door Example**

Suh's Freezer Door system (Example 3.1) is again a useful example. It provides a simple illustration of the importance of independence and its expression in OPM. Figure 4.1 captures the system in a slightly simplified version of Figure 3.12. The evaluation of whether or not the system adheres to the Independence Axiom occurs in the portion of the OPD representing the System Operating Domain; however, processes in the System Design Domain affect the results of the evaluation.



**Figure 4.1: Combined Domain OPD for Refrigerator Door System**

A typical refrigerator has a vertically hung door that provides access to refrigerator contents but also affects energy loss by letting cool air escape when the door is open. Table 4.1 shows the design matrix for the given FRs:

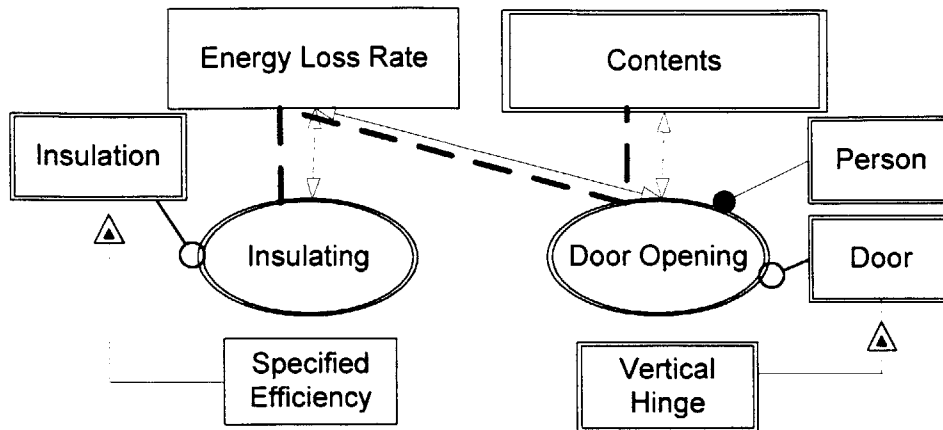
- FR1: Provide access to the items stored
- FR2: Minimize energy consumption.

	Insulating Material	Access via Vert. Door
FR1	X	X
FR2	0	X

**Table 4.1: Design Matrix for Vertically Hung Refrigerator Door**

Using this design matrix, the design appears. However, it is not an acceptable design, because the door must be opened to gain access to the contents, and this causes an unacceptable energy loss rate. The coupling appears in the system OPD in

the form of an effect link path connecting the two FR-related objects. This path is highlighted in Figure 4.2, which shows the System Operating Domain part of Figure 3.12 after values for the DPs have been selected.



**Figure 4.2: OPD for Refrigerator with Vertically Hung Door and Highlighted Effect Path**

**Contents, Insulation, Door, Person, and Vertical Hinge are physical objects.**

**Insulating and Door Opening are physical processes.**

**Insulating and Door Opening affect Energy Loss Rate.**

**Door Opening affects Contents and Energy Loss Rate.**

**Insulating requires Insulation.**

**Insulation exhibits Specified Efficiency.**

**Door exhibits Vertical Hinge.**

**Door Opening requires Door.**

**Person handles Door Opening.**

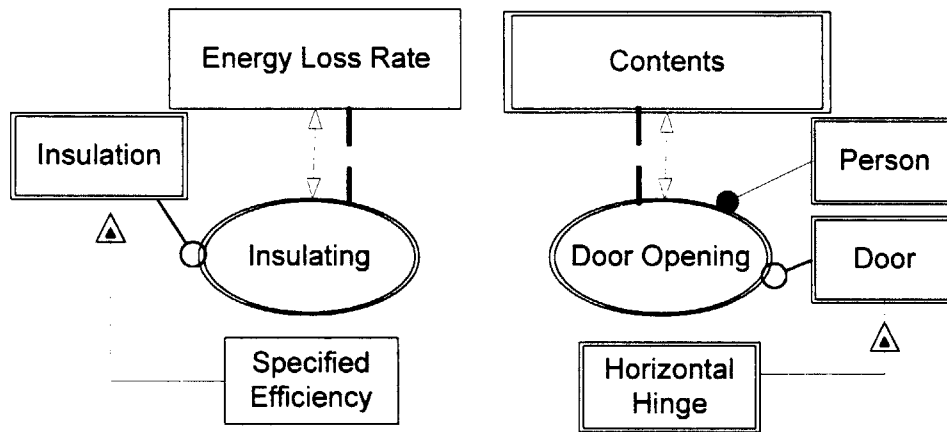
How can this coupling be remedied with a simple design change? One suggestion given by Suh is to change the DP based on door hinge location. If the door is attached with horizontal hinges (like a deep freeze), energy loss when the door is

opened is likely to meet the functional requirement because cool air does not rise out of the refrigerator very quickly. Thus the design matrix shows an uncoupled design.

	Insulating Material	Access via Horiz. Door
FR1	X	0
FR2	0	X

**Table 4.2: Design Matrix for Horizontally Mounted Refrigerator Door**

The corresponding OPD for this design, does not contain the effect link path between the two FR-related objects:



**Figure 4.3: OPD for Refrigerator with Horizontally Hung Door and no Effect Path**

**Contents, Insulation, Door, Person, and Horizontal Hinge** are physical objects.

**Insulating** and **Door Opening** are physical processes.

**Insulating** affects **Energy Loss Rate**.

**Insulating** requires **Insulation**.

**Insulation** exhibits **Specified Efficiency**.

**Door** exhibits **Horizontal Hinge**.

**Door Opening** requires **Door**.

**Door Opening** affects **Contents**.

**Person** handles **Door Opening**.

Figures 4.2 and 4.3 are examples of two OPDs corresponding to two different design decisions by the architect. They are different outcomes of the processes indicated in the "System Design Domain" in Figure 3.12. In the first case, the architect has assigned the **Hinge Location** DP a value of "vertical;" in the second case, a value of "horizontal." The consequences for independence are visually perceptible in the resulting OPDs.

## 4.2 Water Faucet Example

A classic example in the Axiomatic Design literature is the architecture of a water faucet. Figure 4 illustrates two possible designs. Design A has a hot water valve and a cold water valve. Design B has valve for mixing hot and cold water to control temperature and a separate valve for controlling flow.

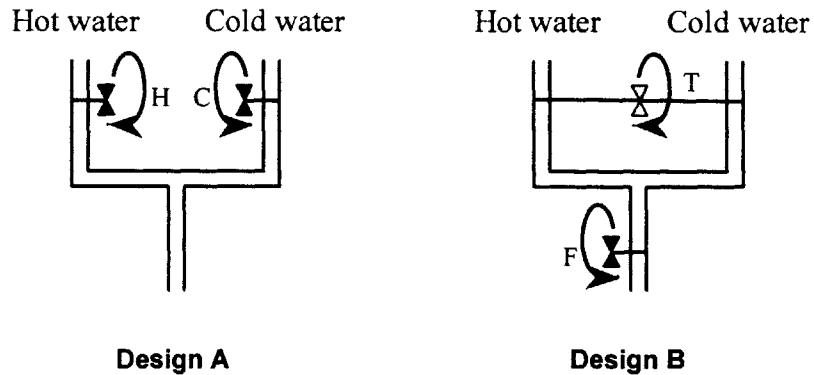


Figure 4.4: Two Possible Architectures for a Water Faucet System

Typical functional requirements for a water faucet are

- FR1: Control the temperature of water
- FR2: Control the flow of water.

Design A is a poor design by Axiomatic Design standards, because the DPs selected—a hot valve and a cold valve—violate the Independence Axiom. This is illustrated in Table 4.3, which documents that adjusting either valve affects both functional requirements.

	H valve	C valve
FR1	X	X
FR2	X	X

Table 4.3: Design Matrix for Coupled Water Faucet, Design A



It is instructive to examine how this design might be represented in OPM. Figure 4.5 shows an OPD for Design A that the system architect might develop if he/she were modeling the system from scratch. The top portion captures the FRs; the bottom portion captures the DPs and associated system operation. Concern should arise when **Flow Controlling** and **Temp Controlling** each specialize to two processes—in fact to the *same two processes*. The OPL script identifies this situation:

**C Valve Adjusting** and **H Valve Adjusting** are **Flow Controlling**.

**C Valve Adjusting** and **H Valve Adjusting** are **Temp Controlling**.

Two HOWs jointly fulfill two separate WHATs. This is a violation of the independence axiom.

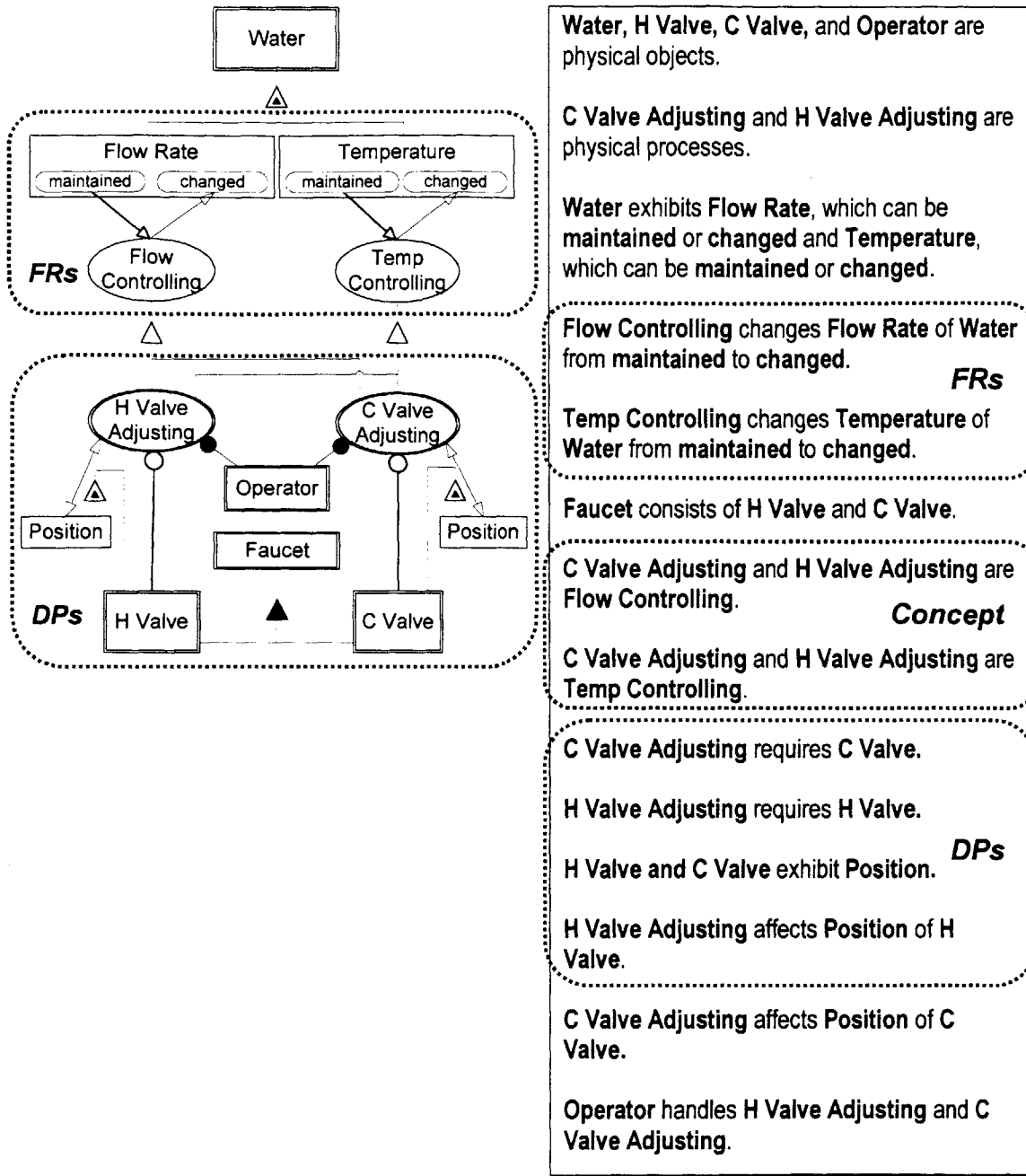


Figure 4.5: OPD and OPL Script for Water Faucet Design A—Design Intent

A verbal indication of coupling in the OPL script is valuable information for the system architect, but it may not always be obvious—embedded as it is in a long OPL paragraph. However OPM provides visual clues from the OPD as well as logical rules for inheritance that reveal coupling more clearly. As a case of specialization, the OPD shows that **H Valve Adjusting** is both **Temp Controlling** and **Flow controlling**.

Because **Temp Controlling** affects **Temperature** and **Flow Controlling** affects **Flow Rate**, the inheritance rules for specialization in OPM dictate that **H Valve Adjusting** must affect both **Flow Rate** and **Temperature**. Similarly, **C Valve Adjusting** must affect both **Flow Rate** and **Temperature**. The OPD in Figure 4.4 does not explicitly include effect links that represent these relationships because it is a first iteration of the system, based on *intent*. However, as the OPD is expanded and refined to clearly detail actual system *operation*, these links will automatically appear when **Temp Controlling** and **Flow Controlling** are replaced by their specializations: **H Valve Adjusting** and **C Valve Adjusting**.

The OPD in Figure 4.6 shows these links along with the **Faucet Operating** process. This OPD is yet another good example of OPM's ability to bridge many domains. True to Dori's vision, it captures structure and behavior in one diagram. In fact, as the annotations in Figure 4.6 indicate, the diagram captures structure of both the function operand and system as well as the behavior of both the system and operator. It would be possible to also show the architect's intent by including function boxes that indicate **Temp Controlling** and **Flow Controlling** (as described in Chapter 3, this is a good practice for documenting system development) but this would make the diagram unnecessarily complex for the current discussion.

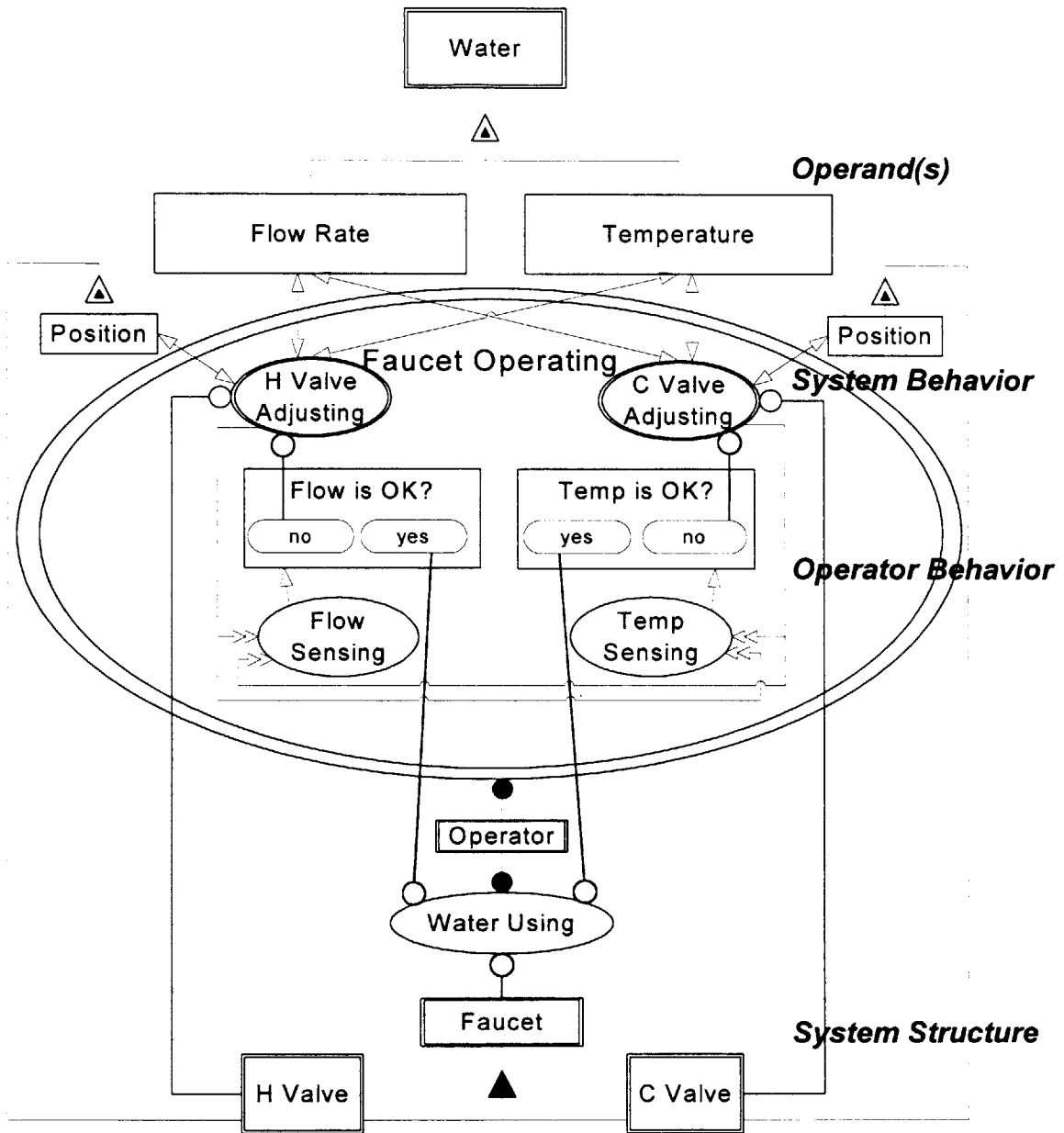


Figure 4.6: OPD for Water Faucet Design A—System Operation

**Water, Valve, Operator, H Valve, and C Valve** are physical objects.

**C Valve Adjusting, H Valve Adjusting, and Faucet Operating** are physical processes.

**Water** exhibits **Flow Rate** and **Temperature**.

**Faucet Operating** zooms into **H Valve Adjusting, C Valve Adjusting, Temp Sensing** and **Flow Sensing**, as well as 'Temp is OK?', and 'Flow is OK?'.  
**Flow Sensing** determines whether **Flow is OK**.  
**Temp Sensing** determines whether **Temp is OK**.

**C Valve Adjusting** and **H Valve Adjusting** affect **Flow Rate**.

**C Valve Adjusting** and **H Valve Adjusting** affect **Temperature**.

**C Valve Adjusting** affects **Flow Rate, Temperature, and Position of C Valve**.

**H Valve Adjusting** affects **Flow Rate, Temperature, and Position of H Valve**.

**H Valve Adjusting** invokes either **Flow Sensing** or **Temp Sensing**.

**C Valve Adjusting** invokes either **Flow Sensing** or **Temp Sensing**.

**Flow Sensing** determines whether **Flow is OK**.

**Temp Sensing** determines whether **Temp is OK**.

**Water Using** occurs if **Flow is OK** and **Temp is OK**.

**Operator** handles **Faucet Operating** and **Water Using**.

**Water Using** requires **Faucet**.

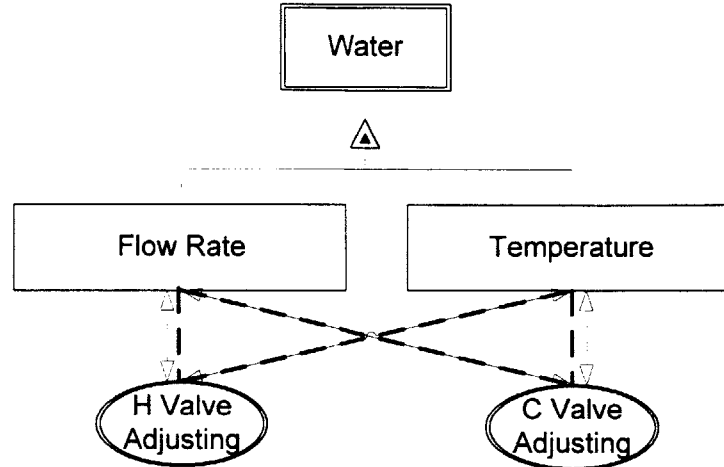
**Faucet** consists of **C Valve** and **H Valve**.

**C Valve Adjusting** requires **C Valve**.

**H Valve Adjusting** requires **H Valve**.

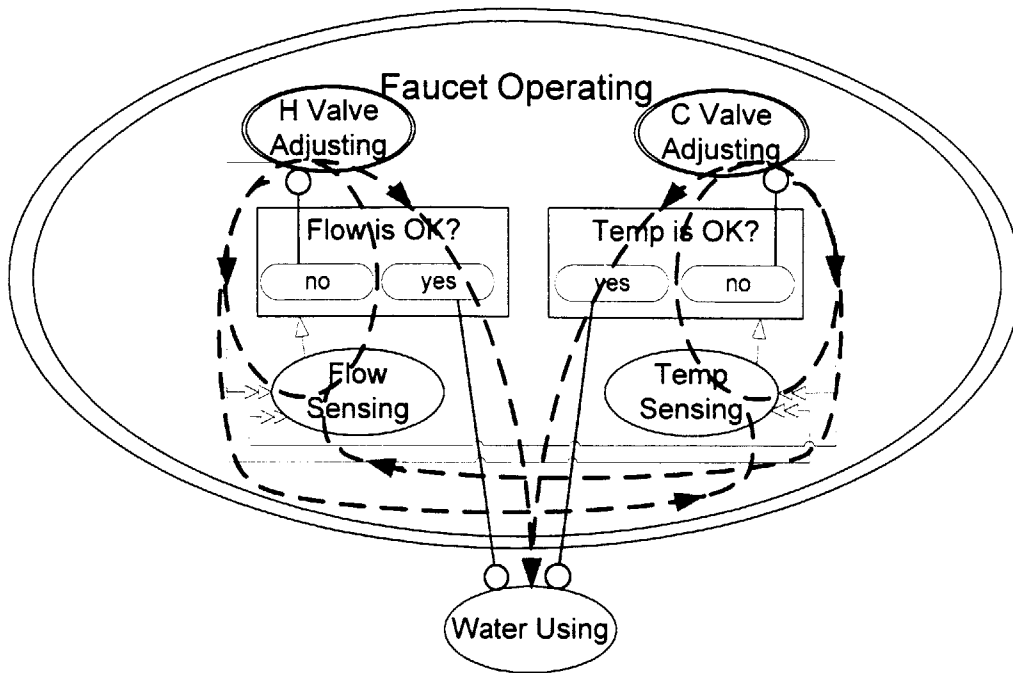
A striking aspect of this OPD and OPL is the complexity required to represent what most people consider a fairly simple system and operation. However, it is important to realize that much of this complexity arises from the fact that the design is coupled. Furthermore it is possible to identify patterns in the OPD associated with this coupling. For example, Figure 4.7 highlights a circuit of effect links that connects the

FR-related objects and the DP-related processes. This circuit is an OPM manifestation of a coupled design.



**Figure 4.7: Circuit of Effect Links for Water Faucet Design A**

A second manifestation of coupling appears in the **Faucet Operating** process in which checking processes occur iteratively—perhaps repeated several times—until the desired flow and temperature are achieved. In this case the OPD path through the process is a circuit of effect and instrument links that proceed serially and iteratively: The operator adjusts a valve, checks the flow and adjusts for the flow, checks the temperature and adjusts for the temperature. But the adjustment for temperature affects the flow, so the operator must repeat the process again until the flow and temperature are sufficiently close to the desired levels.



**Figure 4.8: Serial, Iterative Faucet Operating Process, Design A**

Processes comprised of a series of checks that require repeated iteration indicate complexity if not outright coupling. However, water faucet Design B removes the need for repeated iteration by introducing valves that individually control only one of the desired functions. The design matrix for Design B shows no coupling.

	T valve	F valve
FR1	X	0
FR2	0	X

**Table 4.4: Design Matrix for Uncoupled Water Faucet, Design B**

Corresponding to this is an OPD for Design B that is cleaner than the OPD for Design A. Figure 4.9 show the diagram, which is free of the and extra affect links and invocation links that appeared in the OPD for Design A.

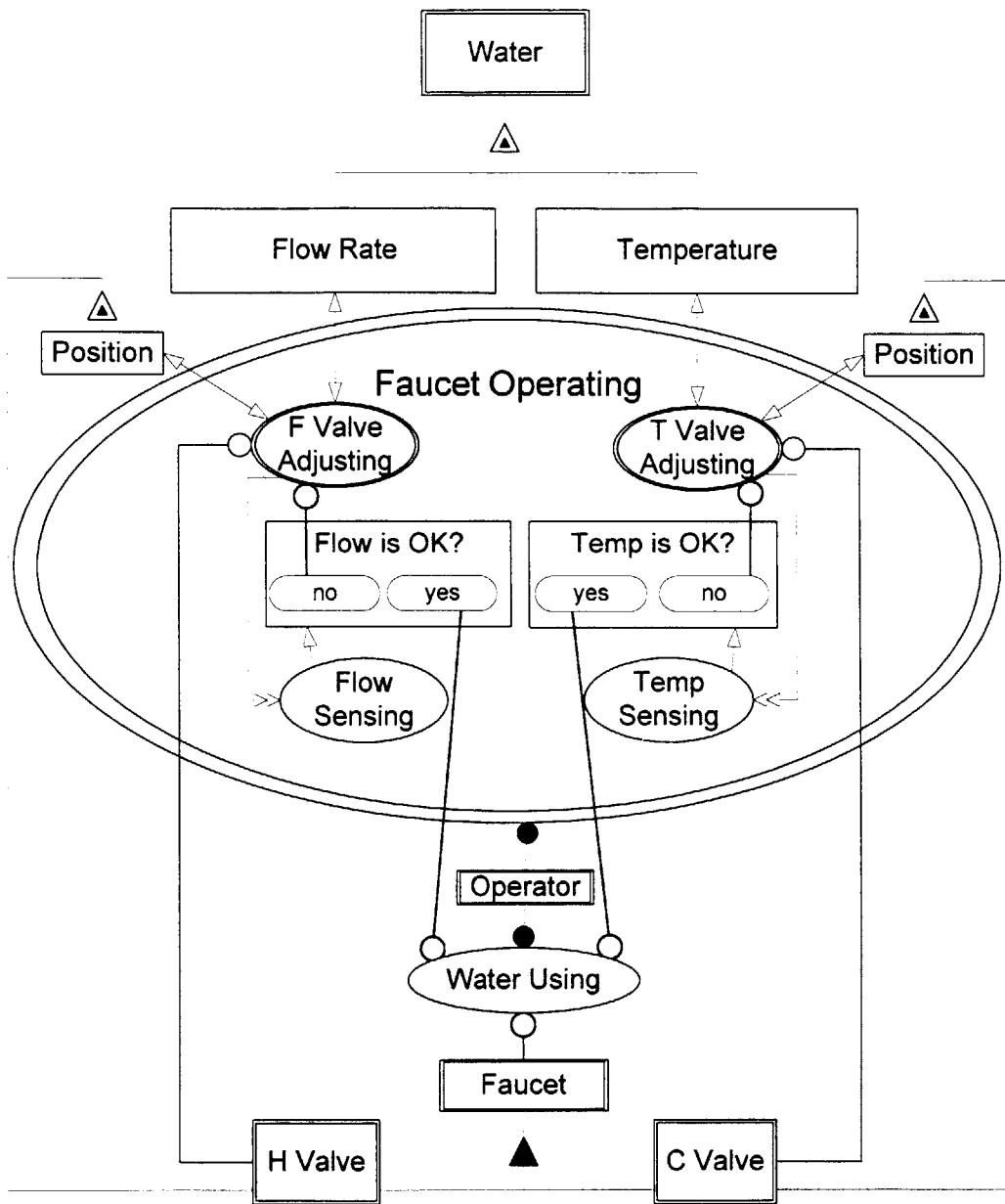


Figure 4.9: OPD for Design B



**Water, Operator, Valve, F Valve, and T Valve are physical objects.**

**F Valve Adjusting, T Valve Adjusting, and Faucet Operating are physical processes.**

**Water exhibits Flow Rate and Temperature.**

**Faucet Operating zooms into F Valve Adjusting, Flow Checking, T Valve Adjusting, and Temp Sensing, as well as 'Flow is OK?' and 'Temp is OK?'**

**F Valve Adjusting affects Flow Rate and Position of F Valve.**

**T Valve Adjusting affects Temperature and Position of T Valve.**

**F Valve Adjusting invokes Flow Sensing.**

**Flow Sensing determines whether Flow is OK.**

**T Valve Adjusting invokes Temp Sensing.**

**Temp Sensing determines whether Temp is OK.**

**Water Using occurs if Flow is OK and Temp is OK.**

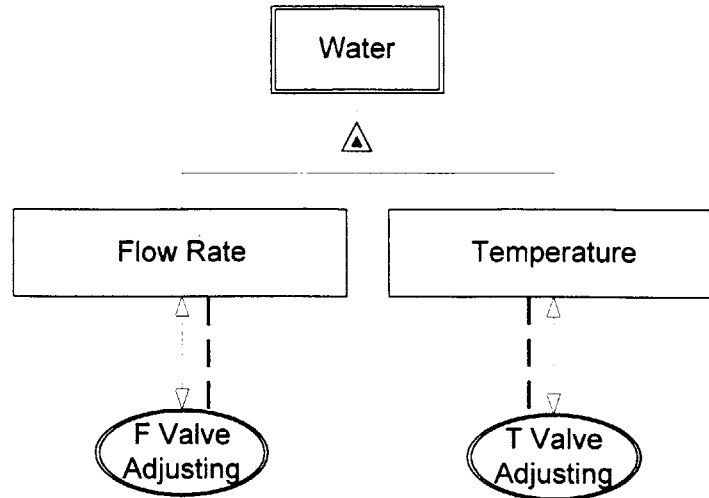
**Operator handles Water Using and Faucet Operating.**

**Faucet consists of F Valve and T Valve.**

**F Valve Adjusting requires F Valve.**

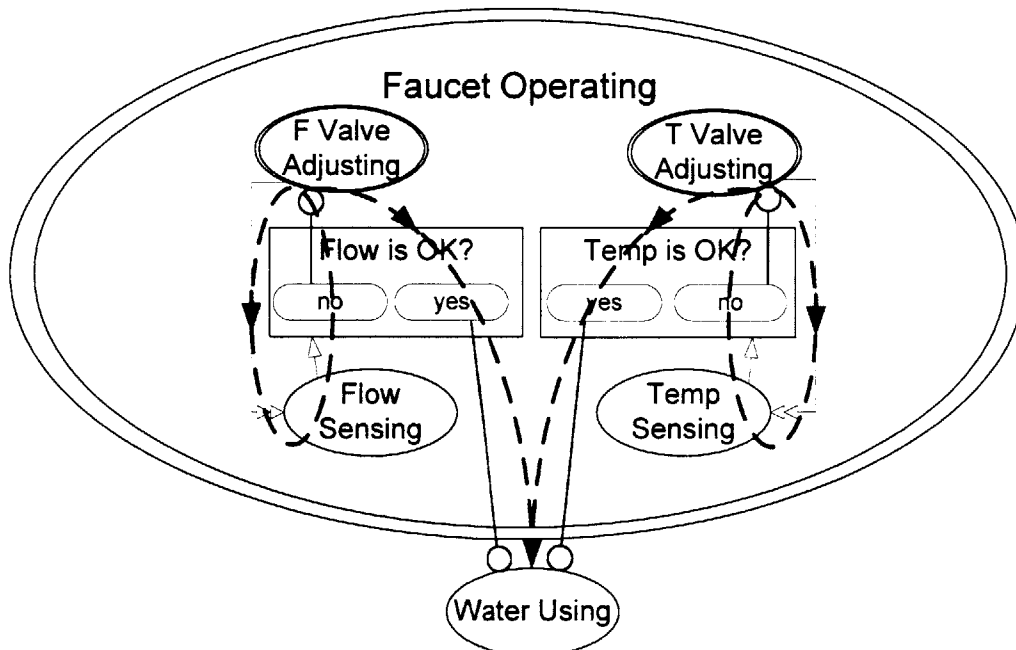
**T Valve Adjusting requires T Valve.**

The OPD for Design B in Figure 4.9 includes no circuit of effect links similar to the circuit in the OPD for Design A. Figure 4.10, highlights this simple, disconnected relationship—a contrast to the circuit of connections highlighted in Figure 4.7.



**Figure 4.10** Portion of OPD for Water Faucet Design B, No Effect Link Circuit

Finally, Figure 4.11 highlights the parallel nature of the Faucet Operating process. It is composed of operations that can be carried out simultaneously without affecting each other or causing iterative adjustments.



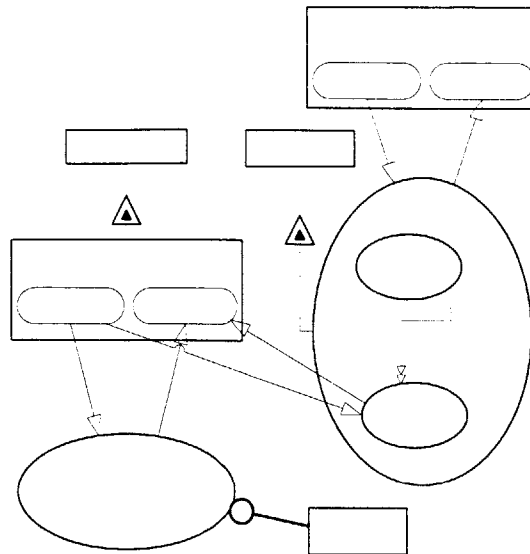
**Figure 4.11:** Parallel Faucet Operating Process, without Effect Link Loop, Design B

### 4.3 Generalization of Coupling in OPDs

Suh's work in Axiomatic Design demonstrates the importance of avoiding coupling in designs. The design matrix is an abstract representation of a design that indicates the presence of coupling, but developing a useful design matrix requires a good understanding of the functions desired as well as the system proposed for fulfilling them. OPM is a tool for developing this understanding in terms of objects and processes and their relationships. Examples presented in this thesis show that coupling may be detected by analyzing OPDs. This analysis does not replace use of the design matrix; however it does provide help for system architects who use OPM. They should be cautious of designs that include effect paths or loops between objects. While such paths may not always indicate coupling of FRs, they do indicate complexity that may cause complications in achieving the desired results.

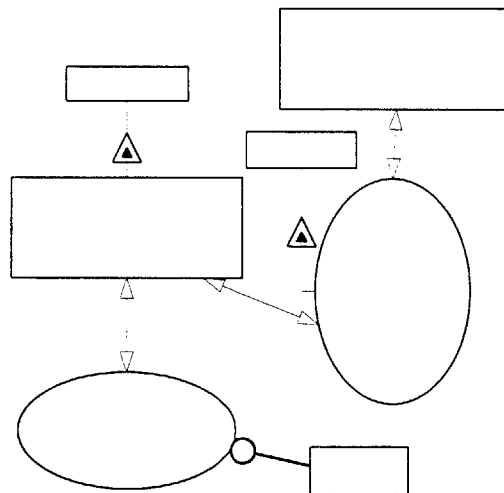
The procedure for evaluating coupling in OPDs begins with reducing the OPD to a basic form that includes objects and processes and procedural links but omits object states and unnecessary structural links. This is accomplished by suppressing states in objects, folding objects, and zooming out of processes. Once simplified in this manner the OPD is easier to evaluate. Within the OPD is a subset graph comprised of objects and processes connected by transformation links. This subset is a "bipartite" graph—a graph comprised of two classes of nodes (objects and processes) in which each link has one end in the first class and one end in the second. Paths in this graph that connect two objects via a process indicate coupling in a general sense. If FR-related objects and DP-related processes are identified within the bipartite graph, paths that connect FR objects to DP processes correspond to x's in the design matrix. This procedure is

illustrated in the following sequence of diagrams. Beginning with an initial OPD structure represented in Figure 4.12.



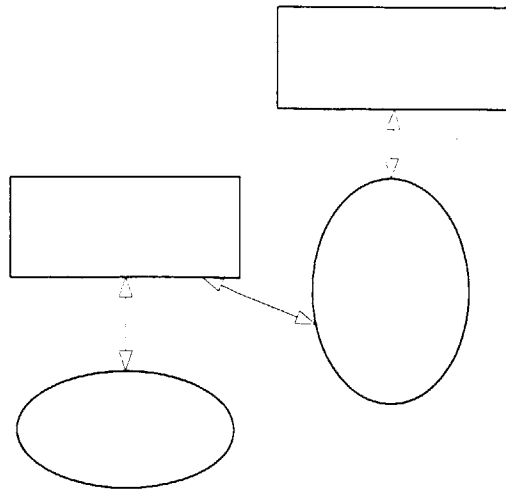
**Figure 4.12: Initial OPD Structure**

This initial structure is simplified by suppressing states, folding objects, and zooming out of processes.



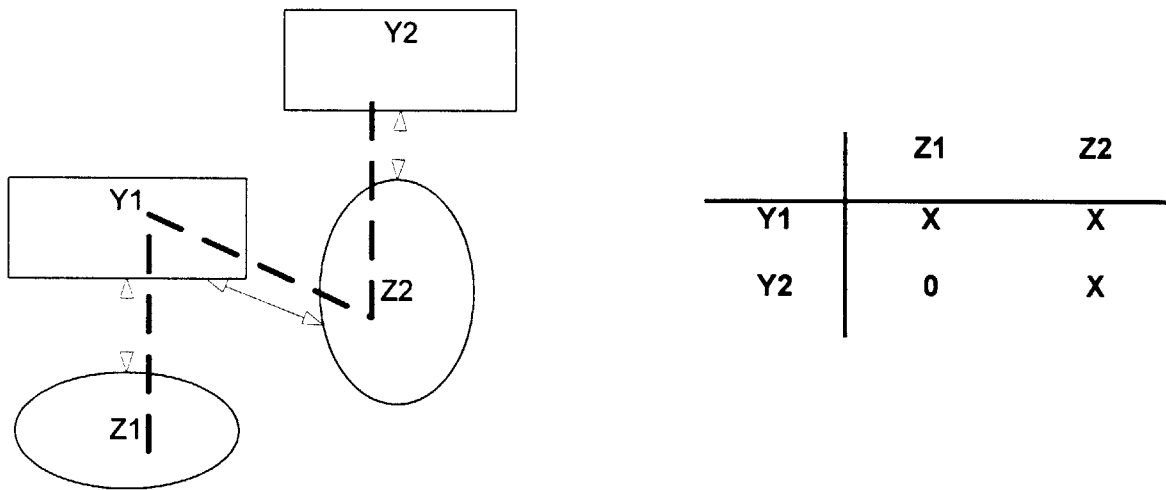
**Figure 4.13: Step 1, Simplified OPD Structure**

Within this simplified OPD structure, the subset, bipartite graph consisting of objects, processes, and effect links is identified.



**Figure 4.14: Step 2, Subset Bipartite Graph**

Paths are now identified that indicate effects that connect processes to objects and objects to objects.



**Figure 4.15: Step 3, Coupling Identified via the Graph and Corresponding Matrix**

## 4.4 Conclusion

This thesis has shown that there is a rich relationship between descriptive and evaluative methods for representing systems. In particular, it has demonstrated that there are synergies between the descriptive techniques of OPM and the evaluative framework of Axiomatic Design.

Using the definitional framework of OPM, a standard method for representing system function and architecture has been presented. This method includes the decomposition of both WHATs and HOWs into a combination of objects and processes. Using OPM templates, such combinations are represented in a standard, repeatable way. The link that connects a WHAT to a HOW corresponds to a concept mapping.

Within templates for WHATs and HOWs, the specific elements of FRs and DPs can be identified. The language and symbols of OPM made explicit in the templates guide good FR and DP formulation. Templates can also be useful for representing constraints in OPM and distinguishing constraints from FRs. Examples illustrate how the FR-DP decomposition can be expressed using OPM by specializing an FR-related process describing intent to a DP-related process describing behavior.

Finally, this thesis begins the exploration of how OPD patterns may reveal adherence to Suh's Independence Axiom. Rules for identifying paths of links that connect objects and form loops are presented. These paths and links correspond to coupling that appears in the Axiomatic Design Matrix.

Several of the topics included here have been developed only far enough to provide a starting point for further study. Many open areas of research in OPM and its synergy with Axiomatic Design remain. Some of these include:

- Refinements of OPM templates to more clearly represent a system in both its Design Domain and Operating Domain and distinguish between these domains.
- Alignment of the "concept" mapping with elements of OPM and development of a representation for concept in the templates.
- Examination of the applicability of the templates for "zig-zagging" through the *entire* system decomposition, beginning with system goals and ending with process parameters.
- Determination of the best way to represent intent in OPDs.
- More complete representation of constraints and other non-functional requirements through OPM.
- Formulation of a more formal definition for DPs in Axiomatic Design.
- Development of formal graph theory-based rules for identifying coupling in OPDs.
- Identification of general OPD relationships that may indicate coupling or provide measures of system complexity.
- Study of OPM's relationship to the Information Axiom.

## BIBLIOGRAPHY

- [ASI] American Supplier Institute, *Quality Function Deployment for Products*, 1995.
- [AHD] *American Heritage Dictionary of the English Language*, Fourth Edition, Houghton Mifflin, 2000.
- [CST] *Chambers Science and Technology Dictionary*, W & R Chambers Ltd. and Cambridge University Press, 1988.
- [Cr1] Crawley, Edward, Lecture Notes for MIT Course ESD.34J, Systems Architecture, January and Fall 2000.
- [Cr2] Crawley, Edward, Lecture Notes for MIT Course ESD.34J, Systems Architecture, January and Fall 2001.
- [De] De Champeaux, Dennis, *Object-Oriented System Development*, Addison-Wesley, 1993.
- [Do1] Dori, Dov, Lecture Notes for MIT Course 16.982, Systems Development with Object Process Methodology, Summer 2000.
- [Do2] Dori, Dov, *Object-Process Methodology: A Holistic Systems Development Paradigm*, to be published by Springer-Verlag, 2002.
- [Fo1] Ford Motor Company, *Failure Mode & Effects Analysis Handbook*, version 3.1, Dearborn, MI, 2000.
- [Fo2] Ford Motor Company, *Systems Engineering Fundamentals*, version 2.0, Ford Technical Education Program, Ford Design Institute, Dearborn, MI, 1997.
- [GW] Gause, Donald C., and Weinberg, Gerald M., *Exploring Requirements: Quality Before Design*, Dorset House Publishing, Inc., New York, 1989.
- [GP] Guinta, Lawrence R. and Praizler, Nancy C., *The QFD Book*, AMACOM, a division of American Management Association, New York, 1993.
- [HC] Hauser, John R. and Clausing, Don, "The House of Quality," *Harvard Business Review*, Vol. 66, No. 3, May-June, 1988, pp.63-73.
- [Ho] Hooks, Ivy, "Why Johnny Can't Write Requirements," American Institute of Aeronautics and Astronautics, Space Programs and Technology Conference, Huntsville, Alabama, ©AIAA, 1990.
- [JJ] James, Robert C., and James, Glenn, *Mathematics Dictionary*, 3<sup>rd</sup> Edition, D. Van Nostrand Company, Inc., 1968.



- [Kr] Krumhauer, P., *Rechnerunterstützung für die Konzeptphase der Konstruktion*, Diss. TU Berlin, 1974, D 83.
- [OW] Otto, Kevin N., and Wood, Kristin L., *Product Design: Techniques in Reverse Engineering and New Product Development*, Prentice Hall, 2001.
- [OED] Oxford English Dictionary, 2<sup>nd</sup> Ed, online version, 1989.
- [PB] Pahl, Gerhard, and Beitz, Wolfgang III, *Engineering Design: A Systematic Approach*, Translated by Ken Wallace, Springer-Verlag London Limited, 1996.
- [RM] Rechtin, Eberhart and Maier, Mark W., *The Art of Systems Architecting*, CRC Press, 1997.
- [RMC] Revelle, Jack B., Moran, John W., and Cox, Charles A., *The QFD Handbook*, John Wiley & Sons, 1998.
- [Sh] Shmuller, Joseph, *Sam's Teach Yourself UML in 24 Hours*, Sam's Publishing, 1999.
- [SS] Sommerville, Ian, and Sawyer, Pete, *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, Chichester, England, 1997.
- [Su1] Suh, Nam P., *The Principles of Design*, Oxford University Press, 1990.
- [Su2] Suh, Nam P., *Axiomatic Design: Advances and Applications*, Oxford University Press, New York, 2001.
- [SCL] Suh, Nam P., Cochran, David S., and Lima, Paulo C., "Manufacturing System Design" *CIRP Annals*, Vol. 47, No. 2, January, 1998
- [UE] Ulrich, Karl T. and Eppinger, Steven D., *Product Design and Development*, 2<sup>nd</sup> Ed., McGraw-Hill, 2000.
- [W] Wright, Frank Lloyd, *Quotes: The Official Website of Frank Lloyd Wright*, <http://www.cmgww.com/historic/flw/quotes.html>, © 1996-2001, Frank Lloyd Wright Foundation c/o <http://www.cmgworldwide.com>.

## BIOGRAPHICAL NOTE

Nathan R. Soderborg  
1221 Snyder Avenue  
Ann Arbor, Michigan 48103

### Professional Experience

- 1994- Ford Motor Company, Dearborn, Michigan  
2002
- 1991- Decision Consultants Incorporated, Southfield, Michigan  
1994

### Educational Experience

- 1986- Ph.D. in Mathematics, University of Michigan, Ann Arbor, Michigan  
1991
- 1981- B.S. in Mathematics, Brigham Young University, Provo, Utah  
1986

### Patents

- 1994 *System and Method for Processing Test Measurements from a Combustion Engine for Diagnostic Purposes*, with K. Marko and B. Bryant. U.S. Patent 5,361,628, November 8, 1994.
- 1994 *System and Method for Filtering a Misfire Detection Data Stream to Yield Optimum Measurement of Misfire Rate*, with J. James and T. Feldkamp. U.S. Patent 5,305,635, April 26, 1994.

### Peer Reviewed Mathematics Publications

- 1994 *A Characterization of Domains Quasiconformally Equivalent to the Unit Ball*, Michigan Mathematical Journal. **41**, (1994) 363-370.
- 1994 *An Ideal Boundary for Domains in  $n$ -Space*, Ann. Acad. Sci. Fenn. (Annals of the Finnish Science Academy) Series A. I. Mathematica **19** (1994) 147-165.
- 1991 *Quasiregular Mappings with Finite Multiplicity and Royden Algebras*, Indiana University Journal of Mathematics **40** (1991) 1143-1167.
- 1991 *Quasiregular Mappings and Royden Algebras*, Ph.D. Thesis, Department of Mathematics, University of Michigan, 1991.

### Undergraduate Publications

- 1986 *A Survey of Recent Research and Applications of Number Theory to RSA Public-Key Cryptography*, Brigham Young University Honors Thesis, Mar. 3, 1986.
- 1985 *Aesthetics in Mathematics*, Insight: A Forum for Student Thought, Brigham Young University, Winter 1985, Vol. 2, No. 1, 9-11.

## Technical Reports and Papers for Industry Forums

- 2001 *An Approach to Robust Design Employing Computer Experiments*, with J. Lee, D. Li, X. Liu, A. Sudjianto, M. Vora, S. Wang, 27<sup>th</sup> Design Automation Conference at 2001 ASME Design Engineering Technical Conferences, September 9-12, 2001, Pittsburgh, PA
- 2000 *Object-Process Methodology as an Industry Enterprise Framework*, with D. Dori, R. Chow, B. Koo, C. Miyachi, T. Speller. 2000 Conference on Object-Oriented Programming, Systems, Languages & Applications, October 15-19, Minneapolis, MN.
- 2000 *Improving an Existing Design Based on Axiomatic Design Principles*, with X. Liu. Proceedings of the First International Conference on Axiomatic Design, Cambridge, MA, June 21-23, 2000, pp. 199-202.
- 1999 *An Approach for Computer Experiment Based Robust Design*, with D. Li, X. Liu, A. Sudjianto, M. Vora, S. Wang, Proceedings of the 5<sup>th</sup> International Society of Science and Applied Technology International Conference on Reliability & Quality in Design, August 11-13, 1999, Las Vegas, NV, pp. 98-107.
- 1999 *Applications and Challenges in Probabilistic and Robust Design Based on Computer Modeling*, Invited Talk, Proceedings of the American Statistical Association Section on Physical and Engineering Sciences, 1999 Spring Research Conference on Statistics in Industry and Technology, June 2, 1999, Minneapolis, MN, pp. 207-212.
- 1997 *Ford Technical Education Program: Reliability*, with J. King, P. Maurin, B. Rutter, and G. Stork, 3-day course material: participant and instructor guides, ©1997, Ford Motor Co., Dearborn MI.
- 1994 *A Failures per Thousand Reliability Verification Test Procedure for the Exponential Life Distribution*, with R. Haboush, Ford Motor Company Technical Report SR-94-86, Jun. 27, 1994.
- 1994 *The Geometric Moving Average in OBD-II Misfire Detection*, with J. James and K. Marko, presentation to On-Board Diagnostic Technologies section, SAE International Congress and Exposition, Detroit, MI, Feb. 28, 1994.
- 1993 *Process Control and Diagnostics: An Evolutionary Approach Based on Adaptive Learning*, with K. Marko and B. Bryant, U.S.-Korea Vibration Engineering Conference, Korea Advanced Institute of Technology, Mar. 1993.
- 1992 *Neural Network Application to Comprehensive Engine Diagnostics*, with K. Marko and B. Bryant, Proc. IEEE International Conf. on Systems, Man, and Cybernetics, Chicago, IL, Oct. 1992, 1016-1022.
- 1992 *A Comparison of Two Averaging Filters for Use in Misfire Detection*, with J. James, Ford Motor Company Technical Report SR-92-107, Aug. 10, 1992.
- 1991 *Engine Cold Test Data Analysis Using Trainable Pattern Classifiers*, with K. Marko, B. Bryant, L. Feldkamp, G. Puskorius, Conference Proceedings, AutoFact '91, Society of Manufacturing Engineers, Dearborn, MI, 18:15-18:22.