

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

# UMI

A Bell & Howell Information Company  
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA  
313/761-4700 800/521-0600



ENABLING ENTERPRISE INTEGRATION THROUGH ARCHITECTURE

by

WILLIAM DALE BURG, B.S.B.A., B.S.

A DISSERTATION

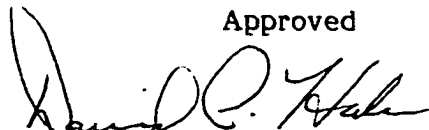
IN

BUSINESS ADMINISTRATION

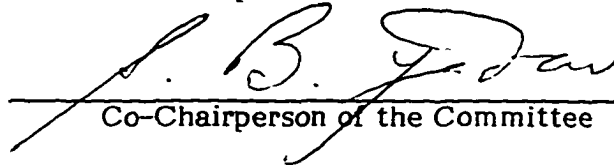
Submitted to the Graduate Faculty  
of Texas Tech University in  
Partial Fulfillment of  
the Requirements for  
the Degree of

DOCTOR OF PHILOSOPHY

Approved



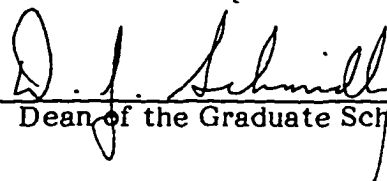
Co-Chairperson of the Committee



Co-Chairperson of the Committee



Accepted



Dean of the Graduate School

December, 1997

UMI Number: 9812006

Copyright 1997 by  
Burg, William Dale

All rights reserved.

---

UMI Microform 9812006  
Copyright 1998, by UMI Company. All rights reserved.

This microform edition is protected against unauthorized  
copying under Title 17, United States Code.

---

**UMI**  
300 North Zeeb Road  
Ann Arbor, MI 48103

© 1997

**William D. Burg**

## ACKNOWLEDGMENTS

I would like to express my sincere appreciation to a number of individuals and organizations who have helped to make this dissertation possible. First of all, I would like to thank the members of my dissertation committee: Dr. D. P. Hale, Dr. S. B. Yadav, Dr. J. G. Hunt, and Dr. J. B. Burns, for their continuous support, encouragement, and suggestions. Specifically, I would like to thank Dr. Hale for his unwavering support. Without it, this dissertation would not be a reality.

I am thankful to Texas Instruments, SEMATECH, and Electronic Data Systems for their generous financial support of this research effort and the ANSI's X3H7 Technical Committee for agreeing to review my efforts. Lastly, I would like to thank Glenn Hollowell his belief that the results of this research effort will be of value to the software practitioner.

## TABLE OF CONTENTS

ACKNOWLEDGMENTS . . . . .	ii
ABSTRACT . . . . .	vi
LIST OF FIGURES. . . . .	viii
CHAPTER	
I: OVERVIEW AND PURPOSE OF RESEARCH . . . . .	1
Introduction . . . . .	1
Problem Statement . . . . .	6
Purpose of Research . . . . .	9
Research Methodology . . . . .	10
Dissertation Structure . . . . .	12
II: METHODOLOGY . . . . .	13
Research Approach . . . . .	13
Research Methodology . . . . .	16
Grounded Theory . . . . .	16
Data Collection. . . . .	17
Analysis or Interpretive Procedures . . . . .	21
Form of Research Findings. . . . .	26
Validation of Research Findings . . . . .	28
Research Agenda . . . . .	30

III:	THE FIRST ROUND . . . . .	36
	Information Systems Architectures . . . . .	36
	Productive Strategies . . . . .	40
	The Mass Production and Invention Strategy . . . . .	44
	The Mass Customization and Continuous Improvement Strategy . . . . .	51
	The Software Development Process . . . . .	57
	Summary . . . . .	62
	Validation of Proposition . . . . .	63
IV:	THE SECOND ROUND . . . . .	64
	Identification of Data Collection Sites . . . . .	64
	Analysis of Interviews . . . . .	68
	Synthesis of Propositions . . . . .	83
	Validation of Propositions . . . . .	94
V:	THE THIRD ROUND . . . . .	96
	Identification of Data Collection Sites . . . . .	96
	Analysis of Interviews . . . . .	98
	Synthesis of Propositions . . . . .	101
	Validation of Propositions . . . . .	108
VI:	THE FOURTH ROUND . . . . .	109
	Identification of Data Collection Sites . . . . .	109
	Analysis of Interviews . . . . .	110
	Synthesis of Propositions . . . . .	111



Validation of Propositions . . . . .	112
VII: SUMMARY . . . . .	115
Integrating the Propositions . . . . .	115
Implications of the Model . . . . .	120
Validation of the Model . . . . .	121
VIII: LIMITATIONS, CONTRIBUTIONS, AND FUTURE RESEARCH . . . . .	123
Limitations of Research . . . . .	123
Contributions of Research . . . . .	124
Future Research Directions . . . . .	125
REFERENCES . . . . .	126
APPENDIX: TRANSCRIPTS OF INTERVIEWS . . . . .	134

## ABSTRACT

For firms to effectively compete in today's turbulent market environment, their supporting software systems must be able to provide new, effective system solutions in a timeframe necessary to enable the business change required to remain competitive. In order to provide flexible responsive information systems, many organizations are pursuing the idea of building software using factory-like concepts. To develop a software factory, information systems professionals focus on building standardized production processes, components, and tools that could be reused across new system solutions. To date, these attempts have resulted in the ability to build domain specific applications, but these applications are limited in their capability to be extensible. Thus, the requirement for systems to rapidly adapt has not been met. One of the major reasons for these limited results has been the failure to design the software factory concept upon an appropriate paradigm. Using the mass customization paradigm, this research effort represents a conceptual step towards building new system solutions based upon these driving business needs by identifying the functional requirements for its use as a referent architectural paradigm for an adaptive software factory.

Using grounded theory, this exploratory research effort attempts to identify the functional requirements of the command, control, and communication mechanism of a mass customization based software factory by evaluating current research and development projects centered around the ideas of the software factory and component reuse. By grounding this research effort in the context in which the solution must

apply, the formal propositions developed thorough this research effort will have a high degree of external consistency.

## LIST OF FIGURES

1.1:	Five Critical Issues Facing Information System Organizations in Turbulent Market Environments . . . . .	2
1.2:	Point Solution Approach to Five Critical Issues . . . . .	7
1.3:	General Systems Theory Approach to Five Critical Issues . . . . .	8
2.1:	Model of Research Output . . . . .	25
2.2:	Research Methodology for Determining the Functional Requirements for a Referent Architecture . . . . .	35
3.1:	Product-Process Change Matrix . . . . .	42
4.1:	Initial List of Organizations Involved in Architectural Efforts . . . . .	65
4.2:	Overall Relationship Between Propositions Developed in Round 2 . . . . .	93
5.1:	Overall Relationship Between Propositions Developed in Round 2 and 3 . . . . .	107
6.1:	Overall Relationship Between Propositions Developed in Round 2, 3, and 4 . . . . .	114
7.1:	A System and Its Environment . . . . .	116
7.2:	Integration of Views . . . . .	118

# CHAPTER I

## OVERVIEW AND PURPOSE OF RESEARCH

### Introduction

So dramatic, persistent, and pervasive are the changes in customer demands, technologies, geographical boundaries, and business processes that the question of *how to effectively compete in today's rapidly changing environment* is an issue of great strategic significance to many firms (Porter, 1985; Miles and Snow, 1986). To successfully compete in a turbulent market environment, firms must simultaneously solve five critical issues (see Figure 1.1): increase the flexibility of their business processes, decrease the cycle time to deliver new products, increase the quality of their products, increase their productivity, and still maintain low costs (Piore and Sable, 1984; Davis, 1987; Dertouzos, Lester, Solow, and The MIT Commission for Industrial Productivity, 1989; Quinn and Paquette, 1990; Boynton and Victor, 1991; Pine 1993). Unfortunately, the underlying structure of most large firms is just too "slow and maladaptive" to provide the flexibility, speed, quality and productivity required to successfully compete in "today's high-speed business world" (Toffler, 1990).

Through efforts to increase flexibility and speed, firms are discovering that inflexible information systems represent a major impediment to change (Hammer, 1990). By automating business processes through rigidly structured information

systems, firms have frozen "their organizations into patterns of behavior and operations that resolutely resist change" (Allen and Boynton, p. 435, 1991). The issue facing information technology organizations today is the fact that they must provide new system solutions in a time frame necessary to enable the business change required for a competitive posture (Stalk, Jr., 1988; Haeckel and Nolan, 1993).

- Flexibility of delivered solutions must be increased to deal with continuous change
- Cycle time to deliver solutions must be reduced
- Quality of delivered solutions must be increased so that solutions work correctly the first time
- Productivity of the software development process must be increased
- Cost to deliver and operate solutions must be dramatically reduced

**Figure 1.1 Five Critical Issues Facing Information System Organizations in Turbulent Market Environments**

Under the traditional system development approach of top-down functional decomposition, software engineers build software solutions by creating components that are a part of the overall application structure. Within the application structure, they specify not only the components but also the interrelationships among components. Because systems built using top-down functional decomposition result in tightly

coupled components under a specific application structure, these components prove too rigid to allow for effective integration under a new application configuration (McGregor and Sykes, 1991; Taylor, 1995). To reconfigure system solutions built using a top-down functional decomposition approach requires firms to rebuild major portions of existing systems. The issue in this rebuilding process is not whether new system solutions can be effectively constructed using a top-down functional decomposition approach. Rather, the issue facing information technology organizations today is the fact that software cannot be produced in a time frame necessary to enable the necessary business change required for a competitive posture (Texas Instruments, 1994).

The requirement for the quick delivery of new systems solutions is most commonly stated as a need for a reduction in cycle time to provision system solutions in response to changing business needs. A requirement that is based upon a number of experiences where top down functional decomposition was used to reconfigure systems such as the following (Texas Instruments, 1994):

**Twelve months cycle time to generate order fulfillment reports.**

During a firm's order fulfillment reengineering process, cycle time became a key business issue. Even though order entry and unit shipment events were automated and captured in databases, twelve months were required to generate order fulfillment cycle time reports. Once these reports were available, the business started to make significant improvements because problem areas could be identified and addressed.

**Excessive time to consolidate two merged divisions.**

When two separate business divisions were merged, a requirement was immediately generated for a common system to support the merged organizations. Within a six month period, the new organizations had sorted through and resolved the organization, facilities, personnel and other issues associated with the merger; however, the common system necessary to enable this merger took an additional eighteen months to deliver. Consequently, the benefits of the merger were delayed for over two years.

**Inability to support business information needs created by an organizational structure change.**

A fundamental organizational change from a functional to a matrix structure required shared "ownership" of resources, better management of human resources, and better communications between project leaders and center of excellence managers. The information technology support for this fundamental change took over two years to realize, resulting in significant delays to the organization.

The success of manufacturing firms in increasing quality and speed through the development of a flexible manufacturing approach has led firms to believe that a similar approach is possible in the production of software (Griss, 1993; Haeckel and Nolan, 1993; Texas Instruments, 1994). By analyzing the flexible manufacturing factory, firms realized that the key to establishing a flexible, adaptable production approach is to focus on achieving economies of scope, obtained by sharing standard parts and processes across a variety of related products (Cusumano, 1991). In developing a



flexible software factory, firms are attempting to obtain economies of scope by producing reusable standardized components that serve as the basic building blocks across different programs (Cusumano, 1989). To avoid the problems of a rigid centralized application structure, firms must attempt to establish a standardized software development process that can assemble software solutions from these plug compatible components in any configuration desired (Swanson, McComb, Smith, and McCubbrey, 1991).

To date, the software factory's ability to provide customized enterprise-wide solutions through reusable components can really be classified as practically nonexistent. Although component reuse rates of 60% have been achieved within a specific domain, usage rates across domains have reached only 20-30% (SRI International, 1993). In today's software factories, the reuse domain is so specific and the reuse level so low that these factories simply represent another development technique that, like many others, is helpful in some contexts and inappropriate in many others. Given the goal of the software factory, the problem with domain specific artifact reuse is that it fails to provide the leverage necessary to achieve the requisite flexibility, quality, and speed required to successfully compete in a turbulent market environment (Griss, 1993; Prieto-Diaz, 1993).

To overcome the limitations of the current software factory, a new comprehensive software development approach that reuses higher-level abstractions across domains needs to be identified and implemented (Barnes and Bolinger, 1991; Prieto-Diaz, 1993). To ensure the interdomain operability of these abstractions, the

new software development approach must provide an application structure that simultaneously avoids the interdomain impedance of bottom-up *ad hoc* approaches and the rigidity of centralized top-down approaches. To be effective, the new application structure must simultaneously possess the flexibility of a decentralized application structure and the efficiency of a centralized application structure (Allen and Boynton, 1991). For firms that cannot avoid a turbulent market environment, restructuring their software development approaches to provide quick delivery of new software solutions within a flexible application structure represents the only way to survive (Texas Instruments, 1994). Because current software development approaches cannot provide the required functionality, a new paradigm for the development of software needs to be established.

### Problem Statement

For firms facing a turbulent market environment, all five of the critical issues facing these firms must be resolved simultaneously in order for these firm to effectively compete in that environment. Point-based solutions that do not simultaneously address all of these critical issues will not be sufficient (see Figure 1.2). The issue for information systems organizations is to specify an information technology development process that enables organizations to effectively compete in a turbulent market environment by providing the organization a tool for simultaneously resolving all five of the critical issues facing these firms.

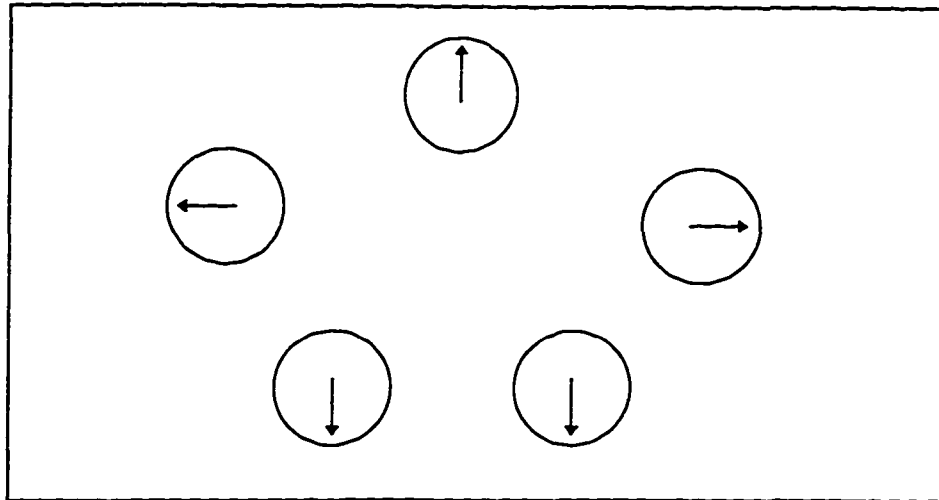


Figure 1.2 Point Solution Approach to the Five Critical Issues

In order to ensure that the emergent properties of a new information technology development process simultaneously resolve all five of the critical issues, general systems theory requires that the relationships among the properties of the set of components in the new solution process be adequately specified (Klir, 1985). Because completely different emergent properties can result for the same component properties combined in distinct ways, the overall structure of the relationships among components determines the emergent properties. In general systems theory, the overall structure of the relationships among the combined components is known as an architecture (Klir, 1985). Thus, as a precursor to specifying a new referent paradigm for the production of software, a referent architecture of the new information technology development process must be specified (see Figure 1.3).

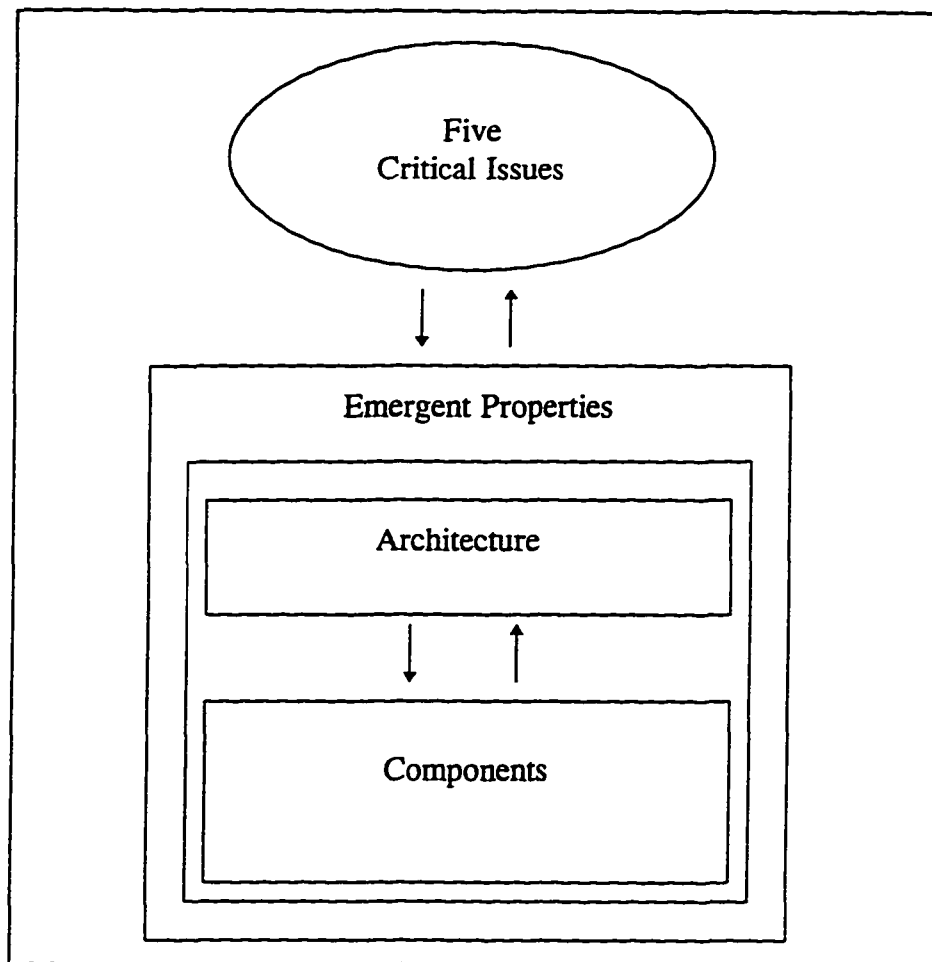


Figure 1.3 General Systems Theory Approach to the Five Critical Issues

Given a referent architecture, one can explicitly evaluate the tradeoffs among the properties of components that a specific design creates in order to determine the effect of these tradeoffs on the emergent properties of a system solution. By capturing the relationships among the properties of the components in the new information technology development process, a referent architecture provides an important conceptual link for simultaneously achieving (1) an increase in the flexibility of system

solutions, (2) a decrease in the cycle time to deliver new or to modify existing system solutions, (3) an increase in the quality of delivered system solutions, (4) an increase in the productivity of the software development process in delivering system solutions, and (5) a decrease in the cost to deliver new or modified system solutions. Without an effective referent architecture, software factories' attempts to build flexible software solutions will produce solutions that are too rigid, too expensive, and will take too long to change (Allen and Boynton, 1991; Texas Instruments, 1994).

### Purpose of Research

Lacking any theory on the appropriate structure of a referent architecture for the rapid provisioning of information system solutions, an exploratory research effort will be required to determine what architectural form a referent architecture must take in order to provide the requisite speed, flexibility, quality, and cost required in today's dynamic business environment. In exploring what form the referent architecture must take, one must remember the first principle of architectural development is that form or structure should follow function (Klir, 1985). To determine the structure or form of a referent architecture, one must first identify its functional requirements. As a starting point towards the development of a new paradigm for the information technology development process, the purpose of this exploratory research effort is to define a set of propositions that specify the functional requirements of a referent architecture for the rapid provisioning of system solutions in a changing business environment.

## Research Methodology

In order to define a set of propositions that specify the functional requirements of a referent architecture for the rapid provisioning of information system solutions, a scientifically-based research methodology capable of inductively generating propositions is required. Within the category of inductive research approaches, ethnography (Van Mannen, 1983), analytic induction (Miles and Huberman, 1984; Yin 1989), and grounded theory (Strauss and Corbin, 1990; Strauss, 1987; Glaser and Strauss, 1967) represent research methodologies that are based upon the canons of the scientific method and have been successfully used to generate propositions (Eisenhardt, 1989). Although all three of these research methodologies are aligned along the dimensions of purpose, data type, and data analysis method with the requirements of this research study, only grounded theory is also aligned along the dimension of unit of analysis (Strauss and Corbin, 1990).

Grounded theory is an iterative approach that uses constant comparison to identify and validate the theoretical relevance of emerging concepts. By constantly comparing the emerging set of relevant conceptual issues with the observations, prior results reported in the literature, and existing theory, grounded theory not only suggests the modification or addition of new conceptual issues but also helps ensure the accuracy of the current set or propositions by triangulation across multiple sources (Glaser and Strauss, 1967). Throughout the constant comparative process, theoretical saturation will determine if the incremental knowledge generation is minimal because the researcher is observing phenomena previously identified (Glaser and Strauss,

1967). If theoretical saturation has not been reached, theoretical sampling of the modified set of relevant conceptual issues will point the researcher toward the next set of data collection points.

When theoretical saturation has been reached, the constant comparative process ensures that the resultant set of propositions is highly likely to be empirically valid. Because the constant comparative process is so intimately tied to the evidence, the likelihood that the process will produce a valid set of propositions is extremely high (Eisenhardt, 1989). Although strong internal validity for the resulting set of propositions is virtually guaranteed, the external validity of the resulting set of propositions is dependent on the diversity of the data collection sites generated through theoretical sampling.

Even with strong internal and external validity, the ultimate test of a set of propositions is its ability to generate useful explanations, predictions, and a framework for action for practitioners (Glaser and Strauss, 1967). Unless the resulting set of propositions is comprehensible and makes sense to those practicing in the area, the potential new insights generated from the resulting set of propositions will be useless (Strauss and Corbin, 1990). To ensure that the resulting set of propositions is useful, Eisenhardt has proposed that theory-building research be evaluated on its ability to produce a set of propositions that is parsimonious, testable, and logically coherent (Eisenhardt, 1989).

Following the canons of grounded theory, an iterative, five-step research agenda for the execution of this research study was specified. In addition to the five

steps, an initial step for bootstrapping the process was also enumerated. The actual implementation of these five steps for each iteration of this research study will be presented in subsequent chapters as this research effort unfolds. The steps in this research agenda appear below.

Step 0: Review previous research to form initial implications

Step 1: Identify data collection sites using theoretical sampling

Step 2: Gather data

Step 3: Develop preliminary notions of propositions using open coding

Step 4: Synthesize preliminary notions into propositions using constant comparison

Step 5: Validate propositions and check for theoretical saturation using constant comparison

### Dissertation Structure

The dissertation is presented in the following manner: Chapter II details the research methodology used in the study. Chapter III presents the initial literature review and provides the development of the preliminary research proposition. Chapters IV, V, and VI present the results of the three complete rounds of the research agenda by providing a detailed presentation of the actual instantiation of each step in each round. Chapter VII summarizes the complete set of propositions developed in Chapters III, IV, V, and VI. In Chapter VIII, the limitations, contributions, and future research directions of this research effort are discussed.



## CHAPTER II

### METHODOLOGY

#### Research Approach

Building new knowledge is a spiral process that involves both inductive generalization and deductive verification (Wallace, 1971). Although deductive verification can be rigorously established through the use of testing of data through hypothesis, the logically flawed process of induction cannot provide a similar guarantee for the result of knowledge generation activities. Lacking any logically consistent or provably correct knowledge generation process, inductive researchers have often relied upon conjecture, intuition, and divine insight as their sources for new theory and knowledge (Eisenhardt, 1989). Although conjecture, intuition, and divine insight have all been cited as sources for the discovery of new knowledge, the serendipitous nature of these sources precludes their use as a research methodology in a rigorously designed scientific study (Eisenhardt, 1989). Given the exploratory purpose of this research effort, the research methodology selected for this study requires the use of a scientifically based exploratory research methodology capable of inductively generating propositions.

In addition to considering the specified purpose of a research study in the selection of a research methodology, the researcher must also consider the type of data (e.g., quantitative or qualitative) that will be gathered and analyzed in a research effort (Yin, 1989). Within the social sciences, qualitative data analysis methodologies are

generally considered to interpret empirical data through means other than statistical procedures or other methods of quantification (Strauss and Corbin, 1990; Strauss, 1987; Miles and Huberman, 1984).

Although no fundamental clash exists between the use of quantitative or qualitative data in inductive research (Glaser and Strauss, 1967), inductive research methodologies favor the use of qualitative data and qualitative data analysis methods over quantitative data and quantitative data analysis methods for the generation of new theory for three reasons. First, qualitative data provides “rich descriptions and explanations of processes occurring in local contexts” (Miles and Huberman, 1984, p. 15) that are generally not capable of representation in a quantitative form. Second, qualitative data analysis benefits from preserving the richness of the data, enabling the researcher to develop meaningful, complex insights that “are more likely to lead to ... new theoretical integrations” because qualitative data analysis “help(s) researchers go beyond initial preconceptions and frameworks” (Miles and Huberman, 1984, p. 15) as compared to solely converting the qualitative data into quantitative data through preselected counting and measuring operations (Strauss, 1987). Third, qualitative research methodologies produce findings whose validity can be judged based upon the relevance, plausibility, and credibility of these findings to subject matter experts (Strauss and Corbin, 1990; Glaser and Strauss, 1967).

Given the purpose of this exploratory research effort, a scientifically based research methodology capable of inductively generating propositions from qualitative data is required. Within the category of qualitative research approaches, ethnography

(Van Mannen, 1983), analytic induction (Miles and Huberman, 1984; Yin 1989), and grounded theory (Strauss and Corbin, 1990; Strauss, 1987; Glaser and Strauss, 1967) represent research methodologies that are based upon the canons of the scientific method and have been successfully used to generate propositions from qualitative data using qualitative data analysis methods (Eisenhardt, 1989). Although all three of these research methodologies are aligned along the dimensions of purpose (i.e., generating propositions), data type (i.e., qualitative), and data analysis method (i.e., qualitative), these three research methodologies differ significantly along the dimension of unit of analysis (Markus, 1992). In ethnography and analytic induction, the unit of analysis are reasonably well-defined entities, such as individuals or social units (e.g., work groups or organizations) (Eisenhardt, 1989; Yin, 1989); whereas, in grounded theory, the unit of analysis is the concept (Strauss and Corbin, 1990).

As a first step toward building a referent architecture for the rapid provisioning of information system solutions, this research effort is focused on discovering the functional requirements of that architecture. Because this referent architecture exists independently of any organization or individual, the unit of analysis of this research effort is a concept. Given the alignment between this research study and the grounded theory research methodology along the dimensions of purpose, data type, data analysis method, and unit of analysis, the grounded theory research methodology was selected as the appropriate one for this effort.

## Research Methodology

Qualitative research methodologies consist of four major components: data, analytic or interpretive procedures, the form of the research findings, and criteria for validation of the research findings (Strauss and Corbin, 1990). In order to understand and properly follow a grounded theory based research methodology, one must first understand how grounded theory instantiates these components and the interrelationships that this instantiation creates among them. In this section, an overview of grounded theory is presented that includes a comparison of grounded theory to analytic induction. The specific research agenda used in this research study is then presented.

### Grounded Theory

Faced with the need to develop new knowledge or theories within their discipline, Glaser and Strauss (1967) began an investigation to determine if a scientifically based methodology could be devised for building new knowledge. Starting with the question *Where do discoveries of new knowledge come from?* Glaser and Strauss (1967) determined that early efforts to determine the sources of new knowledge were often misdirected by researchers reporting where they were when a new insight or idea came to them. "After all, new ideas come from everywhere-- whether stepping off a street car or somewhere else" (Strauss and Corbin, 1990, p. 28). While Strauss and Corbin (1990) point out that this statement is true, they also point out that new ideas do not come to everyone about everything. Instead, they insightfully

point out that new ideas generally come to individuals who are deeply immersed in their area of study wherever they may be physically.

Through immersion, the researcher must become not only steeped in the data but also conversant in the models and meta-models within the area of study. When the data, or some feature of the data, provides a puzzling or unexplained interpretation, meaning, or pattern of events, the researcher must possess sufficient theoretical knowledge and understanding to realize that the current models or meta-models of the area of study are inadequate (Glaser, 1978). To develop theoretical sensitivity, a researcher must gain theoretical knowledge and understanding by posing questions such as the following: (1) What does the current model do?, (2) How is it conceived?, or (3) What are its general propositions? Through posing and answering these questions, a researcher might conclude that the current theoretical underpinnings of an area are inadequate. Lacking a provably correct knowledge generating process, the researcher is then faced with the question of where to begin in developing new or novel models. For Glaser, Strauss, and a number of other qualitative researchers (Glaser and Strauss, 1967; Miles and Huberman, 1984; Yin, 1989), the answer to the question *Where to begin?* lies with the collection and interpretation of data.

### Data Collection

Empirical research exists because of our desire to build theories and models that can describe, explain, or predict real world phenomena. By collecting data, or

observations, on the phenomenon of interest, researchers gather evidence that can then be used in either theory building or theory testing procedures to help ensure that theory mirrors the real world. In order to collect the data, the researcher must first identify a set of data collection sites in which the phenomenon exists.

In statistics, the collection of individual units whose characteristics are to be studied is known as the target population (Iman and Conover, 1989). Given the target population, the researcher can then attempt to identify all of the individual units within that population. Because of limitations in resources, identifying all of the individuals within a target population usually is not possible (Iman and Conover, 1989). In such cases, a population representative of the target population whose individual members are known must be found. Because of the potential for the introduction of a bias, the selection of the representative population, known as the sampled population, must include a thorough analysis of the reasons that individual units would be represented in the sampled population but not the target population.

In collecting data from a sampled population, limitations in resources often preclude conducting a census (Iman and Conover, 1989). In such cases, sampling is used to obtain a sample from the sampled population. To avoid the introduction of a bias, or nonsampling error, in the sample, statistically based methods rely upon the use of simple random sampling (Iman and Conover, 1989). The goal of random sampling is to ensure that every possible sample, with the same number of observations, is just as likely to be selected. In addition to providing a method for selecting samples, statistically based procedures use methods for determining the number of samples to be

selected to ensure diversity. By ensuring the diversity of the sample, simple random sampling enables valid projections to be made to the entire population from which the sample was obtained (Iman and Conover, 1989).

In inductive research, limitations in resources can also preclude conducting a census of the target population (Glaser and Strauss, 1967; Miles and Huberman, 1984). In such cases, the selection of a sampled population and the use of sampling must be employed. To avoid the introduction of a bias into a sample, inductively oriented research methodologies must also rely upon the use of scientifically based sampling procedures (Glaser and Strauss, 1967; Miles and Huberman, 1984; Eisenhardt, 1989; Yin, 1989). In grounded theory, the selection of the target population, the sampled population, and samples are all controlled by the process of theoretical sampling (Glaser and Strauss, 1967). Because theoretical sampling is a data collection process unique to grounded theory (Glaser and Strauss, 1967; Miles and Huberman, 1984), understanding theoretical sampling requires an understanding of how theoretical sampling is different from other data collection processes such as the data collection process used in analytic induction.

As the data collection process of grounded theory, theoretical sampling is driven by the relevance of the emerging concepts (Strauss and Corbin, 1990). In order to recognize the potential relevance of an emerging concept, the grounded theorists must first possess sufficient knowledge and insight, or theoretical sensitivity, of the area of study (Strauss and Corbin, 1990; Glaser and Strauss, 1967). To gain an initial immersion into the area under study, grounded theory relies upon a review of the prior

research as reported in the literature for the generation of a research framework or initial proposition (Glaser and Strauss, 1967). Based upon the research framework or initial proposition, theoretical sampling is used to determine the selection of the initial target population, sampled population, and sample. As a starting point in its data collection processes, analytic induction also utilizes a review of the prior research as reported in the literature for the generation of a research framework or initial proposition and the selection of the target population, sampled population, and sample (Miles and Huberman, 1984; Delbecq, Van de Ven, and Gustafson, 1975).

In the actual collection of the data, the data collection techniques used in grounded theory and analytic induction do not differ significantly. Grounded theory and analytic induction both use personal interviews as their primary source for data collection (Glaser and Strauss, 1967; Miles and Huberman, 1984). In addition to personal interviews, grounded theory also uses literature reviews as a supplemental source to provide insight or new ways of approaching the data (Strauss and Corbin, 1990).

In addition to providing a method for the selection of a sample, a data collection process must also provide a method for determining the sample size. In statistically based procedures, sample size is determined by the desired power and the effect size (Iman and Conover, 1989). In inductively based procedures, an iterative approach that cycles between data collection, analysis, and verification is executed until the data collection process's stopping rule is validated. The major difference between these two data collection processes is that, with each successive iteration, grounded theory



expands until theoretical saturation of the relevant concepts is achieved, whereas, analytical induction spirals towards an ever converging stable set of concepts based upon consensus (Glaser and Strauss, 1967; Miles and Huberman, 1984; Delbecq, Van de Ven, and Gustafson, 1975).

Guided by the relevance of emerging concepts, theoretical sampling requires a re-assessment of the selected target population, sampled population, and sample with each iteration of a grounded theory based study. If the assessment determines that the target population, sampled population, or sample are not sufficient, theoretical sampling requires the re-selection of these units. Although analytic induction also allows new concepts to emerge with each iteration, analytic induction predetermines and fixes the target population, sampled population, and the sample for a given study (Miles and Huberman, 1984; Delbecq et al., 1975).

#### Analysis or Interpretive Procedures

The second component of a qualitative research methodology, analytic or interpretive procedures, represents the set of techniques for conceptualizing data--breaking the data down for the purpose of discovering and verifying theoretical concepts, their properties, and the relationships among concepts (Strauss and Corbin, 1990). To interpret qualitative data, qualitative research methodologies tend to rely upon a similar set of techniques for data conceptualization. These techniques include coding, which can take the form of either margin notes or memo writing, integrative diagramming, which can take the form of a causal network, and counting (Glaser and

Strauss, 1967; Miles and Huberman, 1984). Although different qualitative research methodologies share these same techniques, the actual application of these techniques varies significantly by methodology depending on the essential analytic strategy of the methodology (Markus, 1992).

In analytic induction, the essential analytic strategy is one of data reduction accomplished by the use of all available data coupled with the use of negative cases to distill the few theoretical propositions that hold in every case (Miles and Huberman, 1984; Yin, 1989). When the analytic inductionist finds cases that do not support the emerging theoretical propositions, the relationships must be “thrown out for insufficient evidence” (Eisenhardt, 1989, p. 542) or “the initial proposition must be revised and re-tested with another set of cases” (Yin, 1989, p. 53). To support their strategy, analytic inductionists are adamant that sampling and analysis must proceed from “within case” to “across case” in order to preserve the “web of causality” in each sampled case (Miles and Huberman, 1984; Eisenhardt, 1989).

In analyzing individual cases, analytic inductionists use open coding and causal networks to capture the concepts and the causal relationships between concepts that exist within a site. In open coding, “the data are broken down into discrete parts, closely examined, and compared for similarities and differences ... . Open coding ... fractures the data and allows one to identify concepts, their properties, and dimensional locations” (Strauss and Corbin, 1990, p. 62). In addition to open coding, analytic inductionists develop causal models by linking the identified concepts into a “web of causality.”

After coding all within-case concepts, analytic inductionists proceed to cross case analysis. In cross case analysis, the goal of analytic induction is to distill the causal relationships that hold in all of the studied sites (Eisenhardt, 1989). In addition, verification of the relevance of a specific concept is established by counting the incidence of the occurrences of the concept across cases (Miles and Huberman, 1984). To the analytic inductionists, data or observations are valued because they represent the evidence that constitutes the proof of the relevance of the concept.

In grounded theory, the essential analytic strategy is one of synthesis accomplished by the selective use of available data coupled with the use of negative cases to generate insights into potential theoretical propositions (Glaser and Strauss, 1967). In fact, grounded theorists actively seek out negative cases as a way to generate insights that could expand, enlarge, or enrich the emerging theoretical propositions. Naturally, the insights generated from such a tenuous source are not incorporated into the emerging theoretical propositions unless they are verified by means of further data collection and analysis. However, the value of the seeking out negative cases lies in suggesting to the researcher “the most opportune places, persons, or documents to go to for evidence of emerging concepts” (Strauss and Corbin, 1990, p. 181).

To support their strategy, grounded theorists move freely across the analytic inductionists unit of analysis in an attempt to capture all the relevant theoretical concepts, even though not all of them might be observed in every given setting (Eisenhardt, 1989). Where analytical inductionists use diversity within their predetermined and fixed sample to reduce the set of theoretical propositions, the

grounded theorist deliberately seeks out diversity to expand the emerging theoretical concepts wherever such diversity can be found. In analyzing new cases, grounded theorists use open coding to identify potential theoretical concepts and integrative diagramming to help clarify the relationships between emerging concepts (Strauss and Corbin, 1990).

To the grounded theorists, the relevance of a concept is based upon what it adds to the theoretical understanding of the emerging theory or model regardless of the number of times that the concept is observed (Strauss and Corbin, 1990). Because open coding can only suggest new concepts, grounded theorists rely on constant comparison of emerging new concepts with observations, prior results reported in the literature, and prior theory to verify the relevance of emerging propositions (Glaser and Strauss, 1967). Through the use of a systematic set of constant comparative procedures, the grounded theorists build theoretical propositions that are already provisionally tested or grounded in the phenomenon they represent by triangulation across multiple sources (Glaser and Strauss, 1967). By continuously comparing new concepts in the emerging theory with the study's observations, prior results reported in the literature, and prior theory, both the internal validity and the external validity of the emerging theory are ensured within the scope of the theoretical sampling (Eisenhardt, 1989).

With each iteration, the new theoretical propositions are sorted based upon the perceived relevance to the emerging theory. By identifying the relevance of emerging theoretical propositions through constant comparison, grounded theory allows the

grounded theorists to selectively gather new data to move beyond the data at hand while still remaining thoroughly grounded in them (Glaser and Strauss, 1967). To the grounded theorists, data or observations are valuable because observations suggest new concepts, not because the collective weight of the observations constitutes the proof of a concept.

Based upon the cannons of grounded theory, a model of the research output for this research study was developed and is shown in Figure 2.1. In this model, the

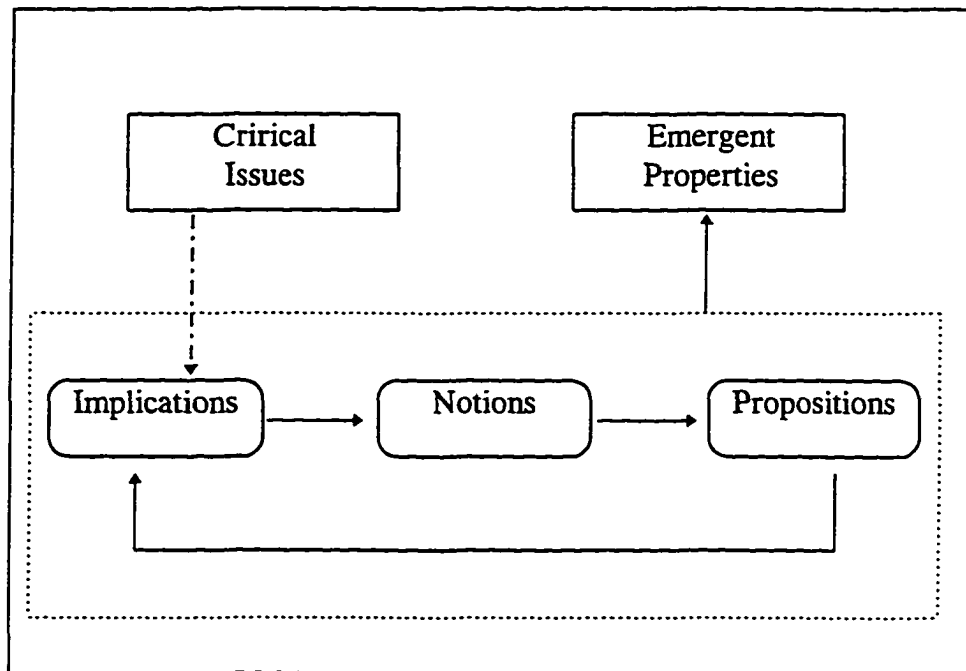


Figure 2.1 Model of Research Output

implications of the issues identified during each research round form the basic notions for emerging concepts. As these notions are explored through theoretical sampling, the

relevance of emerging propositions is established through constant comparisons of emerging new concepts with observations, prior results reported in the literature, and prior theory. Once the relevance of these new propositions has been established, the implications generated by these new propositions form the basic notions for the notions of the next research round.

### Form of the Research Findings

In addition to their agreement on the purpose of a research methodology (i.e., generating new knowledge), analytic inductionists and grounded theorists also agree that new knowledge should be described in terms of the relationships between concepts and that the products of their efforts can be, but need not be, written up in the form of formal propositions (Glaser and Strauss, 1967; Eisenhardt, 1989). Although analytic inductionists and grounded theorists agree on the form that their findings should take, they do not agree on the nature of the concepts presented in their respective findings. Based upon the discussion of the difference in these methodologies relative to their respective essential analytic strategy, the differences in the nature of the concepts presented in analytic induction and grounded theory could be classified as the difference between a descriptive approach versus a prescriptive approach respectively.

In analytic induction, Miles and Huberman (1984) discuss three kinds of conceptual categories: (1) descriptive codes that entail no interpretation, (2) interpretive codes that reflect an understanding of local dynamics, (3) and explanatory codes that illustrate an emergent pattern that the analyst has deciphered while unraveling the

meaning of local events and relationships. By contrast, Glaser (1978) distinguishes two type of codes: (1) substantive and (2) theoretical (1978). Substantive codes conceptualize the empirical substance of the area of research. Theoretical codes conceptualize how the substantive codes may relate to each other as hypotheses to be integrated into theory.

Although a direct comparison between these two sets of conceptual categories would be difficult if not impossible, Strauss makes it quite clear that there is no place in grounded theory for what Miles and Huberman call descriptive codes:

A word should be said here ... about the difficulties novices often have in generating genuine categories. The common tendency is simply to take a bit of the data and translate that into a précis of it. In effect they are, as are many researchers who use other methods of analysis, remaining totally or mostly on a descriptive level, not much different from the actors themselves. (Strauss, 1987, pp. 29-30)

In short, while the propositions generated by analytic induction and grounded theory are very similar in form, they differ significantly in content. In analytic induction, all three layers of codes remain relatively close to the data of the particular cases, in contrast to the substantive and theoretical codes of grounded theory, which rise above the particulars of individual cases and synthesize them. In this research study, the research findings will be reported in the form of formal propositions (see Figure 2.1).

## Validation of Research Findings

As Eisenhardt (1989) points out, there is no generally accepted set of guidelines for evaluating theory building research. Although grounded theorists and analytic inductionists concur that theory building research should be evaluated on the two broad criteria: (1) outcome, e.g., the product, and (2) process, e.g., the procedures performed by the researcher, grounded theorists and analytic inductionists differ significantly on the specific criteria that should be used because the content of the products and the procedures prescribed by each group differ (Markus, 1992).

For the analytic inductionists, the appropriate criteria for evaluation of research findings is based upon “rules of evidence” and “proof.” For instance, both Yin (1989) and Eisenhardt (1988) state that a researcher must display sufficient evidence in support of the theory, and Miles and Huberman believe that the researcher should provide ways for others to “audit” their research procedures:

One has to begin ... by describing one's procedures clearly enough so that others can reconstruct them and, further down, the line, corroborate them and do secondary analysis. ... [T]here has to be an “audit trail.” Without a trail, one cannot determine the dependability or the confirmability of the findings ... and verify the accuracy and legitimacy of the procedures used to establish the researcher's findings. (Miles and Huberman, 1984, p. 224)

For the grounded theorists, the appropriate criteria for evaluating of the research findings is based upon “plausibility” and “credibility.” Glaser (1987) states that “Unfortunately, grounded theory is still read by many as factual descriptions”, and Strauss states that:



The credibility of the theory should be won by its integration, relevance and workability, not by illustration used as if it were proof. ... [A]s grounded concepts they are not proven; they are only suggested. ... Proofs are not the point. It is not incumbent upon the analyst to provide the reader with description or information as to how each hypothesis was reached. Stating the method in the beginning or appendix is sufficient, perhaps with an example of how one went about grounding a code and an hypothesis. (Strauss, 1987, p.134)

Similarly, Corbin and Strauss do not require an “audit trail,” but only “reasonability good grounds” for accessing the quality of a researcher’s procedures:

However, even in a monograph ... there may be no way that readers can accurately judge how the researcher carried out the analysis. The readers are not present during the actual analytic sessions or their sequence. ... The detail [the researcher provides on the method] need not be great even in a monograph. But it should provide some reasonably good grounds for judging the adequacy of the research process. (Strauss and Corbin, 1990, pp. 16-17)

In addition, grounded theorists and analytic inductionists differ significantly on the criterion for selecting a group of “people in the know” to judge the product, e.g., the theory itself, based upon each group’s respective criteria for evaluating theory building research (Markus, 1992). For the analytic inductionists, the relevance of the theory should be judged based upon its primary value to the academic community (Miles and Huberman, 1984; Eisenhardt, 1988; Yin, 1989). In contrast, grounded theorists believe that relevance of the theory should be judged by knowledgeable participants who are the subjects of the research, rather than other researchers:

Why generate grounded theory? Why bother, when in each area of life there are people in the know? These people are so knowledgeable that they think they they can predict, explain and understand just about everything that happens in their terrain, field, area or world. They are the leaders and the consultants; ... What the man in the know does not want is to be told what he already knows. What he wants to be told ... [is] how to handle what he knows with some

increment in control and understanding of his area of action. ... [The grounded theorists provides this when their theory] fits, works and is relevant. The man in the know spots these criteria instantly when the theory he hears rings true and relevant. "That's the way it is," "That's right," are comments we often hear upon presenting a grounded, substantive theory to the knowledgeable. (Glaser, 1987, pp.12-13)

For the grounded theorists, verifying that the emerging theory is grounded in the world in which the problem exists represent validation. To validate that the propositions generated during each round of this research effort are grounded in the world in which the problem lives, a set of knowledgeable participants who where the subjects in the research round will be selected and asked to judge the validity of the set of generated propositions at the end of each round.

### Research Agenda

Based upon the cannons of grounded theory, an iterative, five-step research agenda for the execution of this study was specified (see Figure 2.2). In addition to these five steps, an initial step that allows for the bootstrapping of the whole process was also enumerated. A brief explanation of each step accompanies the presentation of this research agenda.

#### **Step 0: Review previous research to form initial implications**

In order to develop the initial implications for this exploratory investigation, a literature review was conducted to establish a working knowledge of the referent domains of information systems architectures, productive strategies for the production of goods and services, and the application of these two domains to the software

development process. Through this immersion in the literature, the theoretical sensitivity of the researcher became grounded in the area of study. By starting with a properly grounded initial proposition, a researcher is able to specify the type of organization to be approached and the type of data to be collected. Without a research focus, an exploratory research study can easily become overwhelmed by the data.

**Step 1: Identify data collection sites**

Theoretical sampling was used to identify the set of data collection sites. In theoretical sampling, data sites are identified for their potential ability to provide relevant insights into the emerging concepts. In this research study, those organizations investigating architecturally based solutions to the rapid provisioning of information systems solutions were identified as the initial target population. With each subsequent iteration, the evolving relevance of new concepts controlled the selection of the sample (Identified as 1 in Figure 2.2).

**Step 2: Gather data**

During each iteration, the current set of emerging relevant concepts served to drive the selection of the questions asked of participants through personal interviews. The questions and each participant's responses to them were recorded by the researcher in the form of an interview transcript. Once an interview transcript was determined to be correct and complete by the interview participant (i.e., verification), the interview was coded (Identified as 2 in Figure 2.2).

**Step 3: Analyze data and develop preliminary notions of propositions**

The process of coding the respondent's interviews into a set of preliminary notions was accomplished by open coding—breaking the data down into discrete parts and looking for similarities and differences within the observations (Strauss and Corbin, 1990). In this study, the open coding of the interview notes took the form of margin notes and interview write-ups. As a result of this coding process, a preliminary set of propositions was identified and formally presented for each iteration in this study (Identified as 3 in Figure 2.2).

**Step 4: Synthesize preliminary notions into propositions  
using constant comparison**

The synthesis of the preliminary notions into a set of propositions was accomplished by comparing the preliminary notions with prior results reported in the literature and existing theory. By juxtaposing the preliminary notions with prior results reported in the literature and existing theory, grounded theory can establish the theoretical relevance of these notions. When the preliminary notions were supported by the prior results reported in the literature and by existing theory, the researcher interpreted this support as an indication of the relevancy of the notions and an absence of researcher bias. When the preliminary notions were not supported by the prior results reported in the literature and in existing theory, the researcher exercised extreme care in validating the existence and relevancy of the notions with subject matter experts to ensure that the notions were not a product of researcher bias (Identified as 4 in Figure 2.2).

**Step 5: Validate propositions and check for theoretical saturation using constant comparison**

To validate that the set of propositions generated during each round were properly grounded and complete, a set of knowledgeable participants who were the subjects in the research round was selected and asked to judge the validity of the set of generated propositions. These results are reported at the end of each round. To verify that the propositions were accurately captured and theoretical relevant, the set of emerging propositions was verified by triangulation across the opinions of subject matter experts, the empirical observations collected during this study, the prior results reported in the literature, and the existing theory reported in the literature during the synthesis step during each round (Identified as 5a in Figure 2.2).

During the process of validating the generated propositions, the set of propositions were also tested for completeness (Identified as 5b in Figure 2.2). Comments such as “That’s true, but what about [this issue]” or “That’s true, but how are you going to deal with [this issue]” suggested that the generated propositions were accurate but incomplete in the eyes of the knowledgeable participants. When the set of generated propositions was found to be incomplete either by the set of knowledgeable participants or through constant comparison with the empirical observations collected during this study, the prior results reported in the literature, or the existing theory reported in the literature, the researcher determined that theoretical saturation had not been reached. In general, a set of propositions could be found to be incomplete because (1) the emergent properties of the whole set were deemed insufficient (i.e

necessary but not sufficient) for addressing the five critical issues (i.e., because of a lack of breath), or (2) because the set of propositions were deemed insufficient in their specification of the level of detail of the functional requirements (i.e., because of a lack of depth). In either case, the set of implications was modified and theoretical sampling used to point the researcher towards the next set of data collection points. When theoretical saturation had not been established, the research effort returned to Step 1 to begin a new iteration of the research agenda.

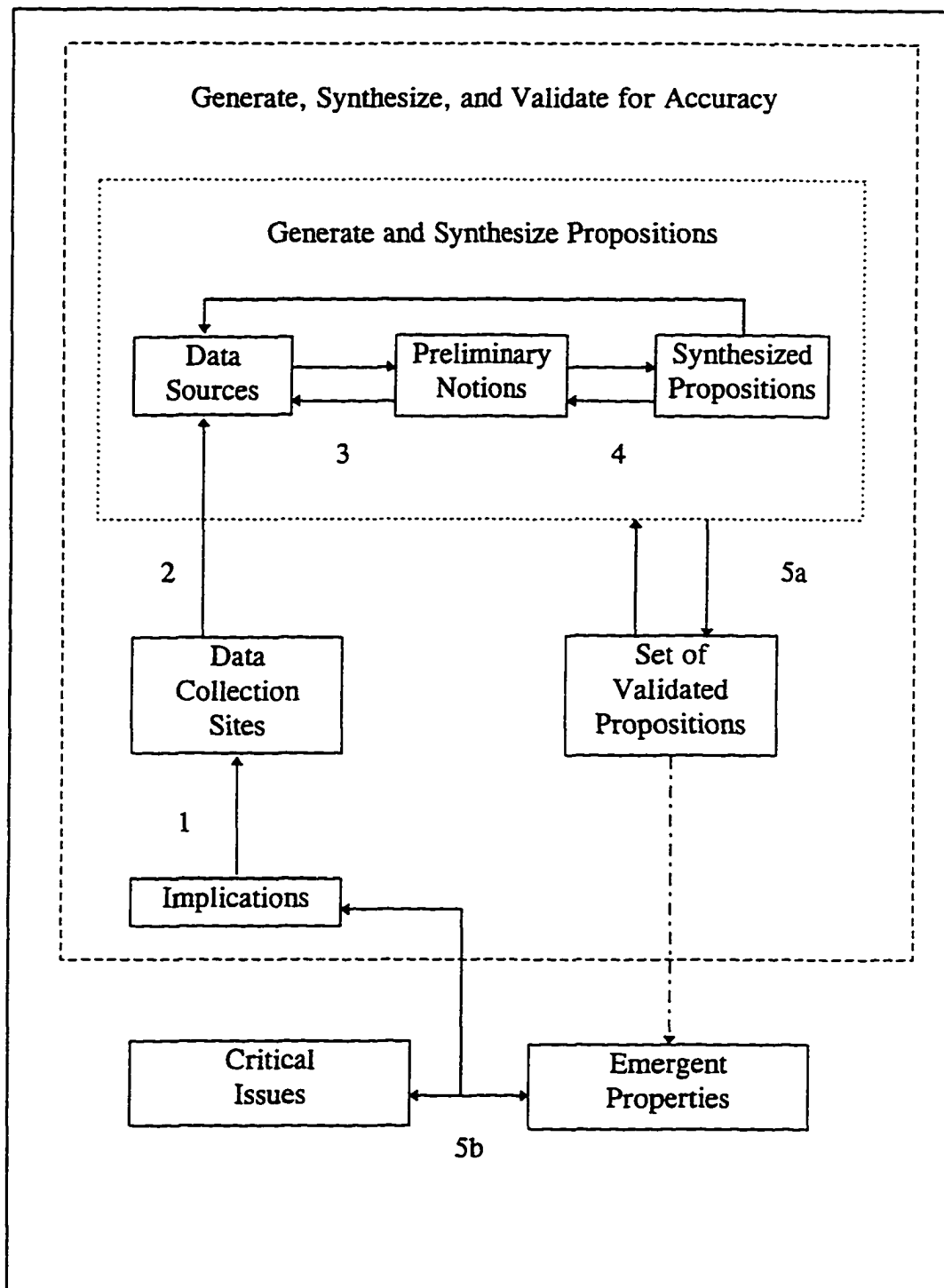


Figure 2.2 Research Methodology for Determining the Functional Requirements for a Referent Architecture

## CHAPTER III

### THE FIRST ROUND

Given the necessity of a general system theory approach, the first implication of this research effort is that a general systems theory solution requires the specification of an appropriate referent architecture. Working from this implication, the first round of this research study undertakes a literature review to establish a working knowledge of the referent domains of information systems architectures, productive strategies for the production of goods and services, and the application of these two referent domains to the software development process.

#### Information Systems Architectures

As computer-based information systems solutions have moved from single hardware-dependent programs to enterprise-wide suites of applications, the need to refine and understand the dimensions of these systems and the relationships between these dimensions has grown correspondingly. In order to properly manage the increased size and complexity of software based systems, organizations find that they must possess a blueprint (or architecture) that can guide them in defining the policies, rules, and requirements of their processes for analyzing, planning, designing, and implementing or acquiring system components from the perspective of the overall resultant system or enterprise perspective (Stecher, 1993).



Because the architecture "specifies how and why the pieces fit together as they do, where they go, when they're needed, and why and how changes will be implemented" (Allen and Boynton, 1991, p. 435), developing an information system without the perspective provided by architecture will result in chaos (Zachman, 1987). To achieve a general systems synergy, the architecture must be based upon a unifying "logical construct for defining and controlling the . . . integration of all the components of the system" (Zachman, 1987, p. 276). Without an architecture, developing "the set of policies and rules that govern an organization's (process for determining the) actual and planned arrangements of computers, data, human resources, communication facilities, software, and management responsibility" (Allen and Boynton, 1991, p. 435) becomes impossible.

The realization of the importance of architecture in modern computer-based information systems emerged to describe the relationship between components at a hardware level. When IBM introduced the System/360 in 1964, the fact that the same hardware components and the same relationships between these components was implemented from the smallest to largest processor in the System/360 family line was a remarkable feature (Stecher, 1993). By specifying the same architecture throughout the System/360 line, IBM guaranteed that hardware dependent programs were portable across all processors in the System/360 line.

Similarly, as advances allowed for compiled languages, hardware independent programs were developed with a file-oriented approach that tightly bound global data to the defining application. By separating hardware and software into two independent

dimensions, information systems professionals were now able to identify not only the hardware's architecture, but also a single dimension software architecture. By using top-down functional decomposition, information systems professionals developed program modules that focused on defining the temporal sequence of the operations performed by the computer and represented these steps with flowcharts. To represent the software components and relationships between components within a specific programmed application, information systems professionals developed data flow diagrams and structure charts (Yourdon and Constantine, 1975).

Although a single dimension software architecture proved adequate for a set of completely independent applications, the tight binding between the data and the functionally decomposed program required under a single dimension software architecture resulted in severe problems as multiple interdependent applications that took slightly different views or perspectives of the organization arose. Problems with redundancy, inconsistency, and inflexibility led to the development of data base management technology aimed at overcoming these limitations by separating the program and the data into two independent dimensions (Codd, 1971). The new dimensions, representing the data components and the relationships between them, became known as the data architecture or information architecture. In order to build software applications based upon the new two-dimensional architecture for the software component of an information system, information systems professionals employed a number of data centric software development methodologies such as Business Systems

Planning developed by IBM (1984) and Information Engineering developed by James Martin (1990).

Coupled with the ability to separate the data from the program came the ability to place the data and program components on completely independent hardware platforms. Linking the independent program and data components, each with its own architecture, across platforms led to the recognition that the communication relationships between these two dimensions could also be described using an architecture--specifically a communications architecture or more appropriately meta-architecture. An early example of such an architecture, designed to permit the communication between on-line peripherals, is IBM's Systems Network Architecture (Stecher, 1993).

As the importance of computer networks grew, the ability to distribute and to link diverse software applications across heterogeneous hardware architectures became essential. To overcome the implementation specific constraints imposed by the selected implementation technology, information system professionals defined a technology architecture that specified the technology and its interrelationships. IBM's Systems Application Architecture, introduced in 1987 and specifically designed to support the porting of applications across diverse hardware implementations, is an example of a technology architecture (Wheeler and Ganek, 1988).

Because the emergence of each new technology for building computer-based information systems gives rise to a corresponding need to specify an architecture for the new technological dimension and refine the meta-architectures encompassing it, the

historically driving force in the development of information systems architectures has been technology (Stecher, 1993). Although technology can provide wonderful capabilities that can enable business solutions, the mere acquisition of technology alone does not ensure these solutions. Only when technology is combined under an appropriately defined enterprise oriented architecture do the desired emergent properties of the overall system appear (Texas Instruments, 1994).

Today's problem is not that organizations lack information systems architectures (cf. Zachman, 1987; Sowa and Zachman, 1992; CIM-OSA, 1993), but that the information system architectures that exist are too rigid and maladaptive to support the development of new, effective system solutions in a time frame necessary to enable the business change required for a competitive posture (Allen and Boynton, 1991). To understand why the structure of current information systems architectures is too rigid and maladaptive, one must examine not only the process under which they are constructed, but more importantly the various productive strategies under which the current process itself is defined.

### Productive Strategies

Although change can be viewed in a number of different ways, one of the most useful ways to view change in order to understand its effects on organizational strategy and structure is through the product-process change matrix (Boynton and Victor, 1991; Pine, 1993; Boynton, Victor, and Pine, 1993). The idea behind the product-process change matrix is that changes in market structure can be decomposed along the two

independent dimensions of product change and process change. As defined by Boynton and Victor (1991), product change includes changes required by demands for new products or services based upon competitor moves, shifting customer preferences, and the firm's entry into new markets. Process change is defined as changes in the "organizational capabilities resulting from the people, systems, technologies, and procedures used to develop, produce, market, and deliver products or services" (Boynton, Victor, and Pine, 1993, p. 42).

Within each of these two dimensions, change itself is further classified as either dynamic or stable. Boynton and Victor (1991) classify change as dynamic if the change is rapid, revolutionary, and generally unpredictable, as opposed to stable change which is slow, evolutionary, and generally predictable. With the ability to classify product change and process change as either stable or dynamic, Boynton and Victor (1991) are able to develop a two-by-two cell product-process change matrix. As refined by the work of Pine (1993), Boynton, Victor, and Pine's (1993) product-process change matrix is shown in Figure 3.1. A review of the product-process change matrix in Figure 3.1 reveals that product change and process change can be viewed as independent of one another and that in combination with the type of change can be viewed as falling into one of four possible categories or quadrants.

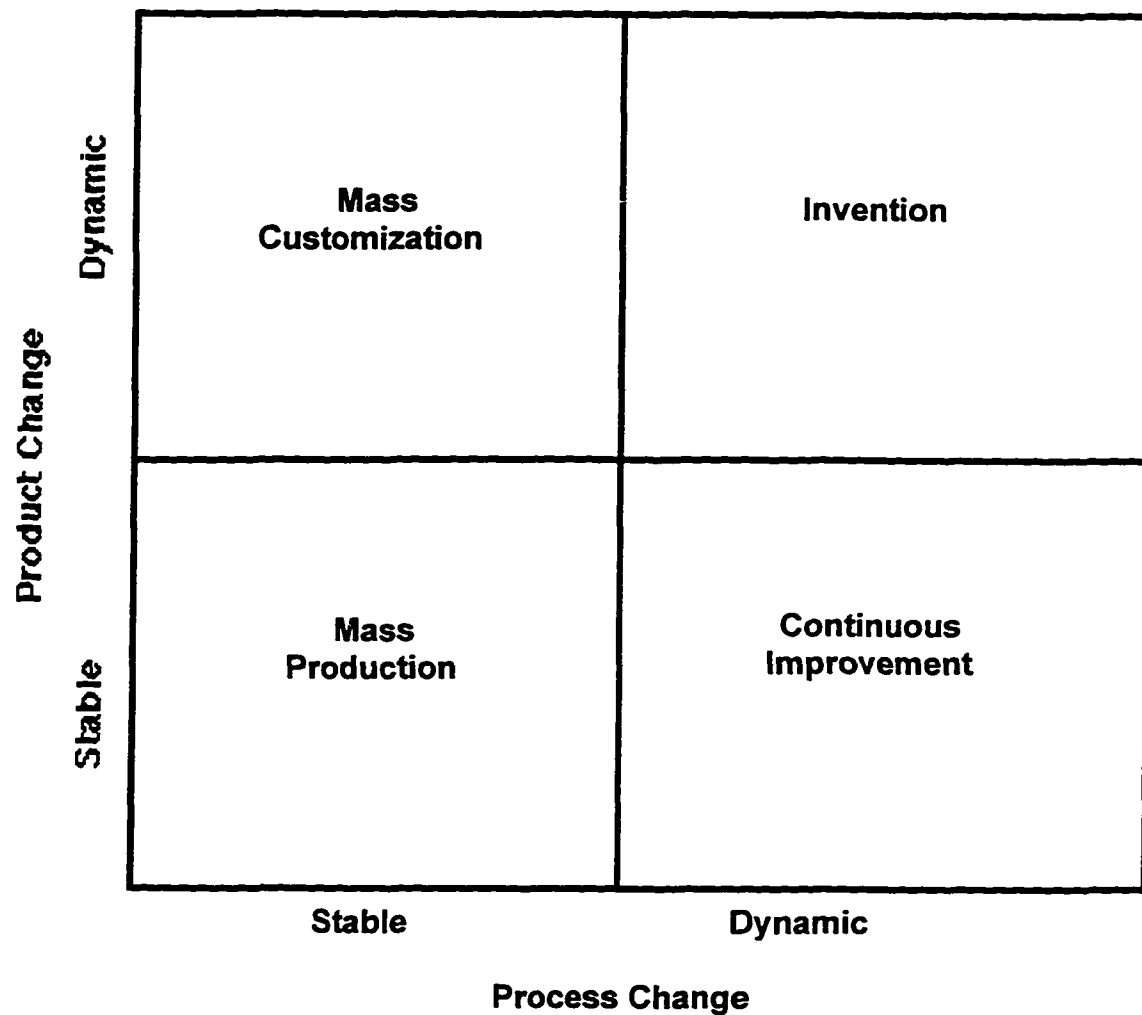


Figure 3.1 Product-Process Change Matrix

Before reviewing the competitive strategies defined by the product-process change matrix, an important point helpful in understanding the product-process matrix needs to be made. The point is that, in reality, the type of change experienced by any one specific firm lies along a continuum from very stable change to very dynamic

change rather than at the representative average determined over a number of firms. In other words, all firms do not discretely cluster in the center of the four quadrants of the product-process change matrix. Instead, each of the four quadrants of the product-process change matrix represents an idealized description or exemplar of a specific organizational structure and strategy. The value of these idealized descriptions is that they provide firms with the ability to assess their own competitive positions relative to market structure, to develop a vision of the structure and strategy their firms must implement to successfully compete in the future, and to develop transitional plans for migrating to that new vision (Boynton, Victor, and Pine, 1993). By providing these capabilities, the product-process change matrix gives managers the ability to proactively plan for change.

Pine (1993) has identified two major competitive strategies based upon the product-process change matrix. The first major strategy is what has been historically known as the mass production and invention strategy and is based upon the linkage of the invention quadrant with the mass production quadrant. The second major strategy, based upon the linkage of the continuous improvement and mass customization quadrants, represents the new competitive strategy of mass customization and continuous improvement. In order to understand either major strategy, one must first understand both of the individual quadrants involved in the strategy and then the synergistic linkage that exists between the two quadrants.

## The Mass Production and Invention Strategy

Historically, the craft, or job shop, approach to production represents the most common form of the economic production of goods and services (Thompson, 1967; Blau and Schoenherr, 1971). The craft production strategy, characterized by its ability to produce unique or novel products on a job-by-job basis, is predicated upon the idea that tools or machines and processes can be used to augment the craftperson's skills, thereby allowing the craftperson to produce a wide variety of unique goods or services. The more flexible the tool and the more widely applicable the process, the more they contribute to the craftperson's capacity for productive expression (Piore and Sable, 1984). At the extreme, the possibilities of the craft based approach are ultimately limited only by the imagination and skill of the craftperson for a given type of material. Because of the ability to produce unique or one-of-a-kind goods, the purest form of craft based production is also known as invention.

For the craft based organization, building production or process expertise aimed at any one specific product represents a strategy counterproductive to its long run viability. Instead, successful craft based organizations hire highly skilled, creative craftspeople and provide them with sophisticated general-purpose tools. Lacking standardized procedures, craft organizations also grant employees a high degree of autonomy over the production process, while simultaneously charging them with the responsibility for each project's success. Success, therefore, requires not only the cooperation of craft employees but also their collaboration on a project by project basis (Burns and Stalker, 1961).



The ability of the craft-based approach to produce a wide and constantly changing assortment of goods and services for unstable markets ensures its viability even today. When the creation of specialized goods in small quantities is required, the highly skilled, general purpose labor and machines of the craft based organization provide the best option. By allowing craftpersons the flexibility to use their skills and general-purpose tools to explore and translate ideas into novel products and processes, the craft-based approach is uniquely designed to compete under the conditions of dynamic change.

The trade-off for this flexibility is, of course, cost. Because of the long learning curves of the craftpersons (through apprentice programs) and the labor intensive nature of craft production, goods and services produced under the craft based approach are produced at low levels of productivity resulting in high costs (Piore and Sable, 1984). As long as the ability to obtain the novel product or service is worth more than the corresponding cost of its production, the craft based approach adds value to the overall economy.

In its purest form, the craft based organization provides the foundation for the invention strategy. To compete under the invention strategy, organizations must be designed to take advantage of conditions involving both dynamic product and process change. The pure craft based organization with its highly skilled, creative employees, sophisticated general-purpose tools, highly decentralized, autonomous decision making process, and flexible workflows provides the organizational structure necessary to pursue the development of new ideas and to support the investments in changing

process capabilities required under dynamic product and process change (Piore and Sable, 1984). Based upon these capabilities, Boynton, Victor, and Pine (1993) select the invention based organization as the exemplar for the dynamic product and dynamic process change quadrant in their product-process change matrix.

The mass production strategy, or Fordism as it is also known, grew out of the need to produce a large volume of a product at a very low cost. Within a given technology, manufacturing firms, as exemplified by Henry Ford's automobile manufacturing company, realized that the only way to achieve large volumes at a low unit cost is to create manufacturing processes that produce standardized products with great operational efficiency. These firms also realized that the key to achieving the operational efficiency required by the mass production strategy is to maximize the through-put of the manufacturing facility, through-put being defined as the number of units produced over a given period of time (Blau and Schoenherr, 1971). By focusing on operational efficiency, firms using the mass production strategy are able to take advantage of economies of scale.

Firms learned that the best way to maximize through-put is to standardize not only the product produced but everything else as well. Through standardization, the handwork tasks of the craft based approach can be decomposed into simple steps that can be performed faster and more accurately by specially built machines dedicated to a specific task than by human hand (Piore and Sable, 1984). The more specialized the machine, the faster it works and the less specialized its operator needs to be. The trade-off for this reliance on specialized dedicated machines is a loss of flexibility.

Long, uninterrupted production runs on the specialized dedicated machinery become the key to achieving the through-put required for mass production. Shutting down a line to make slight adjustments for quality control reasons or to change over to production of a different product results in a decrease in through-put and is therefore strongly discouraged.

In its purest form, the operating goal of the mass production firm becomes the production of a standardized product using standardized interchangeable parts assembled within a standardized process flow controlled through a standardized or centralized structure that uses standardized labor operating standardized (i.e., dedicated) machines that are part of a standardized line in a standardized factory. Standardization is so important to the mass producer that the methodology for standardizing processes by finding and eliminating inefficiencies in tasks through meticulous time and motion studies, known as "scientific management," is synonymous with this form of production (Taylor, 1911). The mass production firm, with its emphasis on achieving operating efficiency through ubiquitous standardization, creates a production based approach capable of producing a large volume of a low unit cost general good or service (Piore and Sable, 1984).

By decomposing and standardizing each and every task in the production process, mass production firms are able to extend the degree of specialization of labor to new levels. Instead of employees being specialized based upon their craft knowledge of production processes, employees are specialized based upon their ability to execute a single repetitive task that is predefined by a single central authority to ensure its unity

within the overall standardized production process. In making the transition to task focused specialization, mass production firms are able to explicitly separate the roles of thinking and doing and redefine the role of the mass production employee as subordinate to the standardized product through the repetitive tasks they perform (Piore and Sable, 1984). By lowering the learning curve/skill set required for doing and relegating the role of doing to labor, mass production firms are able to transform labor into a interchangeable, standardized part itself. The ability to treat labor as an interchangeable, standardized part allows mass production firms to employ the increased number of workers needed to staff their larger production facilities with little or no training.

Because of the operational efficiency achieved under mass production, the total productivity, as measured by through-put, of a group of workers in a mass production facility not only exceeds the total productivity of an equal number of workers in a craft based facility, but also increases at a higher rate the larger the size of the facility. As long as completely stable demand for a large volume of a standard product or service prevails in a market, mass production presents an effective means for meeting that demand at a low unit cost (Chandler, 1962). Under market conditions favorable to mass production, the increased productivity of mass production labor results not only in lower prices for mass produced products and services, but also in a corresponding increase in the wages paid to mass production labor. Within the general economy, the mass production firm's proficiency in simultaneously decreasing the price of consumer

goods and services while increasing the level of employment and wages in the general labor market results in an increased standard of living.

In its purest form, the mass production organization works best for a completely stable product produced by a completely stable process. Because changes in either product or process make specialized dedicated machinery obsolete, causing costly retooling of facilities, change of any kind works against the mass production approach. When competitive advantage is based upon delivering a large volume of a low cost standard good or service, the mass production strategy, with its centralized, hierarchical control system, is the production strategy of choice. Based upon the mass production strategy's ability to excel in the production of goods or services under the conditions of stable product change and stable process change, Boynton, Victor, and Pine (1993) selected the mass production organization as the exemplar for the stable product and stable process change quadrant in their product-process change matrix.

Together, the invention quadrant and the mass production quadrant form the mass production and invention strategy identified by Pine (1993). After reviewing the characteristics of the individual invention and mass production quadrants, one might first conclude that there is no synergistic link between these two quadrants because they seem to be the antithesis of one another. The invention quadrant, designed to deal with dynamic product and dynamic process change, appears to be at the opposite end on the spectrum when compared with the mass production quadrant, designed to deal with stable product and stable process change. Furthermore, the craft based approach uses a decentralized organizational structure that combines thinking and doing through

collaboration, while the mass production based approach uses a centralized organizational structure that separates thinking and doing though competition (Piore and Sable, 1984). The larger the number of dimensions examined, the more evident the contrary nature of these two approaches becomes. Yet, the reason for the synergistic linkage between these two approaches can be paradoxically found in one of these contradictions.

Although the mass production organization is designed to function under conditions of limited change, the mass production organization must occasionally completely retool for new products made using completely new processes as shifting markets, intensifying competition, and advancing technologies make the old products and processes obsolete (Chandler, 1962). Because the mass production organization uses specialized machines to produce general goods, it is not capable of developing and producing the new specialized machines and processes necessary for retooling. Instead, the mass production organization must rely on the invention organization to provide the new specialized machines and processes it requires (Piore and Sable, 1984). Because markets are not completely stable structures immune to change, the symbiotic relationship between the invention quadrant and the mass production quadrant exists to provide the mass production quadrant with one way to deal with dynamic change through a phased approach.

## The Mass Customization and Continuous Improvement Strategy

The continuous improvement approach for the production of a stable good developed as a response to the mass producers needs to increase quality and deal with dynamic process change in a more incremental manner (Womack, Jones, and Roos, 1990). The continuous improvement organization, like the mass production organization, operates in a product market where large volumes of a stable goods compete on price. To remain competitive, both organizations must constantly find new ways to reduce their products' cost. Although both organizations compete in the same market, the way they compete is significantly different, a difference that is based upon each organizations' respective approach to determining product cost.

For a firm competing in a market that demands large volumes of a stable product or service, unit cost represents only one dimension in the purchase decision. As anyone who has ever owned a car knows, the purchase price or unit cost of the car really only represents part of the total cost of owning the car. The cost of maintenance, taxes, fuel, and insurance must also be considered when purchasing the car. Because the total cost of owning the car is ultimately that which concerns a buyer, the quality of the car and the technology contained in it also become significant dimensions. The realization that the total cost, rather than the unit cost, of a product is ultimately important is key to understanding the difference between the continuous improvement and the mass production organizations.

Mass production works because manual tasks can be decomposed into simple, repeatable steps—steps that can be automated by building specialized dedicated machinery that can be operated by unskilled operators over long production runs. Without the ability to decompose and automate the manual production tasks, mass production will not work. By decomposing a product's production process into a fixed sequence of automated steps, mass production transforms these individual steps into the defining operational characteristic. To determine the cost of a product, a mass producer simply totals the costs of "doing" the individual automated steps involved in the production of the product. The accuracy of the summation presupposes that all of the costs of production are captured in the "doing" steps—a supposition that is not correct.

Generated by the need to have extra working capital for work in process inventory sitting idle between "doing" steps, carrying costs represents one cost not captured by summing the costs of "doing." Other costs not captured are the cost of machine downtime for quality control adjustments and setup, the cost of reworking or scraping a defective part, and the cost of waiting for a machine to be repaired (Drucker, 1995). The failure of the summed cost of the individual steps to accurately capture all the costs of production results from the inability of the decomposition approach to account for all of the costs (Johnson and Kaplan, 1987). No one in the mass production organization thought of designing a "doing" step for idly holding inventory between production steps or for producing a defective part. Yet, both of



these activities exist within the total production process. To properly determine the cost of production, the total cost of operating the production process must be captured.

The continuous improvement organization recognizes that determining and controlling the total cost of the production process is important. To determine all of the costs, the continuous improvement organization focuses on finding not only the costs of "doing" but also the costs of the cracks that lie between the "doing" steps. By integrating both the steps and the cracks into a single unified view, the continuous improvement organization builds a global process view that cuts across the individual steps defined by functional decomposition. The key to building a global view of the total production process is to organize employees into cross-functional collaborative teams of functional specialists (Nonaka, 1988; Adler, 1991). These teams that can now focus not only on the individual functional steps but also on the linkages between them. By managing both the steps and the linkages between them, the cross-functional teams of the continuous improvement organization are able to make process based adjustments that achieve efficiency, quality, and ongoing product improvements simultaneously (Womack, Jones, and Roos, 1990).

The continuous improvement organization's focus on managing the total cost of production through a cross-functional process approach allows the continuous improvement organization to incrementally address dynamic process change rather than ignoring process changes until a total retooling is economically required. Because the continuous improvement organization can effectively integrate dynamic process change into an existing process, the continuous improvement organization can produce large

volumes of a stable product capable of not only competing on price, but also having superior quality and technology than a corresponding mass produced product. The ability of the continuous improvement organization to produce goods or services under the condition of stable product change and dynamic process change led Boynton, Victor, and Pine (1993) to select the continuous improvement organization as the exemplar for the stable product and dynamic process change quadrant in their product-process change matrix.

The mass customization approach for the production of goods and services was developed as a response to the firm's need to simultaneously increase flexibility, decrease cycle time, increase quality, increase productivity, and still maintain low costs. Based upon the success of the continuous improvement organization at increasing quality while maintaining low costs, firms realized that the tradeoffs between flexibility, cycle time, quality, productivity, and cost inherent in the mass production strategy could be overcome. By breaking the tightly bound relationship between product and process present in the mass production approach, the continuous improvement organization found that there were multiple ways to produce the same product with some process combinations that resulted in higher quality products than others. With product and process decoupled, firms also realized that a standardized set of processes could be used to produce a wide variety of products (Boynton and Victor, 1991).

The key to producing a wide variety of products with a stable set of processes is to organize the firm as a loosely coupled dynamic structure. In this structure, stable

business processes are placed at the nodes and linked by centrally coordinated, dynamically coupled arcs (Pine, 1993). By selecting world class business processes for the structure's nodes, the loosely coupled dynamic structure ensures high-levels of quality and productivity in the final product's components. By flexibly linking these processes in a dynamic manner, the loosely coupled dynamic structure provides unique products in a very short cycle time. By properly coordinating the linkage between the processes, the loosely coupled dynamic structure produces seamless products that achieve the quality and cost required of mass customization (Pine, 1993).

To achieve mass customization, firms must first decouple their processes from their products and turn their processes into stable modules. Armed with these stable modules, firms must then create an architecture for quickly linking the modules together in a coordinated manner to produce a product with the unique characteristics required by a customer (Pine, Victor, and Boynton, 1993). Without an effective architecture for configuring the links between the modules, achieving the flexibility required for mass customization is impossible. Therefore, the key to achieving mass customization is to define and build an appropriate architecture.

By taking the collaborative approach of the invention organization for the design of products and basing their production on stable processes derived from the continuous improvement organization, the mass customization approach is able to simultaneously achieve variety and efficiency (Boynton, Victor, and Pine, 1993). When competitive advantage is based upon quickly delivering a high quality, unique product or service at a low cost, mass customization is the production strategy of choice. Based upon the

mass customization organization's ability to produce a wide variety of goods or services using stable process, Boynton, Victor, and Pine (1993) selected the mass customization organization as the exemplar for the dynamic product and stable process change quadrant in their product-process change matrix.

Together, the continuous improvement quadrant and the mass customization quadrant form the new mass customization and continuous improvement strategy identified by Pine (1993). Because mass customization requires an adaptive base of world-class processes capabilities, mass customization organizations must rely on the continuous improvement quadrant for continuously refined and enhanced process capabilities (Boynton, Victor, and Pine, 1993). The symbiotic relationship that exists between the continuous improvement and the mass customization quadrants creates a dynamic stability based upon the adoption of the dynamic processes of the continuous improvement quadrant into the stable processes used in the production of mass customized products. The synergy created between the two quadrants requires that the old contradictions of the mass production and invention strategy must now be managed simultaneously. Under the new competitive strategy, firms no longer have to choose between either decentralization or centralization, cycle time or efficiency, and flexibility or cost. Instead, firms must now develop a vision that includes elements of both extremes simultaneously. To intelligently make these choices, firms will need the help of a well defined enterprise-oriented system development architecture.

## The Software Development Process

Organizational units developing information systems have long realized that the need to quickly deliver new system solutions requires finding an alternative to the craft based production of rigidly structured application solutions (Cusumano, 1989). Instead of building each and every new system solution from scratch, information systems professionals understand that they must find ways to capture and then leverage their previous efforts when building new solutions. Encouraged by manufacturing's success at developing standardized production processes, components, and tools that could be reused across new products, information systems professionals are attempting to emulate manufacturing's success by developing a software factory (Cusumano, 1991).

As Cusumano (1989) points out in his review of the historical development of the software factory, no one definition of the software factory exists. With multiple organizations developing software factories applied to a wide spectrum of different software application types produced under different productive strategies, the resulting assortment of software factories defies a single definition. However, the first public proposals for a software factory, introduced in 1968 by General Electric (GE) and by American Telephone and Telegraph (AT&T) (Cusumano, 1991), emphasized slightly distinct approaches to the application of factory-like concepts to the production of software that not only drives the development of software factories, but also remains useful in classifying and understanding the current state of the software factory.

While GE's approach focused on the standardization of tools and processes, AT&T's approach emphasized the production of a standardized set of parameterized

software components that would serve as the reusable building blocks across different applications (Cusumano, 1989). Although both of these approaches have enjoyed limited successes, neither approach by itself nor both approaches taken together have yet produced the interdomain operability required of today's software solutions. In order to understand these limitations of the software factory as it is currently designed and implemented, one must understand the present state of standardized software processes, tools, and components.

Although the goal of systematic software reuse is to build software components capable of being reused across multiple applications, current reuse technology produces components that are so domain specific that their reuse across multiple application domains is precluded. While component reuse rates of 60% have been achieved within a project's domain, usage rates across projects have reached only 20-30% (SRI International, 1993). In order to achieve interdomain reuse, new software components that are predicated upon higher-level interdomain abstractions must be identified and implemented (Barnes and Bolinger, 1991; Prieto-Diaz, 1993). In implementing these new generic components, a mechanism that allows application developers to extensibility instantiate these components to provide new unforeseen software capabilities must also be developed. Until new higher level interdomain abstractions are identified and implemented through a mechanism that provides extensible instantiation, the goal of systematic reuse will not be achieved (Gamma, Helm, Johnson, and Vlissides, 1995).

One example of the problem of extensible instantiation is AT&T's initial attempt at building reusable software from parameterized software components. The idea behind parameterized software components is that component designers can determine *a priori* what functionality to parameterize prior to system assembly. By allowing system assemblers flexibility in selecting various levels of functionality, parameterized software components provide systems assemblers with the ability to tailor (or specialize) the systems functionality within the limitations of the overall design. However, the problem with the parameterized software component's predetermined functionality is that it does not represent a mechanism for achieving the requisite extensibility of the systems overall functionality (Pawson, Bravard, and Cameron, 1995).

Paralleling the effort to build reusable software components is the effort to develop standardized software production processes and tools aimed at increasing the productivity, reliability, and quality of the software solutions through process standardization and control. By creating centralized management-control structures operating standardized processes, software factories can reduce the variability in the software development process. By eliminating variability, software factories can not only maximize productivity (as measured by through put) but also establish a consistent level of quality for each and every product produced in the factory. By providing automated support tools for these standardized processes, the cycle time for delivery of a specific product can also be reduced (Griss, 1993).

In seeking to emulate the success of manufacturing in providing automated support tools for the production of products through CIM (Computer Integrated Manufacturing), information technology organizations have responded by developing CASE (Computer Aided Software Engineering) tools. By automating a standardized process, CASE tools attempt to provide one standardized process for the production of all software types (Cusumano, 1989). The problem with the CASE approach is that there is no one standardized process capable of building all software types. Even under mass production, no one standardized process exists that can produce all goods. Instead, what results is a set of CASE tools, with each tool automating a specific standardized process capable of producing a specific type of software product.

To integrate multiple specific standardized processes into one production facility, the manufacturing community developed the concept of flexible manufacturing. By reducing the set-up time for switching between multiple standardized production processes, flexible manufacturing can produce a finite number of single-lot size prespecified products in a very short cycle time in any sequence desired. Although being able to quickly select and then execute specific mass production oriented processes from a finite group of these processes represents increased flexibility in terms of the production facility, flexible manufacturing does not provide the increased product flexibility required for extensible solutions (Cusumano, 1989).

In an attempt to provide the required product flexibility, a software factory that combines flexible manufacturing with standardized, interchangeable parameterized parts has been developed (Swanson, McComb, Smith, and McCubbrey, 1991).



Although reuse rates of over 90% are reported, along with an order of magnitude improvement in software development productivity, a close examination of the application domain addressed by this joint approach reveals a very narrow domain. Because parameterized software components and flexible manufacturing are both mechanisms that predefine the system's overall functionality, the reuse leverage obtained through this joint approach applies only to the predefined domain. The following quote illustrates the point.

For business applications, several writers point out that reusability may be most successful with transaction processing systems analogous to the two ASF systems described in the article. Such systems share sufficient commonality to make reuse practical, for example, a deposit transaction is similar in both a savings and money market application. (Swanson, McComb, Smith, and McCubbrey, 1991, p. 568)

By defining a standardized deposit transaction template for transaction processing systems, the manner in which a customer can interact with the software system becomes constrained within some overall domain. Within that overall constraint, specialization or extendibility of the system's behavior can be achieved without affecting the system's overall operation. However, extensibility beyond the pre-constrained overall domain cannot be achieved without redesigning the overall relationship between components. To compete in a dynamic market, business process reengineering has taught that underlying system assumptions (or constraints) must be surfaced and broken in order to allow us to build effective complete system solutions in a time frame necessary to enable the business change required for a competitive posture (Hammer, 1990). Since parameterized software solutions constrain a software system's

behavior to a pre-defined domain, parameterized software solutions are not capable of providing the requisite extensibility required in today's turbulent business environment.

### Synthesis of Proposition

Although firms have tried to organize the software development process under the invention, mass production, and continuous improvement strategies (Cusumano, 1989, 1991; Humphrey, 1989), the discussion of the four possible strategies for the production of goods and services indicates that the invention and the mass customization strategies are the only productive strategies capable of providing the flexible software solutions required for a firm to compete in a turbulent market environment. As the literature review reveals, the major difference between these productive strategies' emergent properties is based upon the overall structure (or architecture) that they impose upon the relationship between components--a finding completely consistent with general systems theory.

Therefore, to provide rapid provisioning of information systems solutions, the overall structure (or architecture) of the provisioning process must adhere to the basic architectural principles of the mass customization productive strategy. Stated as a formal proposition, this requirement becomes Proposition 0 listed below.

**Proposition 0:** The ability to rapidly provision new information systems solutions requires that the referent architectural form of a general systems theory solution be based upon the mass customization productive strategy.

Since mass customization is based upon an architecture designed as a centrally coordinated, loosely coupled dynamic structure, selecting mass customization as the referent architectural form implies that the functional requirements of this centrally coordinated, loosely coupled dynamic structure must be specified. In software architectures, the structure responsible for providing coordination between the system's components is known as the architecture's command, control, and communication mechanism (Shaw and Garland, 1996). Therefore, the selection of the mass customization as the referent architectural form implies that the functional requirements of the architecture's command, control, and communication mechanism be specified.

#### Validation of Proposition

To validate Proposition 0, B. Joseph Pine II and Andrew C. Boynton (i.e., the two individuals responsible for the identification of the mass customization paradigm for the production of good and services) were contacted and consented to review the base proposition. Based upon Pine's work in identification of the mass customization paradigm (Pine, 1993; Pine, Victor, and Boynton, 1993) and Boynton's work in identification of the dynamically stable organization (Boynton, and Victor, 1991; Boynton, Victor, and Pine, 1993), Pine and Boynton both agreed that mass customization is the only currently known productive paradigm for the rapid production of goods or services.

## CHAPTER IV

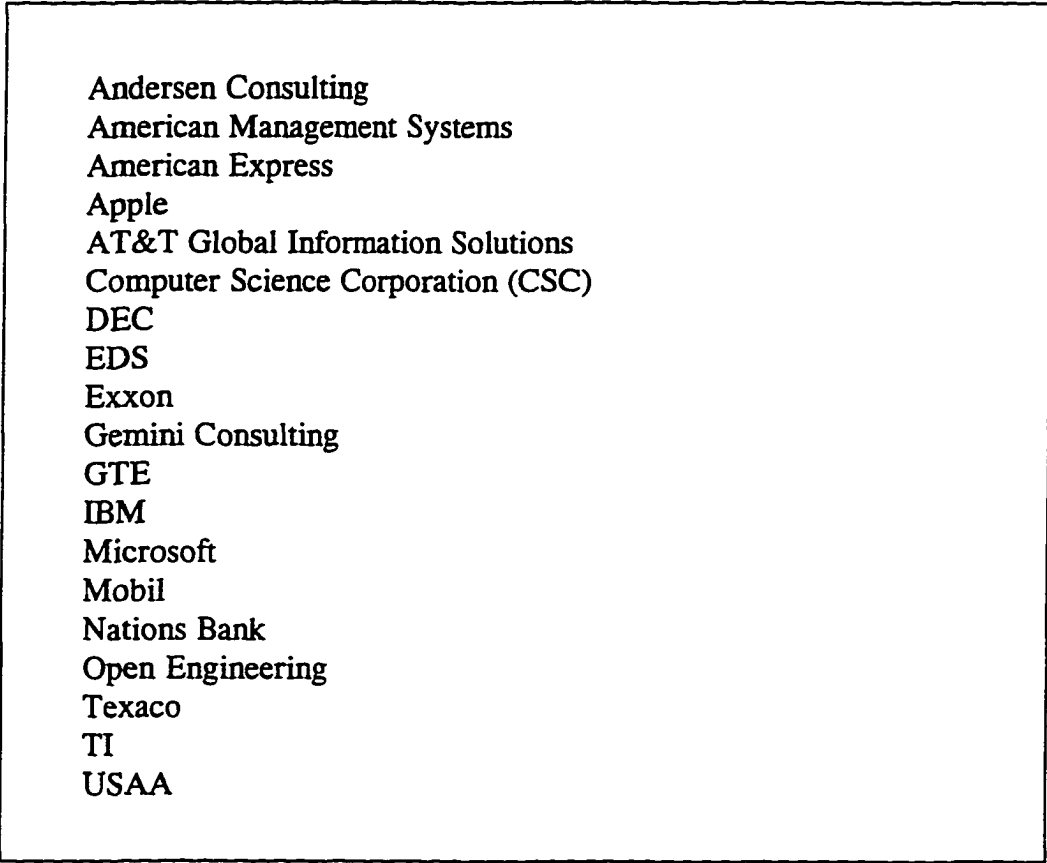
### THE SECOND ROUND

The implication of the base proposition developed in the first round suggests that the development of a general system theory solutions requires that the functional requirements of the architecture's command, control, and communication mechanism for the rapid provisioning of information systems must be specified. In order to discover these functional requirements, the second round of this research effort focuses on identifying and reviewing the prominent architectural efforts of information systems organizations dealing with dynamic change. To discover which information systems organizations were currently working on architecturally based solutions for rapidly provisioning new information systems solutions, the review of the academic literature conducted in the first round was consulted. Although the academic literature review indicated that "Developing an Information Architecture" represented one of the most important issues for organizations queried by the Society for Information Management (Niederman, Brancheau, and Wetherbe, 1991), it failed to identify specific information systems organizations that are currently investigating architectural based solutions.

#### Identification of Data Collection Sites

To identify information systems organizations investigating architectural based solutions, prominent industry based research and monitoring services were identified and asked to participate in this research effort by supplying member company names

and contacts that were currently investigating architecturally based solutions to rapid systems provisioning. After successfully recruiting the Gartner Group, the Patricia Seybold Group, and CSC Index, a list of information systems organizations currently involved in significant architecturally based efforts for rapid provisioning of information systems was developed. The initial group of information systems organization is shown below in Figure 4.1.



- Andersen Consulting
- American Management Systems
- American Express
- Apple
- AT&T Global Information Solutions
- Computer Science Corporation (CSC)
- DEC
- EDS
- Exxon
- Gemini Consulting
- GTE
- IBM
- Microsoft
- Mobil
- Nations Bank
- Open Engineering
- Texaco
- TI
- USAA

Figure 4.1 Initial List of Organizations Involved in Architectural Efforts

Working from this initial list, each of the listed organizations was contacted and asked to participate in this research effort. All of the organizations contacted consented

to some initial level of participation with the exception of IBM. As exposure to these organizations resulted in an increased knowledge of their architectural efforts, the scope of the initial list was modified to reduce overlap. Exxon's, Texaco's, and Mobil's IS architectural efforts were all found to center around the Petroleum Open Systems Corporation's (POSC) architecture resulting in the collapse of Exxon, Texaco, and Mobil into a POSC based architecture group represented by Mobil's POSC architectural staff. In the case of Apple's VITAL Technical Architecture, the overlap between Apple's VITAL and DEC's DSTAR architecture was found to be significant, resulting in the collapse of DSTAR into VITAL.

In some cases, the organizations' willingness to participate was found to be based upon their desire to develop a system integration process that would be built upon a sound architectural approach rather than any actual architecturally based effort within the organization. Although both GTE and AT&T were currently engaged in system integration projects, neither organization possessed a formal architecturally based effort. Because of the lack of any experience with architecturally based integration efforts, these organizations were removed from the target data collection group.

In other cases, the organizations declined to grant access to proprietary information because of the presence of other organizations within the research study that they felt were their competitors. In these cases, a single representative organization was selected to represent the industry in question. For the financial

services industry, USAA became the selected representative while Anderson Consulting was selected to represent the consulting industry.

After this consolidation process, the initial list of nineteen organizations was reduced to the nine organizations that formed the data collection sites for the second round of this research study. These nine organization are listed below in Table 4.1.

Table 4.1 Consolidated List of Organizations Participating in Research Study

Organization	Architecture
Apple	Vital Technical Architecture
Andersen Consulting	STAR Framework
CSC	PRISM
EDS	RightStep
Microsoft	Microsoft Solution Frameworks (MSF)
Mobil	Petroleum Open Systems Corporation (POSC)
Open Engineering	Object Oriented Business Engineering (OOBE)
Texas Instruments (TI)	Enterprise Integration Framework
USAA	USAA Target Technical Architecture

From each of these organizations, access both to copies of their proprietary architectures and their chief architect or architectural group was obtained. Because all of these organizations classify their respective architectures as proprietary, these organizations would not allow copies of their architectures to be directly included in this research study as source documents. However, specific questions regarding the

purpose, design, and implementation of each organization's architecture were addressed based upon a review of the organizations' architecture in personal interviews either with each organization's chief architect or with its architectural group. The researcher's notes from these interviews are included in the appendix of this study.

### Analysis of Interviews

In order to address changing information system requirements, each of the organizations interviewed is attempting to define an information systems architecture that will allow it to deal with change in some systematic manner. Even with a common objective, the underlying architectural approaches taken by these organizations seem to fall into two separate groups based upon their views of an information systems architecture.

The first group, represented by the POSC, VITAL, and MSF architectures (Interviews A.1.1, A.7.1, and A.8.1), views an information systems architecture from a predominantly technology-oriented perspective. For this group, the driving architectural issue is technology and the recent technological shift from a mainframe to a distributed computing environment, such as client/server. Although this group readily understands that an information systems architecture must deal with changing business requirements, each sees client/server technology as the driving force for enabling such change (Interview A.7.1). As client/server based architectures, these architectures are built upon the idea of dividing the information system into three dimensions: data, process, and presentation which is the three tier client-server



architecture. By separating these three dimensions, the architecture provides a mechanism for isolating changes in one dimension so that it will not cause changes in another dimension (Interviews A.7.1, A.8.1, and A.9.1).

Under these organization's implementation of the client/server architectural approach, changing business requirements are facilitated by providing an "empowered end-user" with the capability to quickly change processing requirements within the client (Interview A.7.1). Since each end-user can represent a unique client, the flexibility of a the client/server architectural approach to simultaneously support a heterogeneous set of clients represents the antithesis of the legacy mainframe information systems approach. By moving the binding of data, process, and presentation to the client from the mainframe, the client/server architecture can easily support multiple bindings of these three dimensions instead of the single binding found in legacy mainframe systems. As business requirements change over time, the "empowered end-user" can change its processing needs accordingly.

To support the binding of data, process, and presentation within the client, the architecture must provide the "empowered end-user" with a set of tools for quickly and seamlessly integrating these three dimensions on the client (Interview A.7.1). Without the development of the supporting technology, the "empowered end-users" will be anything but empowered. Since the realization of the client/server approach depends heavily on the development of even more enabling technology, Apple and Microsoft view information systems architectures as having an extremely technologically driven

aspect to them and are relying on technology to provide solutions to the changing information system requirements.

The second group, represented by EDS, Anderson, USAA, and Open Engineering (Interviews A.2.1, A.3.1, A.5.1, and A.9.1), views an information systems architecture from a predominantly business-oriented perspective. For this group, the driving architectural issue is the rapid reengineering of business processes to support changing business needs. Although this group clearly understands that an information systems architecture must include technology in its implementation, recent experiences with business process reengineering led each to see changing business requirements as the driving force in developing an information systems architecture (Interview A.5.1). As USAA's architectural group stated, "The IT architecture must support USAA's new business processes defined through our business process reengineering efforts" (Interview A.5.1).

Although the distinction between a business-oriented view and a technology-oriented view of an information systems architecture might not seem obvious or of any significance, the basis for one's view of the information systems architecture does have important consequences for how the resulting architecture deals with change. In the case of the technology-oriented view, architectural efforts are focused on defining "how" the technology can be used to enable changing information systems requirements. By starting with the technological representation of the architecture and moving toward the business representation, the technology-oriented view takes a "how" first approach to the design of the architecture. As VITAL's head architect so

succinctly observed, “(S)ince the business architecture must be able to be adapted to new environments, a complete definition of its functionality cannot be defined *a priori*. Given the lack of a complete ‘Business Architecture,’ the question becomes how can the technical architecture be designed?” (Interview A.7.1).

In contrast to the “how” first technology-oriented view, the business-oriented view of architecture starts by defining “what” the information systems architecture must accomplish from a business perspective independent of any specific technological implementation. By moving from the business representation toward the technological representation, the business-oriented view takes a “what” first approach to the design of architecture. The following quote by Anderson Consulting EIA’s chief architect captures the business-oriented view.

The architecture should specify what we should build, not how-- which means that an architecture is not restricted to one implementation method or technology. Quite the contrary, an architecture should help identify and guide the selection of an appropriate methodology and technology that are necessary to deliver business results over different generations of technologies and methodologies. (Interview A.3.1, HR, p. 130)

The significant difference between the “what” first business-oriented view and the “how” first technology-oriented view is found in “how” information system architectures built using these perspectives deal with changing information systems requirements. Under the “how” first technology-oriented view, architectural issues that have not been resolved (i.e., the required business functionality) must be deferred. In the case of the VITAL architecture, it attempts to deal with changing business requirements by focusing on providing the invariant portion of the system, in this case

the data, and deferring responsibility for integrating the invariant and variant, in this case the process, portions of the system to the end-user (Interview A.7.1).

By contrast, the “what” first business-oriented view “starts with a business driven approach that requires that the business’ goals, policies, culture, and environment be identified, integrated, and then linked to the architecture’s technology goals” (Interview A.3.1). Under the “what” first approach, the information systems architecture must focus on defining the variant portion of the system from the very beginning. The idea underlying the “what” first business-oriented view is perhaps best captured by the observation made by TI’s chief OO architect that “(w)ithout a map of where you are going, its hard to get there” (Interview A.4.1, JM, p. 135).

Although an information systems architecture must possess both a business and technology perspective and deal with changes in both technology and business requirements, the idea that “without a map of where you are going, its hard to get there” (Interview A.4.1) implies that an information systems architecture must be business driven while also being responsive to changes in technology. The requirements that the information systems architecture be capable of dealing with both changing business requirements and changing technology under an approach that enables the requisite business solution led to the statement of Notion 1 and Notion 2 listed below.

Notion 1:                   The ability of an information systems architecture to remain evergreen requires that the information systems architecture’s command, control, and communication mechanism have the capability to deal with both changing business requirements and changing technology.

**Notion 2:** The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism be business driven and technologically responsive.

Given these two notions, the obvious questions that come to mind are (1) "What are the requirements for dealing with change?" and (2) "What are the requirements for being business driven?" In order to explore the answers to these questions, one must first realize that implicit in both of these notions is the idea that business and technology exist as separate domains (Interviews A.2.1, A.3.1, and A.6.1). When viewed as separate domains, the first notion suggests that an information systems architecture represents a mapping between the two domains and the second notion suggests that the mapping should be a directed mapping from the business domain onto the technology domain (Interviews A.3.1 and A.6.1).

Anderson Consulting EIA's chief architect alludes to the idea of an information systems architecture as a mapping by stating that the key to developing one is to define "the linkages between architectural layers which cause technical decisions to be based upon identified business dependencies and relationships" (Interview A.3.1, HR, p. 130). The insight that an information systems architecture represents a mapping across domains allows one to view the architecture's ability to deal with change within a domain as a requirement for a new mapping across the domains. Likewise, the architecture's requirement for being business driven can be viewed as a requirement for first determining the business domain's requirements that are independent of the technology domain's implementation of these requirements. Formally stated, the

requirement for an independent statement of the business domain's requirements become Notion 3.

**Notion 3:** The ability of an information systems architecture to enable the requisite business solution requires the information systems architecture's command, control, and communication mechanism to first determine the business domain's requirements independent of the technology domain's implementation of these requirements.

As a mapping between domains, the overall effectiveness of the architecture will be based upon the quality of the mapping. In CSC's experience, the current quality of information systems architectures is a direct result of poor quality mappings. As CSC PRISM's chief architect stated, "We believe that the underlying problem of architecture is that there is no context for making decisions about technology and its use from a business perspective" (Interview A.6.1, JS, p. 146). To address the problem, CSC's PRISM architecture focuses exclusively on developing a set of "evaluative criteria ... for making architectural decisions" because "(w)ithout a clear set of driving principles, an architecture lacks a clear context for making decisions about technology and its use from a business perspective" (Interview A.6.1, JS, p. 146).

The source of these principles is often described as the organization's values, shared values, or culture (Interviews A.3.1, A.5.1, and A.6.1). CSC PRISM's chief architect stated that "The key to successful elucidation of principles is the discovery of these values, as opposed to their invention" (Interview A.6.1). For USAA's architectural team, capturing and enforcing the shared values that represent USAA's culture is presented as their biggest architectural challenge. The following passage

from the USAA interview helps to explain why an organization's shared values are of such significance in determining the principles of an information systems architecture.

The biggest problem that this architecture faces is its inability to address the issue of culture. Culture is the glue that binds all of USAA together. It represents our shared values, and USAA has a very well defined set of shared values. These shared values define the way USAA views the world, how we work together, how we solve problems, how we deal with conflict and a whole host of other similar issues—how USAA would work with a customer to build an insurance contract. All IT systems exist within some organization. If the culture of the organization is not compatible with the system design, the organization will reject the system. The system must operate and enforce the shared values of the organization. The architectural problem is not only capturing and enforcing shared values but dealing with the fact that these shared values are changing—in large part enabled by new information technology. So the architecture must not only enforce the shared values but also enable the organization to change them when necessary. (Interview A.5.1, CE, p. 143)

Until an organization can enunciate its values, it lacks a context for making any decisions. Because technology should be selected, given its enabling capabilities, its risk, its reliability, its cost, and number of other factors identified by Anderson Consulting, EDS, and CSC, an organization needs to first develop a set of architectural principles in order to build a context within which it can evaluate possible technological choices. As new information technologies are developed, their enabling capabilities, their risks, their reliability, their costs, and all their other factors can be evaluated within that context.

As USAA's architecture team also points out, the shared values of organizations are changing, "in large part enabled by new information technology. So the architecture must not only enforce the shared values but also enable the organization to

change them when necessary” (Interview A.5.1, CE, p. 143). Because changes in shared values must first be captured and then be enforced through the selected mapping between the business and technology domains, capturing and enforcing shared values represent a requirement for being business driven. The formal statement of the requirements for capturing and enforcing the shared values of the organization become Notion 4 and Notion 5 listed below.

**Notion 4:** The ability of an information systems architecture to enable the requisite business solution requires the information systems architecture’s command, control, and communication mechanism to capture the business organization’s shared values independent of their use in the mapping of business domain’s requirement onto the technology domain’s implementation.

**Notion 5:** The ability of the information systems architecture to enable the requisite business solution requires that the information systems architecture’s command, control, and communication mechanism enforce the business organization’s shared values when mapping the business domain’s requirements onto the technology domain’s implementation.

As USAA’s architectural group stated, the reason for enunciating the organization’s shared values is that “These shared values define the way USAA views the world, how we work together, how we solve problems, how we deal with conflict, and a whole host of other similar issues—how USAA would work with a customer to build an insurance contract” (Interview A.5.1, CE, p. 143). Without an enunciated set of shared values, the organization will not possess a necessary condition for the integration of “all of these issues in some consistent manner” (Interview A.2.1, JN, p.



128). The benefit of having a consistent integration is that it enables the organization to simultaneously address multiple criteria.

For organizations holding a predominately technology-oriented view, the fact that a complete information systems architecture requires the integration of all of these issues in some consistent manner is not at issue (Interviews A.7.1 and A.8.1). In fact, organizations in both the business-oriented and the technology-oriented groups believe that a information systems architecture must integrate both data and process in some consistent manner in order to provide a complete information systems solution (Interviews A.2.1, A.3.1, A.5.1, A.7.1, A.8.1, and A.9.1). The major distinction between the two groups is a matter of when and how the integration should take place in order to provide the consistency, speed of integration, and flexibility required for the rapid provisioning of new information systems solutions (Interviews A.2.1 and A.7.1).

Even those organizations holding a very data centric approach to an information systems architecture realize that an architecture must capture meta-data about the data in order to ensure integrity when using the data (Interviews A.1.1 and A.7.1). Because data and process must both be integrated to provide a complete information systems solution, integration of both data and process in some consistent manner becomes a necessary condition for providing the requisite business solution. The formal statement of the requirements for integration of both data and process becomes Notion 6 listed below.

**Notion 6:** The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism integrate both data and process (or rules) in some consistent manner.

Although a consistent integration itself is necessary for the integrity of the solution, when and how the integration is accomplished determines if the solution also simultaneously provides the integrity, speed of integration, and flexibility required for the rapid provisioning of new information systems solutions. In EDS's experience, when using a top-down approach, integration achieved by defining a single, unified set of requirements results in a consistent integration that is too rigid and cumbersome to provide the requisite flexibility or speed for the rapid provisioning of new information systems solutions. EDS's Enterprise Modeling Project indicates that integration achieved by defining a single, unified set of requirements using a top-down approach can provide a consistent integration when the size of the requirements allows everything to be defined, when the static nature of the requirements allows the integration to be defined once and left unchanged, and when the homogeneous nature of the requirements allows one integration to work in all instances (Interview A.2.1).

Although some would consider a bottom-up approach as the antithesis to a top-down approach, EDS's Meta-modeling Project indicates that integration achieved by combining multiple, heterogeneous sets of requirements using a bottom-up approach results in an integration process that suffers from a similar set of limitations as a top-down approach. As the set of requirements becomes more heterogeneous, the ability to achieve consistency in a bottom-up integration becomes increasingly difficult (Interview

2, 4). As the number of requirements (represented by N) increases, the number of linkages between requirements increases as  $(N * (N - 1)) / 2$  under a bottom-up approach resulting in size limitations (Interviews A.2.1 and A.4.1). If the requirements are not static, then the dynamic nature of the requirements can cause all of the corresponding problems found under a top-down approach (Interview A.2.1). In fact, nothing inherent in a bottom-up approach provides any mechanism for the resolution of the complexity created by the size, the dynamic nature, or the heterogeneous nature of the requirements (Interviews A.2.1 and A.4.1).

To deal with the complexity created by the size of the requirements, TI, Anderson Consulting, and Open Engineering are using abstraction to build a layered model of an information systems solution (Interviews A.3.1, A.4.1, and A.9.1). Through the use of abstraction, details of the information systems solution that are immaterial can be suppressed leaving the details or properties of the solution that are significant. By reducing the number of details or properties, abstraction reduces the complexity created by size. The critical issue in the use of abstraction is the determination of what details or properties are material to the question under consideration. The determination of the number of layers of abstraction should be based upon the specific features of the domain of interest.

To contend with the complexity created by the dynamic nature of the requirements, Open Engineering is deferring the binding time of the integrated requirements to the implementation (Interview A.9.1). By deferring the binding time of the integrated requirements to the point in time at which the implementation

executes, run-time binding provides a more dynamic mechanism for dealing with changing requirements than binding the requirements at the point time at which the code is written. If the integrated set of requirements to be bound to the implementation remains homogeneous across all instances within the implementation, then late binding probably represents a necessary and sufficient condition for dealing with the dynamic nature of the requirements. For cases where the integrated set of requirements must be reconfigured for some instances based upon local requirements, late binding of a single predefined integration will not provide the required solution. In order to determine the appropriate point in time at which the requirements should be bound to the implementation, the specific dynamic nature of the requirements under consideration must be considered.

To cope with the complexity created by the need to have heterogeneous resolution of the requirements across different instances within the implementation, Apple and Open Engineering are using local resolution of the requirements across instances (Interviews A.7.1, and A.9.1). By deferring the integration of the requirements to each local instance, a unique local resolution for each instance can be achieved. Although the increased flexibility that local resolution of the requirements provides is desirable, TI's and USAA's experience suggests that deferring the resolution to the local nodes creates some issues of how to ensure the consistency of resolutions across all instances within the implementation (Interviews A.4.1, A.5.1).

TI's attempts to provide "empowered end-user" with localized resolution of financial analyze requirements resulted in the incongruous resolution of these

requirements across departments due to the inconsistent application of corporate wide financial planning requirements and the inaccurate use of both financial analysis models and corporate wide data (Interview A.4.1). In USAA's case, achieving a consistent resolution represented only part of the requirements. In addition to a consistent resolution, USAA's management also wanted the information systems architecture's command, control, and communication mechanism to inform it when a localized set of requirements were combined to cause a global requirement to be altered or overridden (Interview A.5.1). In general, the determination of the appropriate local requirements and the appropriate resolution mechanism should be based upon the specific nature of the instance and organizational goals.

Unless end-users become all knowing, building the requisite business solution by empowering end-users requires that the command, control, and communication mechanism of an information systems architecture provide them with the ability to resolve the complexity of the requirements created by their size, dynamic nature, and heterogeneous nature in some consistent manner. The formal statement of these requirements for the consistent resolution of requirements becomes Notion 7, Notion 8, and Notion 9 listed below.

Notion 7:                   The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism use abstraction to deal with the complexity of the information system's requirements created as the number of requirements increases.

**Notion 8:** The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism utilize an appropriate binding point based upon the dynamic nature of the information system's requirements.

**Notion 9:** The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism integrate both local and global information system's requirements in order to deal with the heterogeneous nature of local instances.

Through the analysis of the interviews conducted for the second round of this research study, nine notions of the requirements of an information systems architecture's command, control, and communication mechanism for the rapid provisioning of information systems solutions have been identified. With the identification of each new notion, an individual piece of the overall picture of what an information systems architecture's command, control, and communication mechanism capable of rapidly provisioning new information systems solutions is discovered.

Although each individual notion is important in its own right, a complete description of the command, control, and communication mechanism requires the integration of these individual notions into a consistent whole. In the next section, the preliminary notions of the requirements of an information systems architecture's command, control, and communication mechanism capable of rapidly provisioning new information systems solutions will be synthesized into a set of propositions using constant comparison.

### Synthesis of Propositions

The synthesis of the preliminary notions generated from the interviews into propositions requires its comparison with insights from existing formal theories (Glaser and Strauss, 1967). In order to compare these notions with the existing literature, one must first identify the theoretical constructs or ideas that these notions address and then cast these notions in terms of these constructs before comparing and synthesizing them into propositions. Of the nine notions generated in the second round of this study, Notions 1, 2, and 3 predominately deal with the idea of a strategic fit between the business domain and the information technology domain within an organization indicating that these notions should be compared with the appropriate strategic alignment literature. Notions 4 and 5 address the issue that culture or shared values play in the development of information systems indicating that these notions should be compared with the appropriate organizational cultural literature. Notion 6 addresses the definition of the components of an information system and a requirement for their integration, indicating that this notion should be compared with the appropriate systems' development literature. Notions 7, 8, and 9 further deal with the requirements of the integration process itself indicating that these notions should be compared with the appropriate systems' architecture literature.

Cast in terms of the idea of a strategic fit between the business domain and the information technology domain within an organization, Notion 1 suggests that the strategic fit between the business and information technology domains is inherently dynamic. Given a dynamic nature, achieving a strategic fit requires that an

organization engage in a continuous process of strategic alignment. Viewed in terms of a process, Notion 2 suggests that the first step in the strategic alignment process is the determination of the business requirements followed at some later point by another step that maps these requirements onto the technology domain. As a process, Notion 3 suggests that the first step in the strategic alignment process, the determination of business requirements, can and should be executed independently from any subsequent steps.

Notion 4 suggests that the determination of business requirements must include the capture of the shared values or social context of the organization. Notion 5 suggests that these shared values must be used in the alignment process at the point when the business requirements are mapped onto the technology domain to ensure an appropriate alignment. Notion 6 suggests that a complete mapping between the business and technology domains must include both data and process and that a mapping must be conceptually or logically consistent. Notion 7 suggests that the determination of the business requirements should use abstraction to deal with the complexity created when the number of issues becomes large. Notion 8 suggests that late binding of the resolved business requirements to the executable implementation should be used to address the dynamic nature of the alignment process. Last, Notion 9 suggests that local resolution of the business requirements should be used to contend with the heterogeneous nature of system requirement within organizations.

The need for the strategic alignment of the business strategy and the information technology strategy is based upon the research of King (1978), Rockart (1979), and



Pyburn (1983) who concluded that a failure to establish a strategic linkage between the business strategy and the information technology strategy results in dysfunctional systems. Building upon a diverse body of earlier work, Venkatraman and Camillus (1984) first introduced the idea of a strategic fit as the consistent alignment or linkage of strategic choices in the external and internal domains of the organization. In its original formulation, a static linkage between the business strategy and information systems function was developed based upon the use of information technology to provide the most efficient and effective implementations of the required business functions that were themselves based upon the chosen business strategy (Henderson and Venkatraman, 1993). The performance criteria for assessing the fit of this linkage is based upon the cost reduction capabilities of the information system.

As long as organizations seek a competitive advantage based solely upon price, the static alignment of information technology with the business strategy through business functions represents a viable strategy (Luftman, Lewis, and Oldach, 1993). When organizations seek to differentiate themselves using distinctive competence, the use of a static linkage proves inadequate to support a sustained competitive advantage (Henderson and Venkatraman, 1993). The two major reasons for the failure of the traditional static linkage model are its inability (1) to recognize that alignment between business strategy and information technology strategy includes both the alignment of the formulation and the implementation of the strategy and (2) to recognize the inherently dynamic nature of the strategic alignment process (Henderson and Venkatraman, 1993).

In order to achieve strategic alignment, Henderson and Venkatraman's (1993) two by two Strategic Alignment Model shows that organizations can select from four strategic alignment perspectives. In all four cases, the perspectives are strategically driven and functionally responsive. Furthermore, Henderson and Venkatraman (1993) point out that the selected perspective must constantly be reevaluated in light of the dynamic nature of strategic alignment. Comparing the results of the strategic alignment literature with Notion 1, Notion 2, and Notion 3 results in the synthesis of the propositions listed below.

**Proposition 1:** The ability of an information systems architecture to remain evergreen requires that the information systems architecture's command, control, and communication mechanism have the capability to deal with both changing strategic requirements and changing functional capabilities.

**Proposition 2:** The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism be strategically driven and functionally responsive.

**Proposition 3:** The ability of an information systems architecture to enable the requisite business solution requires the information systems architecture's command, control, and communication mechanism to first determine the strategic domain's requirements independent of the functional domain's implementation of these requirements.

The idea that culture plays an important part in the success of the introduction of information technology is not new to the information systems field (Ginzberg, 1981). For example, the interactionist approach (Markus, 1983) and the reinforcement

politics approach (George and King, 1991) have examined the role of the social context in shaping the introduction and use of information technology, while the structuration perspective (Orlikowski and Robey, 1991) has emphasized the centrality of players' deliberate, knowledgeable, and reflective action in shaping and appropriating information technology. In other investigations, the use of technology as an agent of cultural change has been studied (Markus and Robey, 1988; Orlikowski, 1993).

Although these approaches differ on the process through which social context and technology interact over time to affect the outcome of the introduction of information technology, all of these approaches emphasize the critical nature of the alignment of the organization's shared social and technological values when introducing new information technology into organizations. Keen (1993), in his Fusion Map approach, states that,

When every leading firm in an industry has access to the same information technology resource, the management difference determines competitive advantage or disadvantage. The management challenge is to make sure that business process, people, and technology are meshed, instead of being dealt with as separate elements in planning and implementation. (p. 17)

In order to integrate business process, people, and technology into a consistent whole, the organization must first identify the value it places on these elements.

Comparing the results of the organization's cultural literature with Notion 4 and Notion 5 results in the synthesis of the propositions listed below.

**Proposition 4:** The ability of an information systems architecture to enable the requisite business solution requires the information systems architecture's command, control, and communication mechanism to capture both the organization's shared social and technological values.

**Proposition 5:** The ability of the information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism enable the alignment of the organization's shared social and technological values.

The decomposition of an information system into data and process dates back to the earliest work on structured programming and design (Yourdon, 1975). Although the general wisdom in the information systems' field holds that data, process, and timing are all necessary for the complete specification of an information system, some researchers studying the impact of information technology on organizations have assumed that the integration of data alone will provide the necessary communication mechanism for the coordination within and across diverse organizations (Huber, 1990; Malone, 1987). However, subsequent research by Goodhue, Wybo, and Kirsch (1992) into the benefits of data integration have shown that data integration alone has not produced the anticipated benefits due to the inability of data integration methodologies to simultaneously represent unique subunit information requirements or perspectives. This conclusion is in complete agreement with Venkatraman's (1986) research that established the need to view organization performance from multiple perspectives.

In order to simultaneously represent multiple perspectives, the multiple perspectives must be logically or conceptually consistent. Because information systems include data, process, and timing as components, the integration of these components into a system must be accomplished in a consistent manner. Comparing the results of

the systems development literature with Notion 6 results in the synthesis of the proposition listed below.

**Proposition 6:**           The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism enable the integration of data, process (or rules), and timing in a consistent manner.

The use of abstraction to deal with the complexity created by size represents a well known and very powerful technique for dealing with the human limitations for processing information (Miller, 1956). Unable to master the entirety of a complex domain, humans choose to ignore inessential details, dealing instead with a generalized, idealized model of the domain through abstraction. In defining an abstraction, Shaw (1984) states that as "a simplified description, or specification, of a system that emphasizes some of the system's details or properties while suppressing others" (p. 8). She continues, "A good abstraction is one that emphasizes details that are significant to the reader or user and suppresses details that are, at least for the moment, immaterial or diversionary" (p. 8). Because abstractions focus on defining common or shared properties, they arise from a recognition of similarities between certain objects, situations, or processes in the real world (Dahl, Dijkstra, and Horace, 1972).

By identifying the essential similarity or pattern, abstraction allows us to capture reusable pieces of information that contain increasingly greater semantic content within the abstraction's selected perspective. The critical issue in the use of abstraction is the determination of what details or properties are material given the abstraction's

perspective. Comparing the results of the systems architecture literature with Notion 7 results in the synthesis of the proposition listed below.

**Proposition 7:**           The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism use abstraction to deal with the complexity of the information system's requirements created as the number of requirements increases.

As part of the process of specifying the information systems' requirements, symbols will need to be defined that represent concepts in the domain of interest. At some point in the implementation process, these symbols will need to be bound to real world instances of the concepts that they represent (Manna and Waldinger, 1985). The choices for a binding time range from the point when the specification is written to the point at which the implemented specification is executed (Tannenbaum, 1990). The later the point that the instances are bound, the more dynamic the solution (Jacobson, 1992). Comparing the results of the systems architecture literature with Notion 8 results in the synthesis of the proposition listed below.

**Proposition 8:**           The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism use an appropriate binding point based upon the dynamic nature of the information systems requirements.

Because the overall design of the relationships between the components within a system determine the emergent properties of the system (Klir, 1985), the study of different design forms has received significant attention in economics (Piore and Sabel,

1984), organization theory (Duncan, 1979; Galbraith, 1973) , computer science (Shaw and Garland, 1996), and information systems (George and King, 1991). Historically, design choices have been viewed in terms of a centralized versus decentralized design, depending upon where the resolution of the issue under consideration takes place. In information systems, the decentralized design's advantage is found in its ability to provide superior flexibility and responsiveness relative to a centralized design's ability to provide efficiency and interoperability in the information systems' solution (George and King, 1991).

Unless a single organization-wide integration can be found that works equally well in all instances, inclusion of instance specific requirements into the requirements' integration process will be necessary (Allen and Boynton, 1991). In order to provide the flexibility required of today's information system solutions, the command, control, and communication mechanism of the information systems architecture must include both local and global information systems' requirements in the requirements' integration process. Comparing the results of the systems architecture literature with Notion 9 results in the synthesis of the proposition listed below.

**Proposition 9:**            The ability of an information systems architecture to enable the requisite business solution requires that the information systems architecture's command, control, and communication mechanism integrate both local and global information systems' requirements in order to deal with the heterogeneous nature of local instances.

In order to understand the overall relationship between the propositions developed in this round, a graphical representation of these propositions and the

implications that link them is presented in Figure 4.1. In Figure 4.1, the functional requirement that the referent architecture remain evergreen requires that the command, control, and communication mechanism of the referent architecture deal with both changing strategic requirements and changing functional requirements. To deal with these changing requirements implies that the command, control, and communication mechanism of the referent architecture must possess the ability to identify and capture these changing requirements as formally defined in Propositions 2, 3, 4 and 7. Once captured, adapting to these changing requirements implies that the command, control, and communication mechanism of the referent architecture must achieve an alignment between these requirements as formally stated in Proposition 5. Achieving an alignment implies that the command, control, and communication mechanism of the referent architecture must have the ability to bind (i.e., Proposition 8) the data, process, and timing (i.e., Proposition 6) of components that exist in different contexts or environments (i.e., Proposition 9).

In synthesizing these propositions through constant comparison with the appropriate literature, the fact that support for all of these propositions can be found in the literature suggests that these propositions represent significant elements of the evolving referent architecture. Although all of these elements are significant, the adequacy of the specification of the command, control, and communication mechanism of a referent architecture depends on not only the completeness but also the correctness of the propositions. Under the grounded theory research methodology, the validation of the correctness and theoretical saturation or the completeness of the referent



architecture is established by the constant comparison of the evolving architecture (as represented by the propositions) with the data and literature by triangulation.

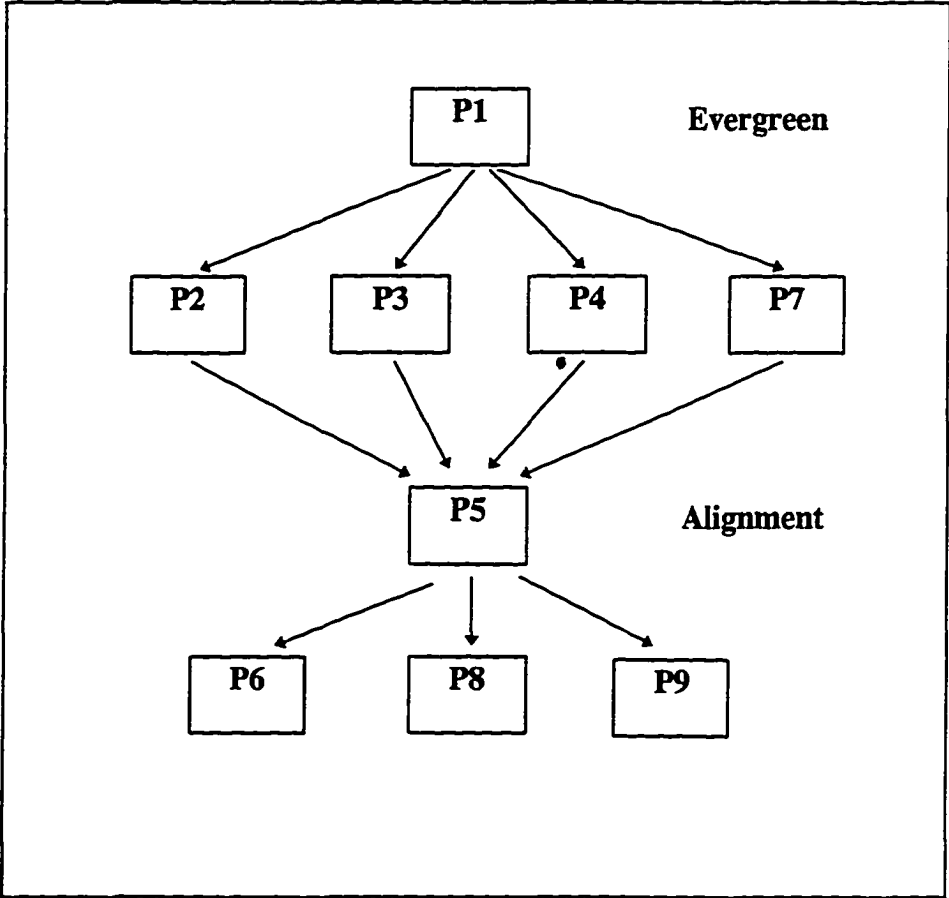


Figure 4.2 Overall Relationship Between Propositions Developed in Round 2

### Validation of Propositions

The validation of the base proposition requires that subsequent propositions, as a group, correctly and completely specify the referent architectural form of the command, control, and communication mechanism. For a referent architecture, the test of the specification's completeness is based upon an implementors ability to build a working instantiation of the referent architecture. Therefore, the validation of the specification must be judged in terms of both its breath and its depth.

To validate the correctness and the completeness of the nine propositions developed in the second round, these nine propositions were shown to representatives of the participating organizations. As a group, these nine propositions where found to be both correct and complete, given their level of specification. However, based upon the comments of reviewers at EDS, TI, and Open Engineering, the level of detail of these propositions was found to be insufficient as a specification of a referent architecture. Specifically, these organizations felt that Proposition 5 needed further investigation in light of the development of the Object Management Group's (OMG's) Common Object Request Broker Architecture (COBRA).

Given these organizations' lack of comments relative to Propositions 1, 2, 3, 4 and 7, one might conclude that these propositions are complete in both terms of their breath and depth. In fact, no conclusion can be drawn at all given the focus of these organizations view that an information systems architecture exists within the functional information technology domain—a view supported by Henderson and Venkatraman's Strategic Alignment Model (1993) that places the information systems architecture

within the internal domain of the organization. In short, these organizations felt that Propositions 1, 2, 3, and 4 were outside the perview required for specifying the referent architectural form of the command, control, and communication mechanism for the rapid provisioning of information systems. Although their conclusion is not, in general, true, their conclusion is correct, since this research study's purpose is to specify a referent architectural form of the command, control, and communication mechanism given that mass customization is selected as the external information technology strategy for the rapid provisioning of information systems.

## CHAPTER V

### THE THIRD ROUND

In order to more fully explore the functional requirements that the command, control, and communication mechanism of the referent architecture must poses to achieve an alignment between the strategic requirements and functional requirements of the enterprise, the third round of this research study focuses on reviewing the architectural principles of the Object Management Group's (OMG) Object Management Architecture (OMA). The OMG, founded in 1989, represents more than 400 companies dedicated to defining a reference architecture for developing and using integrated software systems. Their approach is to adopt specifications based upon available technology and agreed on by member companies. The OMG has defined common terms, interfaces, and a framework for distributed computing in the OMA. In the OMA framework, objects interact through an object request broker that specifies how objects make requests and get responses. To ensure integration, the OMA specifies the basic mechanism that OMA compliant applications must support to use the object request broker (OMG, 1991).

#### Identification of Data Collection Sites

The OMG is organized into task forces that are chartered by the OMG's Technical Committee for the purpose of solving specific problems in the specification of the OMA. The OMG currently charters nine task forces, three of which are

classified as platform task forces and six of which are classified as domain task forces. The platform task forces are the Common Facilities Task Force, Object Analysis and Design Task Force, and the Object Request Broker/Object Services Task Force. The domain task forces include the Business Object Task Force, Electronic Commerce Domain Task Force, Financial Domain Task Force, Manufacturing Domain Task Force, Healthcare Domain Task Force, and Telecommunications Task Force. Of these nine task forces, the Business Object Task Force represents the one chartered with defining the requirements for the integration of independently developed objects into business applications.

As a standards setting body, the OMG has agreed to reuse, through adoption, all International Standards Organization (ISO) and American National Standards Institute (ANSI) standards where appropriate. One such standard that the OMG has agreed to adopt is the ISO's Open Distributed Computing - Reference Model (RM-ODP) (ISO/IEC, 1995). RM-ODP defines five viewpoints that must be specified in order to develop an open distributed information system. These viewpoints are the information, computational, engineering, technology, and enterprise (ISO/IEC, 1995). Under an agreement with ANSI, the enumeration of an enterprise language for specifying the enterprise viewpoint in RM-ODP was delegated to the ANSI's X3H7 Technical Committee. In order to more fully explore the requirements for Propositions 6, 7, 8, and 9, the third round of this research study focuses on collecting the insights of the members of the ANSI's X3H7 Technical Committee and the OMG's Business Object Task Force.

## Analysis of Interviews

To deal with the complexity created by size, the RM-ODP and the OMA use abstraction (Interviews A.10.1 and A.11.1; ISO/IEC, 1995). As a layered model, the OMA uses abstraction based upon the level of detail to define the layers within the architecture (Interviews A.10.1 and A.11.1). Within a layer, the OMA uses abstraction based upon perspectives to define multiple views of that layer (Interviews A.10.1 and A.11.1). By combining abstraction with encapsulation, the OMG believes that the OMA provides a mechanism for flexible instantiation that overcomes the problems of either a pure bottom-up or a pure top-down approach (Interview A.10.1).

To ensure the independence of layers through encapsulation, mechanisms for mapping across layers must be found that are not built upon the inner working of other layers (Interviews A.10.1 and A.11.1). To link across layers using inheritance from or subclassing of classes contained within a higher layer will bind the lower layer to the higher layer's internal structure, resulting in a loss of independence between layers (Interview A.10.1). Instead, mappings that create a clear distinction between "what" a layer must provide and "how" the layer meets its obligations must be found (Interviews A.10.1 and A.11.1).

Within layers, a consistent resolution of the multiple viewpoints of "what" the layer must provide must also be found (Interview A.11.1). Because each viewpoint in the requirements specification is based upon an abstraction developed independently of "how" these viewpoints will be realized, the development of the requirements specification cannot account for the interrelationships between viewpoints until a design

is formulated (Interview A.11.1). Given a design, the interrelationship between viewpoints will either be consistent, inconsistent, or contradictory. To resolve inconsistencies, a partial ordering of the individual viewpoints relative to the issue must be established or a new design specifying “how” the layer’s functionality is to be provided must be devised (Interview A.11.1). To resolve contradictions, a new design specifying “how” the layers functionality is to be provided must be devised (Interview A.11.1).

In specifying a mapping across layers, the specification must enumerate the relationships between aspects of the concepts that exist within each layer (Interview A.11.1). Because all of the aspects of a concept in one layer will rarely, if ever, map onto all of the aspects of a concept in another layer in a one-to-one correspondence, mappings, in general, will produce many-to-many relationships (Interview A.11.1). In addition, the mappings across layers will be constrained both by the design of the layers and the inherent limitations of the underlying concepts themselves (Interview A.11.1). When incomplete or partial mappings are present, inconsistencies or contradictions may result. To resolve these problems, a new specification of “how” the concept will be implemented must be devised (Interviews A.11.1 and A.12.1).

In order for the information systems architecture to use abstraction, the architecture's command, control, and communication mechanism must enable both the consistent resolution of multiple viewpoints within a layer and of concepts mapped across layers. To enable the organization to develop consistent resolutions, the command, control, and communication mechanism of the information systems

architecture must verify, at a minimum, that the implemented resolutions are consistent. Formally stated, the requirements for a consistent mapping both within and across layers become Notion 10 and Notion 11 listed below.

**Notion 10:** The ability of an information systems architecture to enable the use of abstraction requires the information systems architecture's command, control, and communication mechanism to verify that the implemented resolution of mappings between concepts across layers of abstraction is consistent.

**Notion 11:** The ability of an information systems architecture to enable the use of abstraction requires the information systems architecture's command, control, and communication mechanism to verify that the implemented resolution of multiple views within a layer of abstraction is consistent.

To verify the that an implemented resolution is consistent, the information systems architecture must ensure that all shared aspects or interrelationships between viewpoints or layers have been resolved simultaneously (Interviews A.11.1 and A.12.1). Viewing each shared aspect as a dependency along a dimension, interoperability between components within a layer or across layers requires that a consistent set of invariants (i.e., rules that do not change) be simultaneously established for each dependency in the set of dependencies. At a minimum, the architecture must verify, if not actively enable, the resolution of all dependencies.

In resolving dependencies, agreement on both the syntactic form of the specification of the dependency and its semantic content must be achieved (Interview A.12.1). To the extent that limitations in the aspects exists, the specification of the invariant will have to abide by those inherent limitations. Formally stated, the



requirements for establishing a consistent resolution of the shared aspects between viewpoints or layers become Notion 12 and Notion 13 listed below.

**Notion 12:** The ability of the information systems architecture to enable the consistent resolution of the dependencies between viewpoints or layers requires that the command, control, and communication mechanism verify that an invariant has been established for each dependency.

**Notion 13:** The ability of the information systems architecture to enable the consistent resolution of the dependencies between viewpoints or layers requires that the command, control, and communication mechanism verify that both a syntactic and a semantic invariant have been established for each dependency.

In order to deal with complexity created by size, the information system's architecture must enable the use of abstraction. Because individual abstractions emphasize some of the system's properties while suppressing others, important relationships between individual abstractions are often not identified or resolved until a detailed design of the systems specification is undertaken. In order to ensure that all of the perspectives both within a layer of abstraction and across layers of abstraction of the system are simultaneously supported, the specification of the integration of all the perspectives in the detail design must provide a consistent integration.

### Synthesis of Propositions

The four new notions developed in the third round focus on the use of abstraction as a tool in building an information systems architecture. To synthesis these notions into propositions, the validity of the notions must be established through comparison with the appropriate literature. In this section, these four notions are

compared with the systems development literature that focuses on the use of abstraction in systems development.

To properly use abstraction, Berzins, Gray, and Naumann (1986) state that the concept that qualifies as an abstraction must be able to be described, understood, and analyzed independently of the mechanism that will eventually be used to realize the abstraction. When using abstraction across levels of detail, Dijkstra (1982) states that it is “essential for each level to make a clear separation between ‘what it does’ and ‘how it works.’” By separating a level into an external view based upon “what” the level does, that is, a behaviorally based view, and an internal view of “how” the level implements the behavior, an implementation based view, Abelson and Sussman (1985) show that an “abstraction barrier” can be created that allows the abstraction to be described, understood, and analyzed independently of the encapsulated mechanism that will eventually be used to realize the abstraction.

Viewed in terms of an abstraction barrier, the external specification of the behavior of a level of abstraction must take a declarative form while the internal specification of the implementation must take a procedural form (Dijkstra, 1976). Although proper encapsulation allows one to change an implementation with a limited amount of effort, encapsulation by itself does not ensure that any specific implementation preserves the information correctness of the external view (Wirfs-Brock, 1990). To preserve the information correctness of the external declarative view, a consistent mapping between the external view and the internal procedural view of the level must be achieved (Meyer 1988; Hoare, 1994).

In order to use abstraction, the information systems architecture must ensure that any specific implementation of an external view preserves the informational correctness of that view. To preserve the informational correctness of the external view, a consistent mapping between the external view and the internal view of the level must be achieved. Comparing the results of the literature on the use of abstraction with Notion 10 results in the synthesis of the proposition listed below.

**Proposition 10:** The ability of an information systems architecture to enable the use of abstraction requires the information systems architecture's command, control, and communication mechanism to verify that the implemented resolution of mappings between concepts across layers of abstraction preserves the informational correctness of the external view of each layer.

Within a level of abstraction, multiple viewpoints can be defined by the suppression of details irrelevant to the perspective under consideration. Again, a declarative specification of the behavior of each of these viewpoints should be defined independently of the procedural specification of how the specification of the viewpoints will be implemented (Zave, 1993b). To ensure that all of the viewpoints' specifications are simultaneously preserved, the implementation of these viewpoints must satisfy the conjunction of the specific requirements for all viewpoints (Zave, 1993b; Osser, Kaplan, Harrison, Katz, and Kruskal, 1995). Comparing the results of the literature on the use of abstraction with Notion 11 results in the synthesis of the proposition listed below.

**Proposition 11:** The ability of an information systems architecture to enable the use of abstraction requires the information systems architecture's command, control, and communication mechanism to verify that the implemented resolution of viewpoints within layers of abstraction preserves the informational correctness of the conjunction of the external views of each of the individual viewpoints.

In order to rigorously define the behavioral view of a level of abstraction or a viewpoint and correctly map that declarative specification onto a procedural specification, Meyer (1988) believes that each abstraction must be defined in terms of a contract by defining its pre-conditions, post-conditions, and context which is specified by defining the invariants—or what must always be true from the external perspective of the abstraction. In defining and mapping each behavioral specification onto a declarative specification, common dimensions between behavioral views will be found. Given a common dimension between two behavioral views, the consistent resolution of these views will require the consistent resolution between views along common dimensions (Malone and Crowston, 1994; Zave, 1993a). If a consistent resolution is not established, the interaction of the two behavioral views along the common dimension will result in an inconsistent or contradictory set of behaviors (Zave, 1993a).

To ensure the consistent resolution of the behavioral specifications of abstractions, the implementation of the invariants along common dimensions between behavioral views requires that these views implement a consistent set of invariants. To ensure a consistent resolution of the behavioral specifications of abstractions, the information system's architecture must verify that the implementation establish a consistent set of invariants along the common dimension between behavioral

viewpoints. Comparing the results of the literature on the use of abstraction with Notion 12 results in the synthesis of the proposition listed below.

**Proposition 12:** The ability of the information systems architecture to enable the consistent resolution of the dependencies between behavioral abstractions requires that the command, control, and communication mechanism verify that a consistent set of invariants has been established for each dependency between behavioral views.

In order to establish a consistent invariant between behavioral views, the invariant must establish both syntactic and semantic invariance (Dijkstra, 1976; Gries, 1981; Meyer, 1988). Without semantic invariance, syntactic invariance alone will not be sufficient to establish a consistent set of invariants between views. Comparing the results of the literature on the specification of invariants with Notion 13 results in the synthesis of the proposition listed below.

**Proposition 13:** The ability of the information systems architecture to enable the consistent resolution of the dependencies between views requires that the command, control, and communication mechanism verify that both a syntactic and a semantic invariance have been established for each dependency.

In order to present the overall relationship between the propositions developed in this round relative to the proposition developed in the previous two rounds, a graphical representation of all of the propositions and the links between them is presented in Figure 5.1. In Figure 5.1, the functional requirement that the referent architecture's command, control, and communication mechanism achieve an alignment between the strategic requirements and functional requirements of the enterprise implies that the conjunction of different views within each layer of abstraction be conjoined in

a consistent manner (i.e., Proposition 11) and that the mapping between layers of abstraction be isomorphic (i.e., Proposition 10). Taken together, Propositions 6, 8, 9, 10, and 11 imply that the command, control, and communication mechanism of the referent architecture must have the ability to bind (i.e., Proposition 8) the data, process, and timing (i.e., Proposition 6) of different views of a component within each layer of abstraction in a consistent manner (i.e., Proposition 11) while also binding the data, process, and timing of the different components that exist in different contexts (i.e., Proposition 9) across layers of abstraction (i.e., Proposition 10).

To integrate different views or components implies that the command, control, and communication mechanism of the architecture must simultaneously resolve all of the common dimensions (i.e., dependencies) between views or components by establishing a rule or protocol for how each view or component will utilize each common dimension (i.e., Proposition 12). Establishing a common rule or protocol requires that the command, control, and communication mechanism of the architecture establish a common set of concepts by defining their meaning (i.e., Proposition 13) and a notation that can be used to express them (i.e., Proposition 13).

In synthesizing these propositions through constant comparison with the appropriate literature, the fact that support for all of these propositions can be found in the literature suggests that these propositions represent significant elements of the evolving referent architecture. Although all of these propositions are significant, the adequacy of the specification of the command, control, and communication mechanism

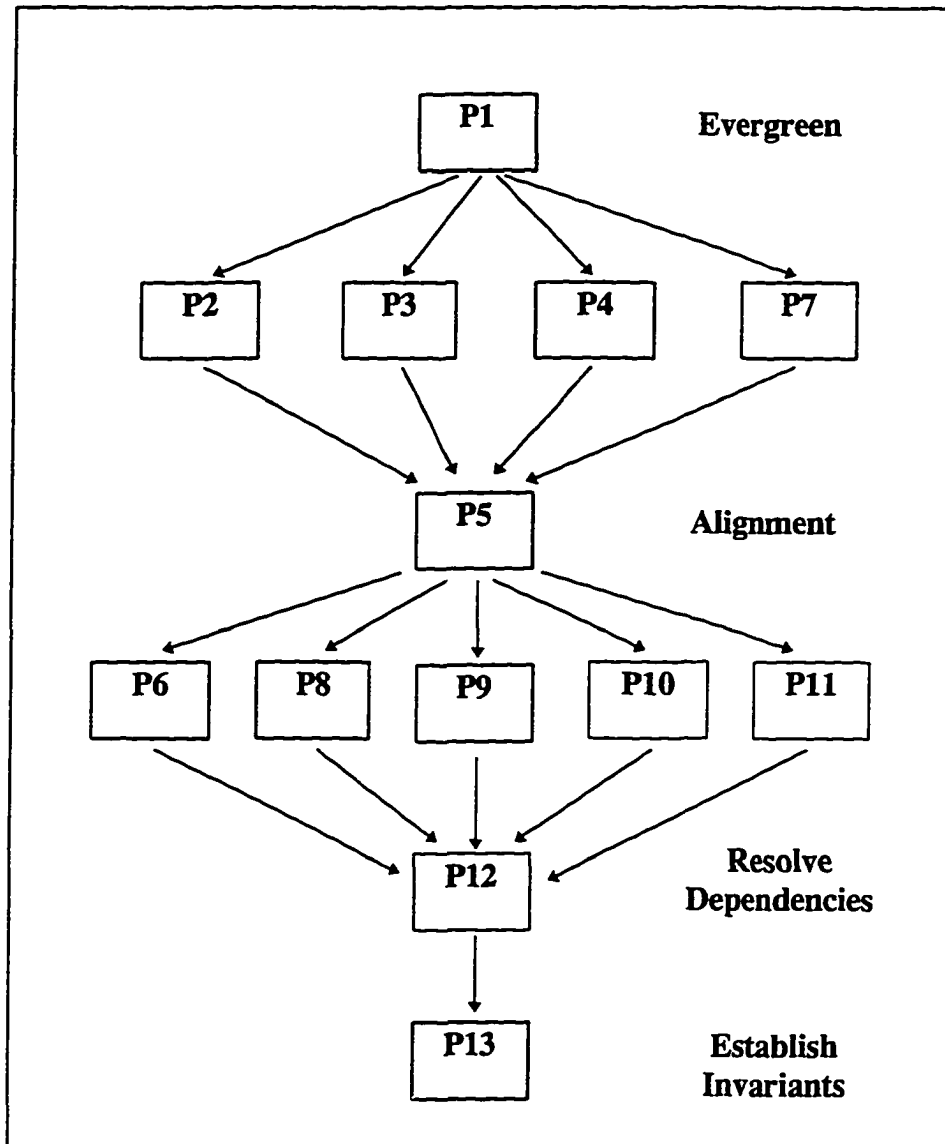


Figure 5.1 Overall Relationship Between Propositions Developed in Round 2 and 3

as a referent architecture depends on the completeness of the propositions as well their correctness. Under the ground theory research methodology, the correctness and the completeness of the referent architecture are established by the constant comparison of the evolving architecture with the data and literature.

### Validation of Propositions

To validate the correctness and the completeness of the four propositions developed in the third round, these four propositions were shown to representatives of the participating organizations. As a group, these four propositions were found to be both correct and complete given their level of specification and the experience of the members with architectural implementations. However, the members of the ANSI's X3H7 Technical Committee and the OMG's Business Object Task Force felt that the completeness of the set of propositions should be verified by comparing the propositions to known architectural implementations that have the same stated goals as this research effort. Specifically, these groups recommended that the set of propositions be compared with SEMATECH's CIM Framework.



## CHAPTER VI

### THE FOURTH ROUND

The fourth round of this research study focuses on learning more about the functional requirements of establishing a consistent resolution between multiple viewpoints within a layer of abstraction and between concepts across layers of abstraction. To gain insight into these requirements, the members of the OMG's Business Object Task Force and the ANSI's X3H7 Technical Committee felt that this research effort would benefit by studying any existing architectural frameworks built to facilitate plug-compatible software solutions. To the knowledge of the OMG's Business Object Task Force and the ANSI's X3H7 Technical Committee, SEMATECH's CIM (Computer Integrated Manufacturing) Framework represents the largest know commercial implementation of a plug-compatible software solution.

#### Identification of Data Collection Sites

SEMATECH's CIM Framework is composed of seven semi-conductor manufacturing abstractions or components that are typically embodied in a wafer fab's Manufacturing Execution System (MES) (SEMATECH, 1994). For each one of these components in SEMATECH's CIM Framework, SEMATECH maintains a dedicated component development team that is responsible for specification of the component's functionality, services, and interfaces. In addition to these component specific teams, SEMATECH maintains a dedicated CIM Framework Architecture Team that is responsible for

resolving dependencies between components in order to ensure interoperability. Given the focus of the fourth round of this reset study, SEMATECH's CIM Framework Architecture Team was selected as the appropriate data collection site.

Through contacts at the OMG and ANSI, SEMATECH was contacted and asked to participate in this research effort. Starting with an invited presentation to SEMATECH's CIM Framework Architecture Team, SEMATECH and its member companies ultimately agreed to provide unlimited access to SEMATECH's CIM Framework Architecture Team and proprietary implementations of its framework by member companies. To achieve this access, non-disclosure agreements were signed with SEMATECH and all member companies. Although SEMATECH's CIM Framework Specification has been released into the public domain, the knowledge gained through member-company implementations of the CIM Framework specification remains proprietary.

#### Analysis of Interviews

Although a number of issues exist within the CIM Framework, all but one of these issues were found to represent the implications of the previously discovered propositions rather than new insights into the functional requirements for a plug-compatible component architecture. By studying CIM Framework implementation projects, a new insight for establishing interoperability between components was found to exist. In addition to the need to establish syntactic and semantic invariance,

interoperability between components requires that the components share a common problem-solving or solution approach (Interview A.13.1).

Since multiple solution approaches exist for most system specifications, the interoperability of plug-compatible components requires that components share a common solution approach. Because the solution approach represents a dependency between components, the requirement that components establish cognitive invariance between components represents an extension of the requirements already detailed in Notion 13—that components establish syntactic and semantic invariance. By extending Notion 13 to include this new requirement, the formal statement of Notion 13 becomes Notion 13a listed below.

Notion 13a:           The ability of the information systems architecture to enable the consistent resolution of the dependencies between viewpoints or layers requires that the command, control, and communication mechanism verify that a syntactic, semantic, and cognitive invariant has been established for each dependency.

### Synthesis of Propositions

Because multiple cognitive approaches exist that will fulfill the same requirements, interoperability between components requires that components share a common cognitive approach (Hoare 1994; Shaw and Garland, 1996). Because the specification of the common cognitive approach is required for a consistent resolution of the dependencies between multiple views, establishing a common cognitive approach represents a requirement for the information systems architecture's command, control,

and communication mechanism. Comparing the results of the literature on software architecture with Notion 13a results in the synthesis of the proposition listed below.

**Proposition 13a:**        The ability of the information systems architecture to enable the consistent resolution of the dependencies between views requires that the command, control, and communication mechanism verify that syntactic, semantic, and cognitive invariance have been established for each dependency.

By substituting Proposition 13a for Proposition 13 in Figure 5.1, Figure 6.1 shows the overall relationship between all of the propositions developed in rounds 2, 3, and 4 of this research effort.

#### Validation of Propositions

All of the functional requirements previously formulated as propositions were found to be necessary to the design of a plug-compatible information systems architecture by SEMATECH's CIM Framework Architecture Team. Although the members of the CIM Framework Architecture Team, OMG's Business Object Task Force, and the ANSI's X3H7 Technical Committee could not think of any additional requirements or issues that should be addressed, none of the members of these groups could determine if the set of propositions, as a whole, represented a complete specification of the referent architecture without providing a complete implementation. Given the existence of theoretical saturation in this research effort, a final validation of the propositions was performed by integrating the set of propositions into a model of an information systems architecture for presentation to the CIM Framework Architecture

Team, OMG's Business Object Task Force, and the ANSI's X3H7 Technical Committee. In the next chapter, the integrated model is presented along with the comments of these groups.

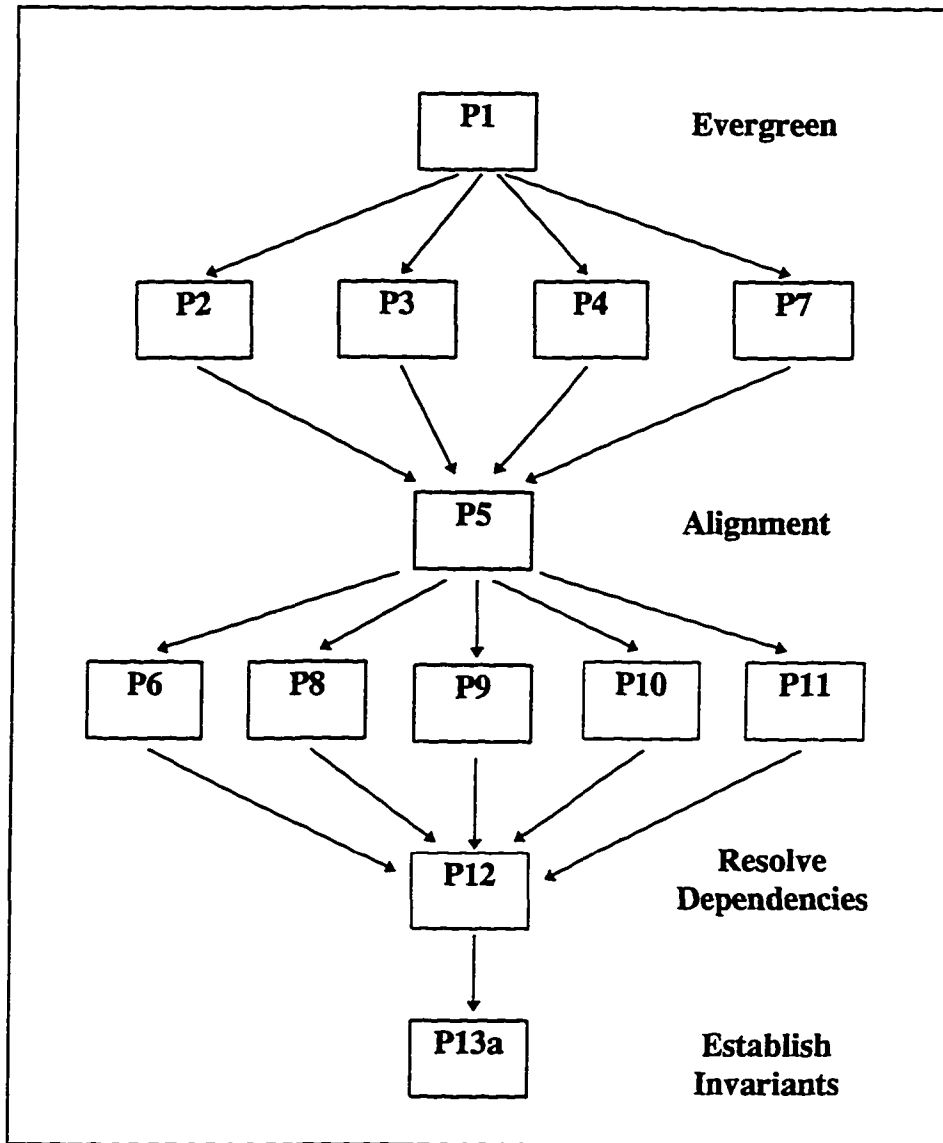


Figure 6.1 Overall Relationship Between Propositions Developed in Round 2, 3, and 4

## CHAPTER VII

### SUMMARY

In order to help validate the set of propositions discovered through this research effort, a model based upon these propositions was developed and shown to members of the OMG's Business Object Task Force and the ANSI's X3H7 Technical Committee. In this chapter, an overview of the model is presented along with some of its implications.

#### Integrating the Propositions

The first three propositions imply that a business exists as a bounded system within an environment as shown in Figure 7.1. As a bounded system, the emergent properties of the system, visible to its external environment, result from the architectural structure of the interrelationships between its internal components. In today's environment, changes both in the external requirements and in the capabilities of the system's internal components are commonplace. To deal with these changes, the information systems architecture must enable the reconfiguration of the structure of the interrelationships between the internal components either to meet changing external requirements or to accommodate new internal component capabilities (Proposition 1).

To help ensure that the requisite external solution is enabled, the architectural design process must be focused on providing the required set of emergent properties by specifying an appropriate architectural structure for the internal components

(Proposition 2). Because “what” the architectural structure must produce needs to be known before the design process can determine “how” to provide it, a independent declarative specification of the external requirements represents the first step in building the requisite business solution (Proposition 3).

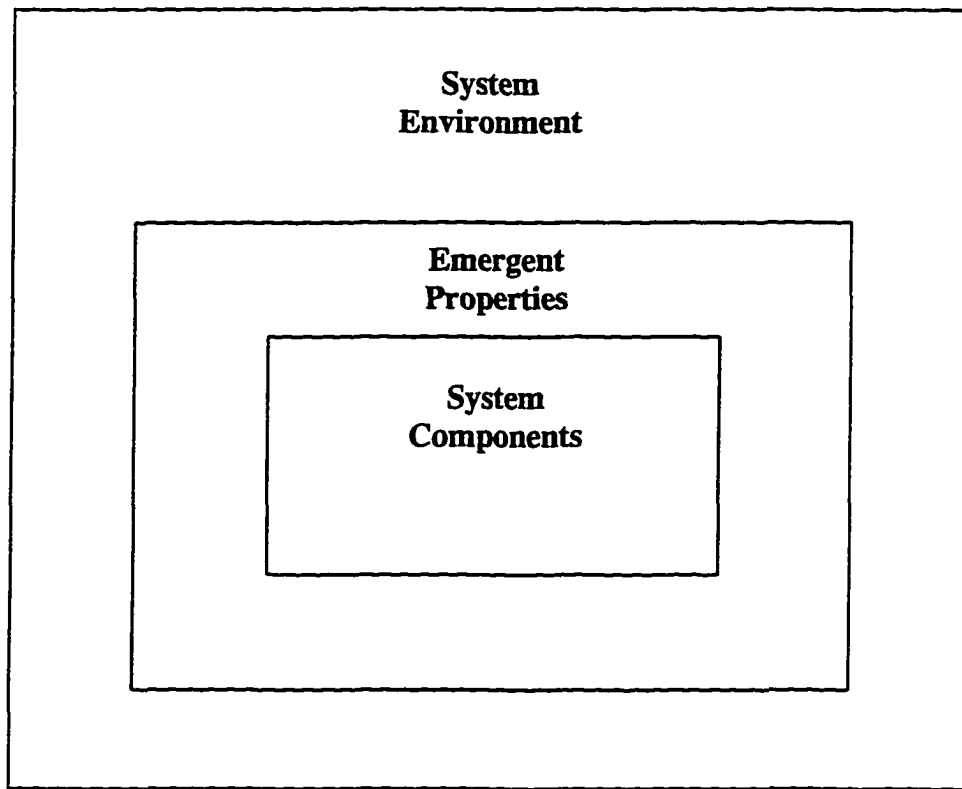


Figure 7.1 A System and Its Environment

In order to specify the external requirements, an organization must decide what aspects of its environment are relevant given its objectives. Because an organization lacks a context for making any decisions until it enunciates its values, an organization



must explicitly state the worth that it places on each aspect of its external environment in order to specify its external requirements (Proposition 4). In designing a system whose emergent properties will meet the external requirements, an organization must also decide what aspects of technology are relevant and explicitly state the worth it places on the aspects of the technology that it selects in implementing its internal components (Proposition 4).

Unless an organization can rigorously specify the value of all relevant aspects, it will lack a means for measuring the extent to which the emergent properties meet the external requirements. If the size of the declarative specification of the external requirements is large, abstraction should be used to reduce the specification to a manageable size (Proposition 7). In forming these abstractions, abstractions based both on the level of detail and perspective can be used. Given an explicitly started set of values, the quality of the fit between the emergent properties and the external requirements will depend on the ability of the information systems architecture to produce a design that aligns all relevant aspects as consistently as possible (Proposition 5). When inconsistencies or contradictions arise, management must specify a partial ordering to resolve the inconsistency. In the case of contradictions, a new design stating “how” the relevant aspects are to be integrated must be specified .

In designing the structure of the interrelationships between the internal components, the information systems architect must specify not only “how” but also “when” the internal components will interact with each other in order to provide the emergent properties. Therefore, the complete system specification must include a

description of “how” and “when” the internal components will interact to produce the emergent properties in addition to a declarative specification of the external requirements of the system (Proposition 6).

As can be seen in Figure 7.2, the top-down mapping of these declarative abstractions onto actual physical entities within the level of detail requires the physical entities to assimilate multiple abstractions or roles both from different viewpoints and from different levels of detail. For the entities in the physical implementation to interoperate properly, a consistent integration of these abstractions must be achieved.

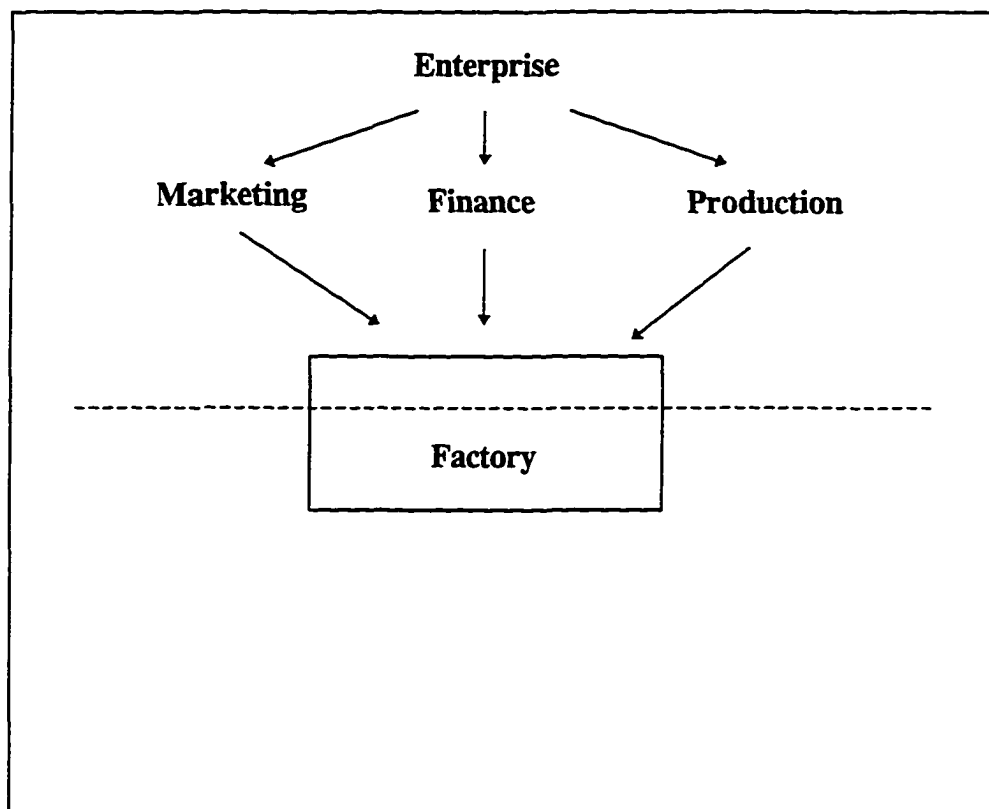


Figure 7.2 Integration of Views

One way to achieve a consistent integration of multiple abstractions is to use ubiquitous standardization. Under ubiquitous standardization, all contradictory or inconsistent integrations are resolved by a centralized authority that specifies a one-size-fits-all solution. Since all the physical entities are standardized, the centrally specified rule will always result in a consistent integration.

If a single organization-wide integration cannot be found that works sufficiently well in all instances, then inclusion of instance specific requirements into the integration process will be necessary (Proposition 9). In this case, the integration of the abstractions cannot be determined *a priori* since the resolution or integration of the abstractions cannot be performed until after the instance specific requirements become known (Proposition 8). To ensure that the emergent properties of the system are preserved during the integration process, the resolution both of abstraction across levels of detail and of perspectives must preserve the informational correctness of the each view (Proposition 10 and 11).

In addition to requiring a consistent resolution of abstractions within each physical entity, interoperability under the plex structure defined in Figure 7.2 also requires a consistent resolution of abstractions between physical entities. For each shared aspect or dimension that exists between physical entities, a common resolution of the dimensions must be established in order to achieve interoperability (Proposition 12). In general, a common resolution requires that a syntactic, semantic, and cognitive invariant be established for each dependent dimension between physical entities (Proposition 13a).

### Implications of the Model

Although this model seems deceptively simple, the implications of this model represent a significant departure from the current architectural principles of at least two of the most prominent architectural efforts in the area, for example, SEMATECH's CIM Framework and the OMG's OMA. Where the OMG and SEMATECH take a bottom-up approach to the design of an architecture, this result shows that achieving interoperability with flexibility requires the use of a top-down layered architecture that can be instantiated layer by layer. While SEMATECH and the OMG believe that the specification of a syntactic interface between components is sufficient for defining interoperability, this research shows that a syntactic, semantic, and cognitive invariant must be established between components to achieve interoperability.

In addition to these two issues, this research also implies that the current implementation of the object-oriented paradigm does not contain the necessary and sufficient capabilities required for building a mass customization based software architecture. Under its current implementation, the object-oriented paradigm assimilates, resolves, and binds the declarative abstractions to methods at the point in time that a method is written. Although methods can subsequently be inherited, no mechanism exists for identifying or resolving inconsistencies or contradictions between methods inherited through multiple inheritance. Until the object-oriented paradigm can provide mechanisms for dynamically inheriting new types and consistently resolving dependencies between them before binding, the paradigm will not contain the

functional requirements necessary for a mass customization based software architecture.

### Validation of the Model

As this model developed through the course of this research effort, the model, along with its implications, was presented to the OMG's Business Object Task Force, ANSI's X3H7 Technical Committee, and SEMATECH's CIM Framework Architecture Team. Because the implications of the model were completely contradictory to some of the basic architectural principles of the OMA and the CIM Framework, the initial reaction to the model was, at times, openly hostile.

Fortunately, a few members of the ANSI's X3H7 Technical Committee felt that the implications of the model were correct from the beginning. Based upon their support, a dialog addressing the implication of the referent architecture was established within the OMG's Business Object Task Force that continues to flourish today. As a result of this dialog, the OMG in its March 1997 meeting in Austin, Texas, a specifically recognized the need to specify its architecture in a top-down layered manner and also formed the Omega's Formal Semantics Specification Special Interest Group.

In addition to the acceptance of two of the implications of this model, the Omega's Business Object Task Force is currently debating how roles should be specified and what will be required to identify and then resolve inconsistencies or contradictions between them. At this time, the Omega's Business Object Task Force,

ANSI's X3H7 Technical Committee, and SEMATECH's CIM Framework Architecture Team believe that the basic propositions of the referent architecture developed through this research effort are correct and most likely complete.

CHAPTER VIII  
LIMITATIONS, CONTRIBUTIONS,  
AND FUTURE RESEARCH

Limitations of the Research

In this exploratory research study, as in all research studies, a number of limitations exist. To help ensure that the research findings are properly interpreted within the context of the selected research methodology, the limitations of this research study must be understood.

Because this research study is based upon an exploratory research methodology, there exists no provably correct way to test that the set of generated propositions in this study are complete or correct. Instead, the internal validity, external validity, and the completeness of the set of propositions was established by achieving a consensus among the participating subject matter experts.

If the subject matter experts participating in this study were to represent the “correct” set of all knowing subject matter experts, then the consensus of these experts would, in fact, ensure that the set of propositions are complete and correct. Short of finding the “correct” set of all knowing subject matter experts, the validity of the findings of this research study are directly related to the limitations found in the group of subject matter experts participating in this study.

Since no known solution exists to the issues addressed in this research study, one can assume that either no subject matter experts exist that have identified a solution

or that the subject matter experts that do exist that have identified a solution are not willing to share their solution with anyone--for competitive reasons. Because all of the subject matter experts participating in this study are members of nationally recognized standard setting bodies, the most one can say about the finding of this research study is that the best available subject matter experts found them to be internally consistent, externally valid, and complete.

### Contributions of the Research

To date, the output of this research effort represents a number of contributions towards the goal of providing rapid development of flexible software solutions. These contributions include:

- The identification of mass customization as the referent paradigm for the rapid development of flexible software solutions,
- The identification of architecture as the critical issue in the development of mass customization solution,
- The identification of command, control, and communication as the critical issue in the development of an architecture,
- The definition of the functional requirements of a command, control, and communication mechanism for a mass customization based architecture,
- The development of a theoretical foundation for the flexible software factory.



### Future Research Directions

In order to provide a complete instantiation of the propositions discovered in this research study, a number of issues will need to be addressed through future research efforts. These research efforts should address the following issues:

- Refine, develop, and test the propositions developed in this research effort,
- Refine and develop the notions of command, control, and communication mechanism for a information system architecture,
- The identification of the dimensions of a software component within a given level of abstraction.

## REFERENCES

- Abelson, H. and Sussman, G. (1985). Structure and Interpretation of Computer Programs. Cambridge: The MIT Press.
- Allen, B. R., & Boynton, A. C. (1991). Information architecture: In search of efficient flexibility. MIS Quarterly, 15, 435-445.
- Adler, P. S. (1991). Workers and flexible manufacturing systems: Three installations compared. Journal of Organizational Behavior, 12(5), 447-460.
- Barnes, B., & Bolinger, T. (1991). Making reuse cost effective. IEEE Software, 8 (1), 13-24.
- Berzins, V., Gray, M., & Naumann, D. (1986). Abstraction-based software development. Communications of the ACM, 29 (5), 1986, 403-412.
- Blau, P. M., & Schoenherr, P. A. (1971). The Structure of Organizations. New York: Basic Books.
- Boynton, A. C., & Victor, B. (1991). Beyond Flexibility: Building and Managing the Dynamically Stable Organization. California Management Review, 34 (1), 53-66.
- Boynton, A. C., Victor, B., & Pine, B. J. (1993). New competitive strategies: Challenges to organizations and information technology. IBM Systems Journal, 32 (1), 40-64.
- Burns, T., & Stalker, G. M. (1961). The Management of Innovation. Chicago: Quadrangle Books.
- Chandler, Jr., A. D. (1962). Strategy and Structure: Chapters in the History of American Enterprise. Cambridge: The M.I.T. Press, Cambridge.
- CIM-OSA (Computer Integrated Manufacturing - Open Systems Architecture). (1993). ESPRIT Project 688. Brussels, Belgium: ESPRIT Consortium AMICE.
- Codd, E. F. (1971). A relational model of data for large shared data banks. Communications of the ACM, 13 (6), 377-387.
- Cusumano, M. (1989). The software factory: A historical interpretation. IEEE Software, 23-30.

- Cusumano, M. A. (1991). Japan's Software Factories: A Challenge to U.S. Management. New York: Oxford University Press.
- Dahl, O., Dijkstra, E., & Horace, C. A. R. (1972). Structured Programming. London, England: Academic Press.
- Davis, S. M. (1987). Future Perfect. Reading, MA: Addison-Wesley.
- Delbecq, A. L., Van de Ven, A. H., & Gustafson, D. H. (1975). Group Techniques for Program Planning: A Guide to Nominal Group and Delphi Processes. Glenview, IL: Scott-Foresman.
- Dertouzos, M., Lester, R. K., Solow, R. M., & the MIT Commission for Industrial Productivity. (1989). Made in America: Regaining the Productive Edge. Cambridge: The MIT Press.
- Dijkstra, E. W. (1976). A Discipline of Programming. Englewood Cliffs, NJ: Prentice Hall.
- Dijkstra, E. W. (1982). Selected Writings on Computing: A Personal Perspective. Berlin, Germany: SpringerVerlag.
- Drucker, P. F. (1995). The information executives truly need. Harvard Business Review, 73 (1), 54-62.
- Duncan, R. (1979). What is the right organization structure? Organization Dynamics. Winter, 59-80.
- Eisenhardt, K. M. (1989). Building Theories from Case Study Research. Academy of Management Review, 14, 532-550.
- Galbraith, J. R. (1973). Designing Complex Organizations. Reading, MA: Addison-Wesley.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns -- Elements for Reusable Oriented Software. Reading, MA: Addison-Wesley.
- George, J. F., and King, J. L. (1991). Examining the computing and centralization debate. Communications of the ACM, 34 (7), 63-77.
- Ginzberg, M. J. (1981). Early Diagnosis of MIS Implementation Failures: Promising Results and Unanswered Questions. Management Science, 27 (4), 459-478.

- Glaser, B. G. (1978). Theoretical Sensitivity: Advances in the Methodology of Grounded Theory. Mill Valley, CA: The Sociology Press.
- Glaser, G. B., and Strauss, A. L. (1967). The Discovery of Grounded Theory: Strategies for Qualitative Research. New York: Aldine Publishing Company.
- Goodhue, D. L., Wybo, M. D., & Krisch, L. J. (1992). The Impact of Data Integration on the Costs and Benefits of Information Systems. MIS Quarterly, 16, 293-311.
- Gries, D. (1981). The Science of Programming. Berlin, Germany: Springer Verlag.
- Griss, M. L. (1993). Software Reuse: From Library to Factory. IBM Systems Journal, 32 (4), 548-566.
- Haeckel, S. H., & Nolan, R. L. (1993). Managing by wire. Harvard Business Review, 71 (5) 122-132.
- Hammer, M. (1990). Reengineering works: Don't automate, obliterate. Harvard Business Review, 104-112.
- Henderson, J. C., & Venkatraman, N. (1993). Strategic alignment: Leveraging information technology for transforming organizations. IBM Systems Journal, 32 (1), 4-16.
- Hoare, C. A. R. (1994). Mathematical Models for Computing Science. United Kingdom: Oxford.
- Huber, G. (1990). A theory of the effects of advanced information technologies on organizational design, intelligence, and decision making. Academy of Management Review, 15, 47-71.
- Humphrey, W. (1989). Managing the Software Process. Reading, MA: Addison-Wesley.
- IBM Corporation. (1984). Business Systems Planning: Information Systems Planning Guide (4th ed.). White Plains, NY: .
- Iman, R. L., and Conover, W. J. (1989). Modern Business Statistics. New York: Wiley.
- ISO/IEC JTC1/SC21/WG7. (1995). Open Distributed Processing - Reference Model: Part 2: Foundations. (IS 10746-2 / ITU-T Recommendation X.902).

- Jacobson, I. (1992). Object-Oriented Software Engineering Reading, MA: Addison-Wesley.
- Johnson, H. T. & Kaplan, R. S. (1987). Relevance Lost: The Rise and Fall of Managerial Accounting. Boston: Harvard Business School Press.
- Keen, P. G. W. (1993). Information technology and the management difference: A fusion map. IBM Systems Journal, 32 (1), 17-39.
- King, W. R. (1978). Strategic Planning for Management Information Systems. MIS Quarterly, 2, 27-37.
- Klir, G. J. (1985). Architecture of Systems Problem Solving. New York: Plenum Press.
- Luftman, J. N., Lewis, P. R., & Oldach, S. H. (1993). Transforming the enterprise: The alignment of business and information technology. IBM Systems Journal, 32 (1), 198-221.
- Malone, T. W. (1987). Modeling coordination in organization and markets. Management Science, 33, 1317-1332.
- Malone, T. W., and Crowston, K. (1994). Toward an interdisciplinary theory of coordination. Computing Surveys, 26, (1), 87-119.
- Manna, Z., and Waldinger, R. (1985). The Logical Basis for Computer Programming. Reading, MA: Addison-Wesley.
- Markus, M. L. (1983). Power, Politics, and MIS Implementation. Communications of the ACM, 26 (6), 430-444.
- Markus, M. L. (1992). Will The Real Qualitative Researcher Please Stand Up? Alternative Approaches to Theory Building Research. The Claremont Graduate School, Programs in Information Science, Working Paper.
- Markus, M. L., and Robey, D. (1988). Information technology and organization change: Causal structure in theory and research. Management Science, 34 (5), 583-598.
- Martin, J. M. (1990). Information Engineering (Volumes 1, 2, and 3). Englewood Cliffs, NJ: Prentice-Hall.
- McGregor, C., and Sykes, D. (1991). A Paradigm for Reuse. American Programmer, 4 (10), 30-39.

- Meyer, B. (1988). Object-Oriented Software Construction. Englewood Cliffs, NJ: Prentice Hall.
- Miles, M. B. and Huberman, A. M. (1984). Qualitative Data Analysis: A Sourcebook of New Methods. Newbury Park, CA: Sage Publications.
- Miles, R. E., and Snow, C. S. (1986). Organizations: New concepts for new forms. California Management Review, 28 (3), 62-73.
- Miller, G. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. The Psychological Review, 63 (2), 86-95.
- Nonaka, I. (1988). Creating organizational order out of chaos: Self-renewal in Japanese firms. California Management Review, 30 (3), 57-73.
- Niederman, F., Brancheau, J. C., & Wetherbe, J. C. (1991). Information systems management issues for the 1990's. MIS Quarterly, 15, 475-500.
- OMG. (1991). Common Object Request Broker Architecture and Specification. Object Management Group, Document 91.12.1.
- Orlikowski, W. J. (1993). CASE tools as organizational change: Investigating incremental and radical changes in systems development. MIS Quarterly, 17, 308-340.
- Orlikowski, W. J. & Robey, D. (1991). Information technology and the structuring of organizations. Information Systems Research, 2 (2), 143-169.
- Osser, H., Kaplan, M., Harrison, W., Katz, A., Kruskal, V. (1995). Subject-oriented composition rules. OOPSLA '95 ACM SIGPLAN Notices, 30 (10), 235-264.
- Pawson, R., Bravard, J., & Cameron, L. (1995). The Case for expressive systems. Sloan Management Review, 36 (2), 41-48.
- Porter, M. (1985). Generic competitive strategies, in Competitive Advantage. New York: The Free Press.
- Pine, B. J. (1993). Mass Customization: The New Frontier in Business Competition. Boston: Harvard Business School Press.
- Pine, B. J., Victor, B., & Boynton, A. C. (1993). Making mass customization work. Harvard Business Review, 93 (6), 108-119.

- Piore, M. J. & Sable, C. F. (1984). The Second Industrial Divide: Possibilities for Prosperity. New York: Basic Books.
- Prieto-Diaz, R. (1993). Status report: Software reusability. IEEE Software, 10 (5), 61-66.
- Pyburn, P. J. (1983). Linking the MIS Plan with corporate strategy: An exploratory study. MIS Quarterly, 7, 1-14.
- Quinn, J. B., & Paquette, P. C. (1990). Technology in services: Creating organizational revolutions. Sloan Management Review, 31 (2), 67-78.
- Rockart, J. F. (1979). Chief executives define their own data needs. Harvard Business Review, 81-93.
- SEMATECH. (1994). Computer Integrated Manufacturing (CIM) Application Framework Specification 1.1. Austin, TX.
- Shaw, M. (1984). Abstraction techniques in modern programming languages. IEEE Software, 1 (4), 8-14.
- Shaw, M., & Garland, D. (1996). Software Architecture: Perspectives on an Emerging Discipline. Englewood Cliffs, NJ: Prentice Hall.
- Sowa, J. F., & Zachman, J. C. (1992). Extending and formalizing the framework for information systems architecture. IBM Systems Journal, 31 (3), 590-616.
- SRI International. (1993). Implementing Software Reuse: Phase 1. Key Findings and Conclusions. Menlo Park, CA.
- Stalk, Jr., G. (1988). Time--The next source of competitive advantage. Harvard Business Review, 66 (4), 41-51.
- Stecher, P. (1993). Building business and application systems with the retail application architecture. IBM Systems Journal, 32 (2),.
- Strauss, A. L. (1987). Qualitative Analysis for Social Scientists. Cambridge: Cambridge University Press.
- Strauss, A. L., & Corbin, J. (1990). Basics of Qualitative Research: Grounded Theory Procedures and Techniques. Newbury Park, CA: Sage Publications.

- Swanson, K., McComb, D., Smith, J., & McCubbrey, D. (1991). The application software factory: Applying total quality techniques to systems development. MIS Quarterly, 15, 566-579.
- Tannenbaum, A. S. (1990). Structure Computer Organization. Englewood Cliffs, NJ: Prentice-Hall.
- Taylor, D. A. (1995). Business Engineering with Object Technology. New York: Wiley.
- Taylor, F. W. (1911). Scientific Management. New York: Harper.
- Texas Instruments. (1994). Reengineering of Information Technology at Texas Instruments: Our Challenge – Change Faster Than Change. Plano, TX.
- Thompson, J. D. (1967). Organizations in Action. New York: McGraw-Hill.
- Toffler, A. (1990). Poweshift. New York: Bantam Books.
- Van Maanen, John (Ed.). (1983). Qualitative Methodology. Newbury Park, CA: Sage Publications.
- Venkatraman, N. (1986). Measurement of business performance in strategic research: A Comparison of approaches. Academy of Management Review, 11, 801-814.
- Venkatraman, N., & Camillus, J. C. (1984). Exploring the concept of 'fit' in strategic management. Academy of Management Review, 9, 513-525.
- Wallace, W. L. (1971). The Logic of Science in Sociology. New York: Aldine.
- Wheeler, E. F., & Ganek, A. G. (1988). Introduction to Systems Application Architecture. IBM Systems Journal, 27 (3), 250-263.
- Womack, J. P., Jones, T. J., & Roos, D. (1990). The Machine that Changed the World. New York: MacMillan.
- Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). Designing Object-Oriented Software. Englewood Cliffs, NJ: Prentice Hall.
- Yin, R. K. (1989). Case Study Research: Design and Methods (revised ed.). Newbury Park, CA: Sage Publications.
- Yourdon, E. (1975). Techniques of Program Structure and Design. Englewood Cliffs, NJ: Prentice Hall.



- Yourdon, E., & Constantine, L. (1975). Structured Design: Fundamentals of a Discipline and Computer Program and Systems Design. Englewood Cliffs, NJ: Yourdon Press.
- Zachman, J. A. (1987). A framework for information systems architecture. IBM Systems Journal, 26 (3), 276-292.
- Zave, P. (1993a). Feature interaction and formal specification in telecommunications. Computer, 26 (8), 20-28.
- Zave, P. (1993b). Conjunction as composition. ACM Transactions on Software Engineering, 2 (4), 379-386.

**APPENDIX**  
**TRANSCRIPTS OF INTERVIEWS**

## A.1. Interview 1

POSC (Mobil)

January, 1995

D. R. - Mobil

### A.1.1. Interview 1 Notes

**WDB:** Most Fortune 500 companies are facing a competitive business environment that requires increased flexibility and reduced time-to-market. In response to these competitive pressures, most companies have recently undertaken a business process re-engineering (BPR) effort. These BPR efforts often reveal that information systems as currently designed and implemented represent a major barrier increasing flexibility and reducing time-to-market. Are these same issues of concern to Mobil and, if so, why?

**DR:** As an "oil company," Mobil's competitive advantage is based upon finding oil with more accuracy and therefore less cost than our competitors. With the cost of drilling a well in the ten million to hundred million dollar range, Mobil is always looking for ways to reduce or manage the risk associated with drilling. In E&P (Exploration and Production), we have always tried to managed that risk by using information. To generate information we collect technical data, obtained from a number of sources such as core samples or seismic surveys, and build geological models that use this technical data to predict where we can find oil. For our IS group, there are two issues in this whole E&P process. The first issue is that not only the amount of data but also the types of data that we gather are increasing dramatically - and this really creates two separate issues itself. The sheer volume of data is not necessarily an insurmountable problem by itself until you consider the processing power you must have to integrate and use all of this data - which is why increases in the types of data create such a big problem. Seismic data is actually real-time analog data that we digitize for convenience; whereas, core samples represent a number of static properties. The basic difference in the format of these different types of data make storage and integration of them a significant problem, because we can have a number of different types of data about the same geological location. So, our first challenge is to find a way to create a standard index that can be used for recording the existence of a large volume of different types of data and the locations of that data. The second issue we have to deal with is the integration

of these different formats on demand as we design and build new geological models. So to get back to your question, yes, Mobil's IS group is faced with reducing cycle-time and increasing flexibility. We must reduce the time to find and integrate different types of data while providing the flexibility to support designing and building new geological models as our petroscintists develop new ways to analyze things.

**WDB:** Does you IS systems currently support your business requirements, and if not, how is your IS department attempting to restructure your systems to address these issues?

**DR:** A study conducted by our the Technical Data Management Study (DATAMS) team of the type of problems encountered by our petroscintists identified serious problems in locating data, data quality, and data accessibility. As a solution to these problems, the DATAMS team proposed the development and implementation of a single strategy for managing and administering shared technical data throughout E&P. The DATAMS team developed a list of seven principles for data management:

- 1) **assets:** Technical data are corporate assets that can give E&P a competitive advantage. As assets, they should be re-used, shared, managed, and secured.
- 2) **access:** Technical data must be readily accessible, in a usable format to authorized individuals.
- 3) **ownership:** Shared data are owned by a specific business unit and formally supported.
- 4) **capture and validation:** Technical data are captured and validated at their point of generation or acquisition to achieve highest quality. Continuing responsibility for data quality lies with each individual user, working within established data guidelines.
- 5) **authorized versions:** Each occurrence of data has a defined authorized version.
- 6) **data definition:** Each shared technical data item has a unique definition.
- 7) **data independence:** Data are independent of applications.

As a result of the DATAMS team's recommendations, Mobil joined in partnership with five other petrochemical companies to form POSC (Petroleum Open Systems Corporation).

The mission of POSC is:

To develop and deliver an E&P industry-specific software integration platform that will be the interface for petrotechnical applications, database management systems, and workstations.

POSC's objectives are to establish E&P-specific open standards by developing:

an integrated E&P data model (that will include a meta-data facility),  
a common E&P-oriented graphical user interface,  
software offerings to facilitate E&P standards,  
and compliance test suites.

Mobil E&P plans to use the POSC deliverables, especially the integrated E&P data model.

WDB: POSC seem to be a very data centric architecture, what about process?

DR: The geological models developed by our petroscintists are considered to be of such competitive advantage that these models are proprietary. In actuality, the data is also considered proprietary, but the risk of drilling has become so large that companies often partner to share that risk. Before agreeing to partner, companies will often agree to share data they possess regarding a potential drilling site. In order to exchange this data, an industry-specific data model is required (i.e., POSC). In non-proprietary areas of the company, developing a common set of industry processes would result in reduced system development costs.

#### A.1.2. Interview 1 Write-Up

POSC is a very data centric architecture developed to support the rapid integration of data in new ways or under new views (a driving business need because new views help find oil). In database terminology, new relational database schemes must be rapidly developed to support the new views. To support the integration of new database schemes, POSC will employ a meta-model that captures all of the entities, relationship, and attributes of all geological data types to be used. The meta-model will

be produced in a top-down manner. By standardizing at a higher-level of abstraction (meta-level), POSC will provide flexibility at a lower level of abstraction (the scheme level). Standardization by itself may or may not be good—how and what you standardize is important.

## A.2. Interview 2

EDS

February, 1995

M. S. - EDS

J. N. - EDS Research

### A.2.1. Interview 2 Notes

MS: RightStep is a roadmap or plan for getting from where you are to where you need to be. RightStep starts with a determination of business objectives - or where the business needs to be (this step could include a BPR effort). Based upon these objectives, a map of the different logical views of the enterprise is developed. Using these views, the functionality and operability of individual components is defined and the relationships between components identified. Gap analysis is performed by comparing the desired enterprise structure to the legacy enterprise structure. Based upon identified gaps, migration strategies for infrastructure and applications are developed based upon established standards and methodologies. The last step is deployment. For EDS, deployment generally means wiring together all the applications (including the legacy systems, because most of our works is done on already operating businesses - very few green-field systems). No one step is any more or less important than another step - what is important is the overall result produced by the process and the speed within which we can delivery the solution. How quickly we need to deliver a new solution depends on the specific industry.

RightStep in methodology and technology independent - which means that we will undertake a project using any methodology and technology that a customer desires. As systems have moved from a mainframe to a client/server environment, the number of tools and environments that these tools must operate in just exploded. We now have clients (i.e., front end windows build using something like PowerBulider) running in any number of environments (OS2, Windows, DOS, UNIX on different pieces of hardware) that can talk across the network to a server again running in any number of environments that store both data and procedures (i.e., Remote Procedures accessed through Remote Procedure Calls) written in "C" or some other language. The number of interconnected software products that fit into this client/server approach is in

the hundreds (see ComputerWorld's Client/Server Infrastructure Roadmap) which means that the number of possible software combinations is in the ten of thousands (this is an  $N(N-1)/2$  problem). So, integration is a major concern for EDS from a technological stand point alone. RightStep must be able to bridge across different generations of technology. Our enterprise modeling efforts and our AI & OO group, both based in Detroit (GM CIM manufacturing), have the most experience with integration issues.

JN: EDS Enterprise Modeling Project (undertaken with General Motors) - The EDS /GM Enterprise Modeling effort represents the largest enterprise modeling effort EDS has ever undertaken. To our knowledge, this project is the largest Enterprise Modeling project ever undertaken anywhere. The project was based upon an IE (Information Engineering) approach that started with ER (entity-relationship) data modeling. To build a top-down IE enterprise model, you must specify all of the entities and all of the relationships within the enterprise. The first problem one encounters is the fact that an enterprise modeling effort within a major enterprise requires you to find thousands of entities and thousands of relationships. Even if one can identify and uniquely name all of these, the time required to complete a modeling effort of this size is years. During that time, the enterprise is changing which means that your enterprise model is trying to model a moving target - which really cannot be done. The second problem with a unified enterprise model is that it dictates or imposes a specific model on the whole enterprise. At any point in time, some individual unit within the enterprise will be developing a new way of operating their business. Invariably, this new way of operating will not be consistent with the current enterprise model. There is no one unified model that fits all the possible ways a business unit could operate and there is no way to anticipate all of these ways either. An effective enterprise model must allow for flexible extension. In analyzing why an specific enterprise model could not be extended to support a new way of operating a business unit, we discovered that the relationships between entities are what caused some of the problems.

JN: EDS Meta-Modeling Project - In response to the enterprise modeling project problems, EDS research initiated a meta-modeling project to try to understand how to capture meta-level information about the relationships between entities. Every relationship has the potential of being a N:M relationship to different types - but we generally choose to restrict the number of types for current business reasons. Over time, the business restrictions change and we must move from an 1:1 to an N:M set of relationships types. When you change the relationship type from 1:1 to 1:N and then N:M you get a dynamic cascade through the systems as other relationships are affected. The coordination issues that these cascading changes impose on the system are significant. As an example just think about what is happening in banking. Traditionally, banks



only offered customers two basic account types: an interest bearing saving account and fee-based checking account. Today, you can choose from any number of different types of different accounts. You can have an interest bearing checking account with the fees waived if you keep a minimum balance that is tied to your savings account for overdraft protection. Or you can have all of that with the overdraft protection provided as a line-of-credit or as a loan. Or just about any combination that you want. Now add to all this the idea that the bank can now have different types of customers. Each time a new type of account or customer is created, the rules that represent the relationship between the entities customer and account change. For each type of account and each type of customer, you have a different set of rules. Each specific view of the new account type and new customer type (i.e., marketing, accounting, etc.) will have a set of rules associated with it that the system must be able to integrate in a consistent manner. To build a system that is flexible or extendible you must be able to quickly integrate all of these issues in some consistent manner.

In this project, large grained objects were dynamically built by integrating small grained objects, that contained both the data and process, together in a federated manner (bottom-up). The idea was that the meta-modeling layer would be able to resolve conflicts between small grained objects, so that they could be integrated into a consistent whole, by capturing rules in a bottom-up manner (generalization). Unfortunately, we found that some rule conflicts could not be resolved in an automated fashion given the level of sophistication of our meta-level facility. We were never able to solve the integration problem.

#### A.2.2. Interview 2 Write-Up

Systems integration is a significant architectural problem for EDS's clients.

Systems integration requires that both data and processing rules be flexibly integrated into some consistent whole. Integration is one problem but the overall speed of the integration process is also critical.

EDS's experience with using a ubiquitous top-down standard (1 and only one mapping) to achieve integration shows that ubiquitous standardization can solve the integration problem but fails to provide other necessary capabilities required in today's

business environment. Specifically, the size, speed, and flexibility of a system built using a ubiquitous approach prove limiting in real world system.

EDS's attempts to achieve dynamic integration using a bottom-up (or federated) approach suffered from rule conflicts created as domain specific objects were integrated into larger domain objects (an integration issue not a speed issue). Even if these conflicts can be resolved, the number of mappings required under a pure federated approach is very large  $N ( N - 1 ) / 2$ . To be able to respond to a dynamic business environment the architecture's C3 structure must be responsive to both driving business needs and technological changes--the C3 structure must make the architecture evergreen.

### A.3. Interview 3

Anderson Consulting's EIA Architecture

March, 1995

H. R. - Anderson Consulting

#### A.3.1. Interview 3 Notes

**WDB:** Why did Anderson Consulting develop EIA and what problems does it solve for you or your clients?

**HR:** We see the trends towards user-driven systems, integration of the enterprise, and the complexity and fragmentation of technology continuing through the close of this decade. As our clients demand flexibility, cost efficiency, compressed time-to-market, and open systems, we believe that these challenges can be met through the use of a comprehensive enterprise architecture or blue print that must define the connections and relationships of information systems components that are used to deliver business results. The architecture should specify what we should build not how—which means that an architecture is not restricted to one implementation method or technology. Quite the contrary, an architecture should help identify and guide the selection of an appropriate methodology and technology that are necessary to deliver business results over different generations of technologies and methodologies.

**WDB:** How does your architecture, EIA (Enterprise Information Architecture), help you address the issues of integration, flexibility, and speed in delivering IS solutions?

**HR:** The value of EIA is that it starts with a business driven approach that requires that the business' goals, policies, culture, and environment be identified, integrated, and then linked to the architecture's technology goals. Since EIA is neither methodology-dependent nor technology-dependent, an EIA as defined for a specific enterprise can be designed to support any business needs. As goals, methods, and technology change, the defined EIA can be refined and improved through evolution. The key to the whole EIA approach is to define the linkages between architectural layers which causes technical decisions to be based upon identified business dependencies and relationships. By defining the

architecture in a layered manner, changes in business requirements or technology can be isolated within a layer and its linkages to adjacent layers. The layer by layer encapsulation allows us to make changes with less impact on the overall system. By using CASE tools to model each layer and mappings across layers, we can gain increase speed - IE would represent an example that includes a mapping from the business requirements layer to a data architecture layer.

EIA is composed of seven distinct, yet related layers. They are:

environment  
    business requirements  
        data architecture  
            application architecture  
                infrastructure  
                    system software  
                        hardware/network

HR: These layers are interdependent; that is, a change to one layer can affect and cause changes to any or all of the other layers. As a rule, a change in one layer will usually have the greatest impact on the layers that are next to it. Again, the key is to define the dependencies that exist between layers for a specific architectural design.

WDB: What would be some examples of these dependencies?

HR: In the environment layer, external factors such as government regulations, market competition, and customer expectations can have a major impact on business requirements. Compressed time-to-market would represent one competitive factor that many of our clients would consider a strategically significant issue today--a business requirements layer issue. In a manufacturing firm, compressed time-to-market almost always creates the need to tightly couple the CAD (Computer Aided Design) system with the CAM (Computer Aided Manufacturing) system to create a CIM (Computer Integrated Manufacturing) system. Coupling these systems requires a common data model and the appropriate choice of infrastructure--a data architecture layer issue. Without identifying these dependencies and selecting technology aimed at addressing these dependencies from a business perspective, the desired business results will never be delivered.

### A.3.2. Interview 3 Write-Up:

An architecture must be able to cope with change - both changing business needs and changing technology (it must be evergreen). The architecture must be able to integrate the business's goals, policies, and culture (everything) into a consistent whole and then map the linkages between these business drivers and technological solutions (a mapping across two different domains). The use of a layered model for the architecture (use of abstraction) allows for an isolation or encapsulation of "what" each layer must provide from "how" it is provided. The encapsulation is useful in dealing with change - because only the mapping across domains must change.

## A.4. Interview 4

Texas Instrument

April, 1995

J. M. - TI

T. D. - TI

J. B. - TI

### A.4.1. Interview 4 Notes

**WDB:** Your CIM (Computer Integrated Manufacturing) application framework represents the largest know OO (object-oriented) framework in use today. In your CIM application framework, you have focused on building a structure (which you call a framework) that defines one layer in TI's EIF into which plug-compatible reusable components can be plugged. Could you please describe the framework and why it works?

**JM:** Our OO framework provides a set of "standard services" that are defined through a set of interfaces that call standard methods - much like the ANSI C library. The standard service defines what the method must do - not how. These services represent the relationships between components in the framework. Each component provides a certain set of services. In any given activation of the framework, the specific set of services and the order in which the services are called can be different - but the basic set of standard services remains unchanged. Since the framework captures the standard set of services required for semiconductor manufacturing, we believe that the framework itself is reusable - not only the standard services provided by a component. The services and framework are reusable because they capture the invariant relationships between components. The framework and services are both reusable because they specify what must be done - not how.

**WDB:** So your standard services are define in a top-down manner but allowed to be instantiated by each vendor?

**JM:** Yes. By specifying the standard service's interfaces, we are specifying "what" the service must provide but not "how" the service is provided.

**WDB:** By defining a single standard set of services (i.e., your framework) for the manufacturing layer you have reduced the number of mappings from a possible  $N(N + 1)/2$  to  $N + 1$  mappings at each layer.

**JM:** Right.

**WDB:** In your framework you have a planner and a scheduler, how can you make sure that when you plug these two components into your OO framework that they interoperate in a consistent manner?

**JM:** Currently, we handle that issue of "application logic" by hand. To interoperate consistently, the planner and scheduler must both use the same queuing discipline. You can either push material through the fab or pull material through the fab - but whichever discipline you choose both the planner and scheduler must operate using the same discipline for the framework to operate effectively.

**WDB:** What do you mean by "application logic"?

**JM:** The rules and the order in which the rules fire.

**WDB:** You have no automated mechanism in the framework for checking "application logic" issues and the issue of "application logic" is not captured in the interface is it?

**JM:** No, "application logic" is not captured in the interface and we do not have any automated mechanism for checking it at this time.

**WDB:** What if the scheduler wanted to optimize through-put for the whole fab and the planner wanted to optimize the productivity of individual machines?

**JM:** That situation would lead to sever problems with the overall operation of the framework. Again, since we developed all of framework components ourselves, we handled this issue in the component design phase.

**WDB:** So you if you wanted to change the "application logic" of your framework you would need to select a different set of components?

**JM:** Yes and no. To change the queuing discipline of the scheduler and planner we would need to select a new set of components - but to change the work-flow within a queuing discipline we would not because in any given activation of the framework, the specific set of services and the order in which the services are called can be different

**WDB:** So what happens when you want to purchase a framework component from a outside vendor? How do you know if the component is plug compatible?

**JM:** By checking both the interfaces specifications and the "application logic." In the future, we hope that standards like the OMG's (Object Management Group) OMA (Object Management Architecture), which will include an number of facilities like their Interface Definition Language (IDL), will evolve to the point where any OMA compliant component will be plug compatible with any other OMA component. T. D. is one of TI's designated representatives to the OMG.

**WDB:** Since the "application logic" is not captured in the IDL, how or where else could you capture it?

**JM:** In our Enterprise Integration Framework (EIF) document which was written by T. D., we have proposed that application logic be extracted from applications and placed in a repository so we can establish the overall integrity of the of enterprise model and instantiate the rules into domain specific contexts in a consistent manner (working groups within the OMG are developing the requirements for these type of repositories). As part of the EIF, we have proposed building an virtual enterprise model that would include a virtual comprehensive planning and scheduling application. The virtual planning and scheduling application would be used to test the ability to selectively interchange components. One of our next projects will be looking at building a business rule framework.

**WDB -** So the purpose of building a comprehensive virtual enterprise model is to provide a top-down logical model (i.e., abstract model) of how the organization should be put together?

**JM:** I believe that a top-down enterprise model should be built to provide the coordination needed to successfully build the correct lower level frameworks. As we built the CIM application framework, we only worried about the framework from the fab's operating perspective. Because of this domain specific view, our efforts to scale the framework to the enterprise level (across domains in a bottom-up manner) have not been successful. Without a map of where you are going, its hard to get there.

**WDB:** So without knowing what the issues are that you have to address at the enterprise level, you might not build the lower level frameworks with that capability?

**JM:** Yes.



- WDB:** Your framework specification and documentation seem to be very process centric. Is there a reason that data is not as important as process in your framework?
- JM:** Because the framework is built to provide a set of "standard services" for a real time manufacturing environment, the framework is more process centric. In an real time manufacturing environment, the half-life of most process data is of a very short duration and ends up being passed between procedures as parameters.
- WDB:** I would like to know more about how your EIF (Enterprise Integration Framework) will promote rapid application development of user tailored system solutions.
- TD:** The basic idea behind the EIF is to isolate application logic, data, and user interface. By providing end users with repositories or reference libraries of reusable industry-specific models, end users will be able to combine generic "piece parts" with custom components to quickly construct a uniquely tailored application system. Conformance with the EIF implies a number of capabilities. Reuse of generic designs in one essential capability. The fastest application development generally occurs when applications are created from preexisting designs or building blocks. The key to reusable design is having an environment that makes relevant designs easy to find, easy to understand, easy to modify, easy to integrate with other components, and easy to check the integrity and consistency of modified/integrated components. Visual programming tools that remove complex user interface code from application logic allow application logic to be maintained, configured, and distributed independent from technical implementations. As database technology and visualization technology change over time, the impact of these changes will be isolated due to the separation of these components. This independence will also allow user interfaces to be optimized for unique work flow requirements domain by domain. As a consequence, visualization technology will have to provide tools that will be able to operate with radically different visual, syntactic, and semantic characteristics across domains.
- WDB:** This interoperation across domains implies that the EIF integrates all the tools and domains in some seamless fashion through a centralized repository. What EIF interoperation standards or tools exist today?
- TD:** No complete standard exists today but a number of groups are working on standards for interoperability like OMG's CORBA standard (Object Management Group Common Object Request Broker Architecture) or ECMA's PCTE (Portable Common Tool Environment) standard. As EIF compliant components are developed, implementation choices will be made in consideration of cost, performance, scalability, portability, training, quality,

compatibility, and availability factors relevant to the target installation. Not all EIF compliant tools will be created equal - which is why we are developing the framework in the first place. By supporting plug compatibility, the EIF allows the user to select individual components from a wide range of components based upon the users specific criteria. Only when a users decides that the propriety extension or even complete development of a specific component will provide a strategic advantage should the user use anything but a generic component.

**WDB:** A number of people believe that reusable components represent the cornerstone of the new plug-and-play systems architecture. Given your experience, what do you think are the fundamental requirements for a component to be reusable?

**JB:** I not sure I have an answer for you but I can tell you about my experiences. When we talk reuse, we generally also talk about empowering the end-users by allowing them to tailor or extend these plug-and-play systems as they see fit. When I look at our recent experiences in these areas, I worry that we (systems people) are just really moving the reuse problem from the IT shop out to the end-user. Look at our experiences with financial analysis using Lotus spreadsheets. Before Lotus, financial analysis was a centralized and controlled activity. A very small group of well trained people performed all of the analysis and knew exactly what types of financial analysis where being done for what parts of the organization. Then along comes Lotus - which I view as a tool built of reusable low-level financial functions. If we had centralized Lotus in the financial planning department, the limited number of financial analysis would have been able to track what spreadsheets they had already built in support of a specific type of analysis. Instead, we distributed Lotus throughout the organization. Now everybody started doing their own financial analysis - which resulted in duplication of effort everywhere which is completely contrary to our reuse goals. Even worse, a lot of these spreadsheets contained serious errors that lead to some unique problems. As the inconsistency between individual spreadsheets surfaced, the organization felt that more control or coordination between individual spreadsheet was necessary. To solve this problem, we produced a standardized set of Lotus templates that performed financial analyze (Lotus spreadsheets with menu driven selections) and then required each department to use them for their individual financial analyze. The problem with these templates was that we could not anticipate all of the specific financial conditions that one could encounter. For a specific set of condition, the assumptions used in developing a specific template where fine and within those assumptions the templates worked fine. When a factor outside the defined assumptions became important, the templates no longer applied. So end-users rejected the templates in favor of building their own worksheets.

**JB:** Selecting Lotus as the organization's standard spreadsheet did promote reuse at another level. See, when you talk about reuse you must realize that reuse can

occur at all levels in the organization and across all processes. Lotus provided end-users with a single consistent look and feel for all our financial analysis applications - which reuses end-users knowledge of how to either build or navigate a Lotus based application. Using Windows as a front-end on all of your applications would be another example of this idea.

WDB: Did you know why there were so many discrepancies between the individual spreadsheets developed by different end-users?

JB: From what we know, there are really three issues that contributed to the problem. The first issue has to do with the skill or experience level of the spreadsheet user. A number of individuals were using Lotus functions without truly understanding exactly how the financial function worked. The second issue has to do with the data. When we connected the P.C. to the network, a lot of people got access to databases and started using down-loaded data in their spreadsheets. We discovered that a lot of people did not really understand the meaning of the data that they were down-loading into their spreadsheets and therefore used the data in ways that were not appropriate to this analysis. The third problem we discovered is that most individuals building spreadsheets did not understand the assumptions and limitations of the types of financial models that they were using or how these assumptions and limitations were built into spreadsheets in general.

#### A.4.2. Interview 4 Write-Up

TI's CIM (Computer Integrated Manufacturing) application framework was built in response to TI's need to reduce systems development lead times to 90 days from 18-24 months (a driving business need). A layered architecture provides a number of useful capabilities. Layering by levels of abstraction can support/allow for within scope changes as you specialize layer by layer - the OO model. A top-down specification is required for interoperation across layers - but by specifying the requirements of the lower layer (i.e., the "what" top-down) a bottom-up instantiation of the "how" allows for integration with flexibility.

Standardization by layer reduces the mapping problem from  $N ( N + 1 ) / 2 )$  to  $( N + 1 )$ . Layering also provided for encapsulation by layer by splitting “what” the layer must do and “how” the requirement is accomplished. By standardization of the “what” not the “how” you allow individual’s vendors freedom in building solutions.

The standardization of the interfaces alone is not sufficient for specifying interoperability. The architecture must integrate data, process, and rules/policies into a consistent whole. The architecture must allow the integrated whole to be tailored/modified by end-users to fit their specific needs - which is why the OO layered approach. Allowing end-users to tailor/modify applications requires the architecture to provide for a formal control mechanism for checking consistency of the integration.

How and what you reuse is important—a strict standard provides reuse but without flexibility—we need reuse that allows extension—so we must reuse higher level abstractions.

## A.5. Interview 5

USAA

May, 1995

USAA IT Architecture Team: P. C., C. E., W. B., D. J.

### A.5.1. Interview 5 Notes

PC: The architecture (USAA's Target IT Architecture) is patterned after a building architecture. Specifically, USAA used a book titled "How Buildings Learn" as a template for building an IT architecture. Using the approach found in this book, USAA has identified six layers in its new IT architecture. These six layers are labeled:

- 1) site,
- 2) skin,
- 3) structure,
- 4) services,
- 5) space plan,
- 6) and stuff.

In USAA's architecture:

site - corresponds to the site of a physical building in an electronic sense. Site is comprised of USAA's external connectivity to the external electronic world. USAA has identified 10 service activities that require external connectivity.

skin - corresponds to the outer layer of a physical building. In USAA's architecture, skin provides an external IT appearance and provides receiving, queuing, and security. USAA has identified 6 service activities that the architecture must address in the skin layer.

structure - corresponds to the load-bearing components of a physical building. Structure is comprised of operating systems, process control frameworks, business object frameworks, and document frameworks. USAA believes that this layer must be the most stable portion of the IT architecture because it supports, surrounds, and holds together all of USAA's IT components.

services - correspond to the electrical, plumbing, heating and air conditioning, and similar services provided throughout a physical building. Electronic services exist to enable the sharing of IT components throughout the enterprise which supports reuse and economy. Services layer components include an integrated communication network, object communication middleware, and a set of composite services embodied in "output."

space plan - corresponds to the how the internal space in a building is divided into offices and hallways and assigned to various departments. In our IT architecture, the space plan corresponds to lines of business. Traditionally, these lines of business were supported by a fixed space plan built of rigid COBAL programs (or walls) that separated and hid the functionality of business process automation. Just as commercial building have minimized this layer, shifting to modular furniture and moveable partitions to enable responsive change, our IT architecture must also minimize this layer by removing business rules and business processes from business automation programs. Rules and processes that are removed are placed in the structure, services, or stuff layers in things like our business object frameworks or process control frameworks. With rules and processes removed from this layer, the space plan layer must ensure that messages, items, documents, phone calls, etc. get to the right area/person for proper handling. Our space plan provides us with the information necessary to route these electronic items. Routing does not determine detailed work flow, that functionality is imbedded in the stuff layer.

stuff - corresponds to everything in your office that is not walls or furniture. Stuff consists of all those things that are mobile and can be easily reconfigured to accommodate business process change and personal preferences. Most of the components in this layer have traditionally been imbedded in highly structured and business-area-specific programs. In our architecture, the various functional components of these traditional applications have to be extracted, modularized, standardized, and placed on the end users' electronic desktop so these objects can be easily accessed, modified, and executed by the business user. We must extract presentation, data, rules, and process as we are going to support IT components that can be placed/moved/configured/reconfigured by end-users.

The real goal of our architecture is to make everything stuff.

**WDB:** Can you tell me why USAA is developing a new IT architecture and what you expect it to do for you?

**PC:** The reason USAA is developing a new IT architecture is because the business processes and products that the systems group must support are going through dramatic changes of their own. In general, the change is from supporting a

predesigned or mass produced insurance product to supporting a tailored or integrated mix-and-match approach to an insurance product. Take an auto policy for example. In a standard auto policy today, the overall structure of the policy, including the types of coverage and the relationship between parties, is pre-defined. The only issue left for the customer and underwriting to decide is the amount of coverage and the cost at which USAA will provide it. If a customer does not want a specific type of coverage, then the customer is really asking for \$ 0.00 of coverage for that type of coverage. In the future, USAA believes that customers will want to design their own tailor made coverage from scratch. To support this mix-and-match approach, our IT architecture must be able to support the production of a new one-of-a-kind insurance contract on a customer by customer basis. The issue in producing a new contract is not being able to cut and paste the appropriate paragraphs into the contract with some super word processing program. The real issue is combining different type of insurance coverage into one contract in a consistent manner. Each type of insurance has it own set of rules and regulations mandated by outside regulatory bodies. On top of these regulations, you have specific company policies regarding underwriting that must be considered. Building an IT architecture that captures the business rules that would allow an underwriter to combine different types and levels of coverage in a consistent single contract is the real challenge. To support this mix-and-match approach, the IT architecture uses the idea of a "framework" - which captures the structural relationships between components within a specific composite component. An example would be our "Contract Business Framework" in the structure layer.

WDB: Section VIIa opens by stating that while USAA's new IT architecture "must support all the strategic business concepts expressed in Section V, the most stringent and pervasive requirements of the IT architecture are flexibility and economy" (USAA Target IT Architecture, Section VIIa, pg. 1). Why is flexibility so important?

DT: The flexibility requirement is necessary because we (USAA) believe that we cannot specify *a priori* all of the possible insurance contracts that our customers will want and therefore the best way to provide them. One of the basic principles of our architecture is that the business user (either the customer or the customer working with our representative) should decide what types of coverage and the amounts of these coverage they desire. Because of this requirement, the architecture should not force a specific work flow or coverage combination on a business user *a priori*. In order to provide this flexibility, the architecture must provide a way to delegate the authority that flexibility requires but still ensure that a consistent insurance contract is produced both from a legal and business perspective. Economy is also important because customers will not want to pay

any more than necessary for insurance. We hope to achieve economy through reuse.

**WDB:** Your architecture uses frameworks to define the structural relationships between the generic elements in things like a contract, a process, or an insurance policy. Are these frameworks sufficient to capture all the dimensions that this architecture must address?

**CE:** No. The biggest problem that this architecture faces is its inability to address the issue of culture. Culture is the glue that binds all of USAA together. It represents our shared values and USAA has a very well defined set of shared values. These shared values define the way USAA views the world, how we work together, how we solve problems, how we deal with conflict, and a whole host of other similar issues - how USAA would work with a customer to build an insurance contract. All IT systems exist within some organization. If the culture of the organization is not compatible with the system design, the organization will reject the system. The system must operate and enforce the shared values of the organization. The architectural problem is not only capturing and enforcing shared values but dealing with the fact that these shared values are changing - in large part enabled by new information technology. So the architecture must not only enforce the shared values but also enable the organization to change them when necessary. Since we view USAA as an adaptive learning organization, the IT architecture must be adaptive (or adaptable) to support the organization. Specifically, this means the IT architecture must support USAA's new business processes defined through our business process reengineering efforts. Our new business process driven look is in contrast to our traditional organizational structure that was based upon lines of business like life, property and causality, etc. The big problem we are having in building our new enterprise-wide IT architecture is getting people to focus on the common elements across the enterprise rather than the specific elements that make a specific insurance line unique. Because of this mind set, people in this organization have traditionally not even wanted to share data across lines of insurance. A lot of this mind set is based upon empire building, but some of it is justified. When you have shared data, who becomes responsible for its accuracy? You could say that "since nobody really owns the shared data, nobody is responsible for the data" or you could say that "since everyone owns the data, everyone is responsible for its accuracy and use." When you have shared data, everyone must also agree on its meaning and resolving that issue across line of insurance creates another set of problems.

**WDB:** In your new IT architecture, what role does the architect play?

**PC:** We see an IT architect playing the roles of a zoning board and a building inspector. The role of the zoning board is to set building codes as the building



inspector role is to ensure that the codes were followed during construction. An IT architect would define or set the IT policies regarding specific architectural components. Like specifying a specific object communication middleware standard. When components are built or purchased from outside vendors, the IT architect must ensure that connectivity to our prescribed standards is enforced.

**WDB:** The document (USAA Target IT Architecture) that you gave us is labeled "DRAFT." What parts or layers of your architecture exist today?

**PC:** Things like external connectivity, receiving/queuing/security, operating systems, network routing, and desktop presentation already exist. The parts that do not yet exist but are critical are the frameworks and the object communication middleware in the structure layer. These two parts represent the reusable components and the mechanism for combining reusable components together.

**WDB:** Do you know of anyone that is currently developing these reusable components or the object communication middleware?

**PC:** IBM is developing an object-oriented insurance architecture based upon reusable components. We are also working with Miles, Burke, and Associates in developing a set of reusable components for financial applications. The OMG (Object Management Group) is developing the CORBA (Common Object Request Broker Architecture) standard for object communication middleware. CORBA represents a communication standard that components developers will use when building components. When we buy a CORBA compliant component we know that they will be able to talk to other CORBA compliant components. Robert Shelton is another person that we are aware of who is trying to develop reusable components. Robert is the chair of OMG's BOMSIG (Business Object Management Special Interest Group).

#### A.5.2. Interview 5 Write-Up

The architecture must integrate data, process, rules/policies into a consistent whole in some dynamic manner--not one-size-fits-all approach. Dynamically responding requires that you have a lot of degrees of freedom in your solution approach--so try to maximize the number of degrees of freedom as you build the

overall structure. Dynamic integration allow for a customer driven approach to our business - a business driven approach. Dynamic integration requires that the authority to change things be delegated to end-users so changes can be made on a case by case basis - but the C3 component of the architecture must be able to capture, check, approve, and log these changes.

The architecture must integrate data, process, rules/policies by using the culture or shared values of the organization - therefore the must capture these shared values. The architecture must be able to deal with change by being adaptive--it must be evergreen. The architecture must not only support flexibility but must enforce integrity rules to ensure that the a change in one area impact another area is some adverse manner.

## A.6. Interview 6

CSC

June, 1995

J. S. - CSC

### A.6.1. Interview 6 Notes

**WDB:** PRISM (Partnership for Research in Information Systems Management) has developed an architectural approach that you believe helps a company solve the problems of rapidly changing technology and technologies that are incapable of communicating with each other. Please tell me about this effort and how you think your specific architecture can help solve these problems.

**JS:** We believe that the underlying problem of architecture is that there is no context for making decisions about technology and its use from a business perspective. Very few of the organizations that we have seen have any evaluative criteria or structures in place for making architectural decisions. In trying to identify a comprehensive structure for architectural decisions, we identified four different types of architectures, and four different domains in which each type must be applied. The most important type is the principles on which the architecture is based. These principles embody the organization's philosophy of information systems, and its objectives for technology and its management. The other types are inventory -- a simple structured listing of architectural elements; models or diagrams of the desired architectural state; and standards for selecting and using architectural elements. The elements to which architecture is applied are infrastructure, applications, data, and the organization that supports it all.

A complete architecture involves an approach to all combinations of domains and types. Some cells will be more important than others depending on the organization. However, principles are the most important aspect of architecture and drive all other types. They must be based on the technological values of the organization in such areas as user autonomy, degree of risk acceptance, and the overall role of information and information systems in the organization. Principles should be relatively few in number, specific enough to drive behavior, distinctive to the organization, and articulated rather than invented. They should be stated by senior management and the role of IS in this process is to assist in their articulation. Without a clear set of driving principles, an architecture lacks a clear context for making decisions about technology and its use from a business perspective.

WDB: What is the source of an organization's technology principles?

JS: The main determinant of architectural principles is the organization's technology values, which are a set of underlying attitudes and perspectives that shape the organization's fundamental approach to information systems. Values cut across technology domains, and are even more long-lived than principles. Technology values exist in all organizations, whether or not they are articulated as such. The key to successful elucidation of principles is the discovery of these values, as opposed to their invention. All technology principles can be constructed from three primary areas value:

Orientation to risk: from aversion to tolerance

User autonomy: from low to high

Technology perspective: from cost displacement to strategic tool

However, values are not the sole determinant of principles. The organization's business situation and condition, its market position, competitive environment, and other elements of business strategy all impact technology values and principles directly, and in turn therefore underlie the entire architectural endeavor. A principle might specify, for example, that emphasis be placed on applications which increase the organization strategic advantage. If short-term business conditions force a reduction in system spending, systems with a strategic orientation might be cut first as opposed to cost displacement systems. Architectural principles must also be informed by, but are not determined by, an assessment of the current state and future direction of technology.

WDB: Why do you think most architectural efforts have failed to solve the problems of rapidly changing technology and technologies that are incapable of communicating with each other?

JS: With the advent of powerful, affordable, and interconnected distributed computing technologies coupled with the rapidly changing business environment, the traditional technical and economic assumptions that guided the actions of IS departments in the past are no longer valid. The resulting uncertainty has created confusion and anxiety, and left organizations without a way to evaluate choices in a straightforward and analytical way. The fundamental problem is that we lack evaluative criteria and decision-making mechanisms which can guide the development of systems and the choice of technologies.

Traditional evaluation techniques fail for a number of reasons:

applications and their requirements are not yet know,

future vendor offerings and developments are unknown,  
evaluative criteria are unclear and undefined,  
and often the alternatives being evaluated are sufficiently different in nature as  
to be impossible to compare.

Architecture is not a formal solution to these problems, but rather a mechanism  
for identifying and resolving conflicts and building consensus on technology  
directions and strategies. What is needed is a set of long-term, consensual  
criteria for evaluating and managing information technologies that will persist in  
the face of change.

**WDB:** What are the types of decisions that an architecture must address?

**JS:** In our research, we have identified a number of issues that the architecture must  
address in each of the domains.

**Cross and inter-domain**  
cost sensitivity  
time horizons  
degree and extent of standardization  
simplicity/complexity  
generality/optimalty  
sharing  
organizing theme

**Infrastructure**  
extent of access  
vendor stance  
degree of interconnection

**Data**  
ownership  
responsibility or stewardship  
location  
access

**Application**  
location of processing  
extent of interfacing and integration demanded  
responsibility for development and maintenance

**Organization**  
focus of system responsibility  
definition of roles

### career paths

**WDB:** What are the functional requirements of an architecture given today's rapidly changing technology and business environment?

**JS:** To be successful, an architecture should be able to:  
avoid "re-inventing the wheel" by capturing the general principles of technology decisions and management,  
over time, move the technology toward increased compatibility, interconnection, and integration, where appropriate,  
enable growth, extension, and enhancement of the existing installed base,  
create a consistent and coherent development environment,  
and establish basic guidelines for conducting IS business.

#### A.6.2. Interview 6 Write-Up

The choices that the organization makes when building an IS architecture must be driven by the organizations shared values or culture as well as the organization's market position, competitive environment, and other elements of its business strategy. PRISM represents a formal approach for identifying these values.

## A.7. Interview 7

Apple

June, 1995

L. S. - Apple

### A.7.1. Interview 7 Notes

**WDB:** What is VITAL and what problems do you think that it will solve for Apple?

**LS:** VITAL is our plan or blueprint that not only describes the various components of an information system but also details how they all come together to achieve the final objective. The first step of the VITAL taskforce was to define a set of fundamental design points or architectural principles. After reviewing the literature on business strategy and architectural frameworks, the task force reached the following conclusions:

Flagship systems inherently shared much common data and contain substantial dependencies for both information and process.

Current systems cannot be integrated nor can they be economically modified to share information in a consistent manner.

The opportunity exists to identify and incorporate commonalities and dependencies. However, the new architectural effort cannot impede progress on current projects due to their urgency.

Therefore, a common denominator is needed among all developmental and data center groups to provide a common set of design points, measurable principles, and basic structures that enables sharability and integratability of new systems.

Based upon these initial conclusions, the taskforce resolved to define an architecture to enable individually-constructed systems to be sharable, integratable, independent of any specific vendor platform, and to leverage work done by one project across many future developments.

**WDB:** How does VITAL identify and capture these common denominators?

LS: You need to understand that VITAL architecture is really a technical architecture - which is only one of four architectural layers that the taskforce identified. The top layer starts with a business architecture that defines how you organize your business. The principles defined at this level drive the systems architecture which defines how you synchronize your systems. Again the principles at this level drive the technical architecture which defines how you build your applications. The last layer is the product architecture which defines what technologies you apply. So VITAL defines a common set of functionality at the application level.

WDB: What architectural principles did the taskforce define at the business level?

LS: VITAL defines a single business architectural principle that basing systems on business function rather than on organizations provides a stabilizing effect on information systems, reduces redundancy, and supplies the flexibility for organizational change.

WDB: So you start by defining a basic set of business functions or business processes that exist at the business level and then drive that down through the other layers?

LS: No. Traditionally, functionality was defined in a "top-down" sequential approach. Today, the dynamic business environment precludes us from using a methodical, step-wise regression through these layers. We cannot "freeze" the business rules and business structure long enough to document them, much less validate them across business boundaries, before we must react to new environmental and organizational challenges.

Since the business architecture must be able to be adapted to new environments, a complete definition of its functionality cannot be defined *a priori*. Given the lack of a complete "Business Architecture," the question becomes how can the technical architecture be designed?

The VITAL taskforce feels that the answer is to view the technical architecture from the perspective of the knowledge worker. We began with the concept of empowering the individual - which placed the individual at the beginning of the architectural design process rather than an abstract concept called "the business." Using this perspective, we defined the requirement of the technical architecture from both the individuals and organizations perspective.

We feel that the individual wants: maximum access to information, with a minimum of intervention from IS, timely access to accurate information across organizational and job boundaries, a single system image on the desktop, regardless of how many individual systems are integrated/accessed, applications



that behave consistently across multi-vendor software and hardware, and interoperability among software packages on the desktop.

We feel that the organization wants: consistently applied business rules and assured security, maximum information sharability with minimal cost, ability to achieve "virtual" systems integration across dissimilar hardware and software platforms and applications, and the ability to handle dynamic growth and change.

WDB: How does VITAL achieve all of this?

LS: The general concepts adopted by the VITAL taskforce for achieving these goals where to:  
design information systems from the desktop perspective,  
construct applications from standardized components,  
use standardized interfaces to reduce life cycle cost and leverage prior efforts,  
and gain maximum flexibility in meeting business needs by reusable,  
generalized modular components that are hardware independent.

Specifically, VITAL is built using a data warehouse that not only captures data at its lowest or atomic level but also a meta-data facility for insuring the accuracy, integrity, and compatibility of integrated solutions. The actual integration takes place in the Desktop Integration component of the VITAL architecture. Desktop Integration is itself made up of four components: Information Navigator, Information Synthesis, Help and Assistance, and Information Server Access.

WDB: So VITAL is really a client-server data warehouse based architecture in the sense that the integration and sharing that takes place through VITAL is data across processes not shared process?

VITAL is a data centric architecture that does not specifically address shared process.

#### A.7.2. Interview 7 Write-Up

The term VITAL is formed from the first three letters of the platforms the architecture supports (VAX, IBM, Tandem), as well identifying the charter of the product (Architecture) and its intended area of impact (development Lifecycle). In

addition to the architectures objective, the VITAL architecture effort included DEC's DSTAR head architect.

The developers of VITAL understand that data, process, and rules are required to build an information system. However, the rules and processing are the elements in a new system implementations that must change. Therefore, VITAL is a data centric architecture that attempts to integrate data only - through a data warehouse - leaving the process and rules dimensions wide open or with the maximum number of degrees of freedom.

## A.8. Interview 8

Microsoft

June, 1995

K. G. - Microsoft

### A.8.1. Interview 8 Notes

**WDB:** Could you please explain what MSF (Microsoft Solution Frameworks) is and what it does?

**KG:** We define MSF as a services based architectural approach which is loosely referred to as "client-server" by others. This approach is characterized by the formation of industry/business partnerships to deliver building-block components of business applications solutions. Specifically, a services based approach is characterized by:

distributed data,  
distributed processing,  
interactive, GUI front-ends,  
and heterogeneous platforms, tools, and solution components.

We see the client-server approach as the new model for enterprise business systems. As business migrates toward this new model, there are a number of complex issues that must be addressed.

These issues include:

strategic business planning and, possibly, re-engineering,  
organizational infrastructure issues regarding how to roll out and operate distributed, heterogeneous systems and how to support them,  
enterprise information modeling, to ensure that data as a corporate resource is managed and shared effectively across applications,  
and applications development, from the systems development life cycle to infrastructure considerations surrounding reusability and interoperability,  
cooperative development, quality, and on-going improvement.

MSF provides a strategic planning framework for helping customers development guidelines, methods, techniques, and tools to assist in this migration process. It delivers an applications development discipline for client-server applications. MSF reflects a 'toolkit' or 'template' approach rather than a methodology. The key to the leveraging MSF is to select the tools, methods, and techniques that best fit an organization's business strategy, technology decisions, skill sets, and support structures.

**WDB:** What are the functional requirements or capabilities that MSF, or any similar architecture, must possess to provide for the migration to client-server based enterprise business systems?

**LS:** To provide a successful framework for migrating from legacy systems, an architecture must possess:

extendibility - which means that the architecture must be able to be extended to address new elements as they become important to the business,  
tailorability - which means that the architecture must be able to be tailored to the organization's environment in terms of its methodologies, skill set, goals, and technologies,  
and customizability - which means that the architecture must be able to be customized to specific end-users perspectives.

**WDB:** How do you intend to provide this functionality? or How do you intend to implement client-serve based enterprise business systems?

From a business systems perspective we believe that you must:

move user interface and analysis services to the users while still retaining control over corporate resources like data,  
make it possible to rapidly develop a range of end-user applications that can be customized to specific user profiles, by decoupling the user services front-end from business processing and data,  
encapsulate business objects for better control over application of business rules to information access and decision making,  
encapsulate business and application services to facilitate process distribution and replication, component reusability, and implementation independence,  
and preserve legacy systems in the migration process.

**WDB:** How do you achieve these objectives from a technology perspective?

**LS:** We think that there are a number of technological dimensions that are evolving simultaneously that are required to support client-server based enterprise business systems.

These would include:

evolving industry standards for achieving interoperability and connectivity between clients and servers (such as ODBC),  
increased use of the object-oriented paradigm for encapsulating services, allowing applications to be built from a collection of reusable components from a variety of sources,  
technologies that take advantage of applications-as-services (like OLE),  
movement of common application services into the operating system as core shared services (like MS Windows NT),  
and increasing support for distributed processing and data (such as Remote Stored Procedures in SQL Server).

#### A.8.2. Interview 8 Write-Up

MSF represents an example of the current three tier client-server architecture (it separates data, process, and presentation). MSF uses the OO paradigm to provide encapsulation (rules and processes are encapsulated within objects), reuse, and extendibility.

## A.9. Interview 9

Open Engineering

July, 1995

R. S. - Open Engineering

### A.9.1. Interview 9 Notes

**WDB:** You believe that a business object architecture is the only way to provide the leverage, flexibility, and reduced time-to-market required in today's business environment. What features of a business object architecture make it uniquely capable of providing the leverage, flexibility, and reduced cycle-time?

**RS:** The difference between a business object architecture and previous approaches to business modeling and systems design is that under previous approaches the business models were static blueprints of business data and business function. These static models were primarily useful for designing corporate-wide shared databases. Using methods like Information Engineering (IE), their construction was a long, involved process that took place in parallel with applications development and ongoing business change. To have a significant impact, most or all of the existing systems inventory had to be rewritten to use the "corporate database." After a two to three year business modeling effort, the added five-plus years required to fully re-implement all business systems left several fundamental problems: (1) the business usually changed enough during this time to obsolete the original work, (2) businesses could afford neither the time nor the money for this process, (3) and nothing in the approach or the underlying technology made business or system change any easier to manage - we really had a one-size-fits-all approach.

A business object architecture provides the ability to create business models out of active representation of the business world, not just static data entities. Instead of limiting business to sharing data only, a business object architecture allows sharing of process, policy, and data. By defining the components of a business object architecture using the concepts that business people work with every day, the business object architecture is able to focus on the operation of the business instead of a set of separate data and program modules.

Let's take "customer" as an example. In an IE model, one would include a corporate data entity for customer. On a separate hierarchical functional decomposition model, one would find the business functions that use customer data. One model would provide the "data picture" and the other an abstract and arbitrary decomposition of what functions comprise the business. Nowhere would a complete picture of a customer be developed. In a business object architecture, a single component exists that represents all aspects of the business concept "customer": what we know about it, what it does in the business context, the rules that constrain its behavior, and the interactions and relationships that it forms with other business objects.

Because the components of a business object architecture are active players, a business object architecture provides a dynamic model of the business. Because these components are designed to expose "what" services are provided and hide or encapsulate "how" the service is implemented, components in a business object architecture can be built and modified incrementally - which allows them to be used to integrate and migrate parts of legacy systems and data structures by wrapping existing legacy implementations. IE models typically cannot be approached in piece-parts because representations of the business components are split across separate modeling paradigms - network data models and hierarchical function models.

By raising the level of abstraction to "what" services are provided instead of "how" the services are implemented, components can be designed that are re-used providing leverage, compactness of design, and economies of scale. Consider an example business object like "order." Sales people call them "customer orders." Procurement staff know them as "purchase orders." When defective merchandise is returned, the receiving staff expect a "return materials authorization." All of them are actually orders. Only the two parties playing the role of buyer and seller change. For a customer order, the customer is the buyer and the company is the seller. For a purchase order, the company is the buyer and the vendor is the seller. In each case, the basic behavior of an order is the same. Only minor differences exist for each specific type of order. By providing a mechanism for abstracting and capturing the common behavior along with the ability to tailor this behavior to a specific type of order, object components can leverage the common business patterns found in every business.

**WDB:** What is the difference between a business object architecture and object oriented systems development?

**RS:** The major difference is that a business object architecture is driven by business factors. The business object architecture must define the business objects, the business context in which they exist, the rules that constrain their behavior, and the interactions and relationships between business objects from a business

perspective. Object oriented technology is a key enabler for building systems that require encapsulation of implementation and selective refinement of generic behavior (i.e., polymorphism), but object oriented development by itself does not automatically provide you with plug compatible components. We consult with a number of large organizations that have already "gone objects" in search of such immortal values as reuse, flexibility, reduced cycle-time, and lower costs. Many of these organizations have ended up with immense stockpiles of custom-built code-level objects - which represent brand-new stovepipe systems each built out of different objects by programmers that re-defined fundamental business concepts within their stove-pipe perspective. What these organization have done is develop systems with object technology without first thinking about objects and how they should be used. It is the intelligent application of technology that solves problems, not the technology itself.

A business object architecture allows us to focus our thinking on composing business system solutions from collections of interacting objects rather than on decomposing functionality into stovepipes. The architecture allows us a way to package the policy and controls which determine the behavior of the composite objects. In the client-server paradigm, the architecture provides the basis for a middle "business logic" layer so the business logic is not embedded in the presentation layer or the back-end database (as required by two-tier tools such as PowerBuilder or Visual Basic). Without a "business logic" middle layer, each individual developer can build front-end applications that re-define fundamental business concepts within their stove-pipe perspective.

**WDB:** Since you believe that object oriented technology is only an enabler of a business object architecture what are the functional requirements of an this architecture independent of technology?

**RS:** The functional requirements of the architecture are that it:

be driven by the business vision,  
capture the process, policy, and data for each business object consistently integrated one place,  
provide for abstraction and instantiation of domain concepts (like the order example),  
provide for piece-wise or incremental modification in order to support migration from legacy and changing infrastructure technology,  
and that it be focused on solution by composition rather than decomposition which means that it must define the interactions and relationships between business objects.



### A.9.2. Interview 9 Write-Up

An architecture must integrate data, process, and policy - which is why an OO based architectures that provides integration by requiring data, process, and rules to be combined in one place (i.e., inside an object) is useful. The architecture must allow us to separate data, logic (i.e., rule, policy, processing), and presentation into distinct dimensions so changes in one will not affect the others. The architecture must also make use of abstraction because abstraction provides for reuse and also provides for extendibility as more general relationships are specilized. The architecture must be business driven.

## A.10. Interview 10

OMG/X3H7

October, 1995

C. C. - DataAccess

J. S. - VMark

H. K. - AT&T

F. C. - EDS

O. S. - IBM

G. H. - SEMATECH

F. M. - GTE

### A.10.1. Interview 10 Notes

**WDB:** Why did SEMATECH develop the CIM application framework?

**GH:** SEMATECH became involved in developing the CIM application framework because our member companies were experiencing a number of problems managing their CIM systems using the current software lifecycle practices.

Specifically, our members identified a number of issues that included:

rising development costs,  
a lack of interoperability and distributed integration,  
limited extensibility,  
data inconsistency and redundancy,  
difficulty in changing and maintaining complex, monolithic legacy systems,  
and a scarcity of global models for software engineering, maintenance, support,  
and operations.

SEMATECH's analysis determined that all of these issues could be traced back to our members companies usage of traditional lifecycle software development

practices that result in the deployment of monolithic legacy systems. We felt that in order to successfully address all of these issues simultaneously, a new software development lifecycle would need to be defined and implemented. Since SEMATECH is not in the business of building software, SEMATECH felt that its role in this effort should be to start by developing a new lifecycle architecture. The new lifecycle architecture is founded on a number of principles:

An architecture is a long-term roadmap: the effort needed to develop an architecture can only be justified if this investment reduces development costs for many projects. Therefore, an architecture must cover the long term by taking into account future use of emerging technologies.

This requires two things:

that the architecture be inherently flexible, and that a mechanism be put in place to revise it periodically as technology evolves.

One architecture: in order to integrate the enterprise, a company must be able to interconnect a broad range of systems. Systems that are interconnected must share one overall architecture.

Develop specifications, not products: architectures that define sets of detailed standards (i.e., TCP/IP or SNA) become obsolete and cannot evolve. If an architecture is properly designed, multiple products can coexist within the architecture as long as they meet the same architectural specification.

Understand, Simplify, and Automate: a successful architecture should maximize the systems' responsiveness to business needs. It should not be purely technology-driven. On the other hand, it should not consist of blindly automating the current manual procedures. Therefore, a business must first understand the problem by determining what issues are really symptoms and which are really causes by surfacing underlying assumptions. Given a complete understanding of the problem, a business must simplify by elimination of non-value-adding steps. In the simplify phase, the business must determine what organizational changes can be effected to improve the business process, regardless of whether this is a manual or automated process. In the automate phase, the organizational changes identified in the simplify phase are evaluated to see which changes should be automated.

Systems = Development + Execution: this principle says that the architecture should allow, but not require, the development and execution environments to use the same technologies.

**Adopt applicable standards:** obedience to applicable standards allows multiple applications to interoperate.

**Include legacy applications through evolution not revolution:** in order to be accepted, most architectures must include support for migration from existing systems, rather than having to throw them away completely to introduce new capabilities.

**Work through partnerships and consortia:** the architecture will be more successful, and the risk inherent in any change will be lower, if the architecture is adopted and pursued in concert by all elements of the extended enterprise. By developing a consensus on the definition of what areas are open to cooperation and those that must remain closed in each organization's competitive domain, a better understanding of the requirements of your architecture will evolve.

**Empower the users - aim for model driven systems:** a successful architecture must reduce the gap between the users' understanding of their activity and the way in which applications that support or automate this activity are described. Model driven computing with feedback is the key to empowering end users. End users should work directly with models that reflect the business domain not the artifacts of the models of the business domain translated into some technological model.

**WDB:** Based upon your experience with the CIM application framework are any of these principles any more important than any of the others?

**GH:** Not really, they are all important to success, but we did identify four critical success factors in developing the framework.

**reuse:** is often falsely seen as dealing principally with code. In reality, software specification and designs must become reusable if reduced cycle-time and reduced costs are going to be achieved.

**tractability:** both in terms of technological decisions made based upon driving business needs and also relationships that are established between artifacts produced because of different methodologies or technologies being integrated under the architecture. Systems people are always failing to capture analysis and design assumptions and when integrated systems evolve over time the missing knowledge always proves troublesome at best.

process integration: we must consider the entire software development process to decide how to best execute each task, instead of selecting each component independently. Integration must address the compatibility between everything.

continuous improvement: both business needs and technology are going to evolve over time. If the architecture cannot address this evolution through rapid, incremental continuous improvement, systems built using the architecture will not support the business.

**WDB:** The Object Management Group's (OMG) stated mission is to establish an architecture, based on available technology, to enable distributed integrated applications - or what most of us call plug-and-play. What does the OMG's architecture look like and what are the underlying principles that it is built upon that are going to provide us with plug-and-play?

**CC:** The Object Management Architecture (OMA) consists of three groups of heterogeneous objects interconnected by a Object Request Broker (ORB). The three groups are CORBA services (Common Object Request Broker Architecture services), CORBA facilities, and application objects. CORBA services are defined by OMG as low-level application support functions. OMG has a specified set of fifteen services that include services like: naming, persistence, event notification, concurrency control, and access control. CORBA facilities are defined as higher-level object services that reside between CORBA services and application objects. CORBA facilities include user interface facilities, information management facilities, systems management facilities, and task management facilities. Application objects are coarse grained objects that exist in the execution environment. The ORB provides location-independent connections between objects.

The basic idea behind the OMA is to provide an architecture that allows us to build systems from plug-compatible components. Specifically, interoperable, reusable plug-compatible components. Increased flexibility, reduced cycle-time, and lower costs are all dependent on reusability. To support plug-compatible reuse, a business component must be encapsulated in two directions. The external world must not know anything about component internals, and the internals must not know anything about external components, other than allowing interested components to register for notification of specific events or exception conditions. The principles behind the OMA are:

interoperability: application components developed independently, with different tools, operating systems, standards, and requirements should be able to interoperate at a business semantics level without undue requirements for integration of technology.

flexibility, adaptability, and extensibility: applications should be able to be produced and changed rapidly so as to reflect changing business requirements. Existing components, whether internally or externally developed, should be able to be integrated into existing systems.

scalability: application components should be adaptable to different engineering tradeoffs so as to allow the information system to scale without changing the business or application model.

and reusability.

The OMA is an open standard, which means that it specifies the basic or minimum set of services that are required to be plug-compatible. Interfaces and capabilities can be added as required based upon the business's needs.

**WDB:** Why is encapsulation so important to reuse?

**CC:** Encapsulation allows the developer to limit the effects of implementation issues to the internal world of the object. By encapsulating how methods are implemented, objects in the external world do not have to know anything about the internal world of an object with which they wish to communicate. The only requirement for communication is the specification of a well defined communication interface between objects. The OMG provides this communication interface through IDL (the OMG's - Interface Definition Language).

**WDB:** You said that the OMG's architecture is built upon three groups of object. The third group that you mentioned was application objects. What is an "application" or "application object" in the OMG world?

**CC:** In the OMG, "applications" are defined as collections of interworking/cooperating objects that together provides some user functionality - as opposed to the idea of a traditional monolithic procedural application. This idea of an ensemble of collaborating object is a very dynamic notion. In the new world of distributed objects, objects can be shared between applications and which objects actually participate in an application can and may be determined at runtime.

To build an application, application objects must connect a set of service points (i.e., OMG interfaces) together to provide some user functionality. The OMG defines these service points but not "how" they are used to build any specific application - and this is where the OMG's and BOMSIG's (the OMG's Business Object Modeling Special Interest Group) views differ significantly. The OMG takes a service-up view (based upon their defined services) whereas BOMSIG

takes a model-down view (based upon their application centric view of the world) and the interplay between these two views is not yet clear.

Some people, like SEMATECH, are trying to build frameworks - reusable patterns or templates - that provide a structure into which the service points can be plugged into. The idea behind a framework is that the framework calls the users code which is the exact inverse of traditional systems where the user calls the system code. Perhaps these frameworks represent the transition between the services-up vs. model down-down. In the case of OO frameworks, a framework represents a set of classes - with generic behavior - that the developer is expected to subclass in order to provide specific implementations - hence the idea of the system calling the users code which is programming by difference and should be only 20 % or so of any given application.

JS: I strongly agree with the OMG's principles of flexibility, adaptability, and extensibility as well as the idea of reuse but think that subclassing framework classes to build an implementation will result in a situation where your application will break when you upgrade the framework. Because you cannot transparently upgrade a subclassed framework, I don't believe that inheritance is the appropriate reuse mechanism in this case. I think that delegation should be the reuse mechanism of choice and that we should be talking about ensembles of interaction objects built using delegation rather than the idea of frameworks built using inheritance.

HK: As systems people, it is the collective behavior or composition of a set of objects that is of interest. In order to quickly build new compositions or revise old ones, the business rules that define the composition's behavior must be able to be respecified and applied to the composite set of objects. If we embed Business rule in methods inside objects, then we will bind the business rules to the object in such a manner that we preclude respecification of the current binding when we need to create new compositions.

FM: In our experience at GTE, a software architecture built upon the idea of inheritance of methods or business rules creates a large number of unanswered problems when one considers the issue of legacy system integration. None of our legacy systems are OO based implementations. However, I think that there are really two issues here that need to be distinguished. We need to distinguish between separation of re-usable "stuff" like Business rules and the implementation of the separation and necessary re-combination.

Separating rules from business objects is really a design issue - as in design for reuse methodologies - and this should make sense independently of how it is implemented. The ideas of refication and reflection are mechanisms that can be used to combine separate re-usable components in a variety of contexts. We

believe that reflection could be used in the underlying object model to provide such inclusion of rules in legacy components or any components that may not have been designed to reference such rules, but should.

CC: That distinction is very important. Inheritance, delegation, rule binding and such are all implementation issues. From the point of view of the business user, business objects are objects that must respond appropriately if ANY business rule is broken ANYWHERE as a result of ANY action on a specific business object. So from the point of view of a business user of an object, the rules are absolutely encapsulated by the object. From a development point of view, is it far better that each rule be expressed exactly once, which implies a system of collaborating objects rather than having each and every object have the rule embedded within it.

Likewise, subclassing - not subtyping - is an implementation issue. I think that there is little doubt that an order is a subtype of some base business object. The implementation system may or may not use subclassing to create the object - the order could be in RPG. I am quite comfortable with the idea of building a framework of subtypes to represent business objects. From an implementation perspective, I can live with or without subclassing. Again, I am not sure how we bridge between the model-down view that would use types and the services-up view of the OMG that uses classes.

FC: While I agree that it is probably useful to centralize the behavior of rule processing either through inheritance or delegation, objects should be reflexive with regards to rules - that is - I should be able to ask an object about which rules it supports, tell an object to add a new rule, tell an object to change an existing rule, etc. My experience has taught me that you should always challenge a design which centralizes some behavioral aspect because centralized behavior and states represents a design pattern which I've found carries much undesirable maintenance baggage itself.

CC: Another very important distinction that must be made clear in the architecture is the relationship between what will be modeled and/or built and any architectural infrastructure that will execute/or-be-used-by what is built. Again, this represents the bridging issue. Just as application objects must connect a set of service points (i.e., OMG interfaces) together to provide some user functionality, Business Processes will be needed to connect together Business Objects to achieve some behavior. While not entirely different from a single service of a business object delegating responsibility, the idea of a business process is so important to a business that we must provide business users a way to look at the system from this perspective.



From the business users perspective, it is meaningful to focus on process as the driver of the business and business objects as the driven portion of the business. From a programming perspective, processes have state and multiple instantiations; therefore, they are objects - not simply a method on some object - since methods do not have state. I really do think that there is a difference between a customer or order and the product development or order fulfillment processes.

In one sense, processes (like order fulfillment) span objects and are not really owned by one particular object - which means that business behavior (a set of operations) does not have to belong to a specific object in most cases - which gets us back to the idea of collective behavior. If several objects interact then this interaction is jointly owned by the participating objects. Therefore, an architecture must have some mechanism for combining objects into ensembles of well behaved interaction objects - whether this mechanism is centralized or decentralized.

FC: I think that C. C. raises a very good point regarding collective behaviors - but there is another issue that I feel we really have not addressed yet which is multiple views. Not only will we have multiple objects interacting as an ensemble but we must also realize that each ensemble's as well as each object's behavior will represent an integration of a number of different views all of which must be consistently and simultaneously supported or enforced.

In our experience with legacy systems, we have found that a change in just one of these views or perspectives can invalidate the overall system's design - I mean just consider the impact that changes from external regulatory agencies have had on our business systems. So I think that we must figure out how to built-in a capability for integrating change into a system's overall design.

WDB: Isn't this idea of constant regeneration of the system really what the use of CASE tools is all about? I mean, in theory, we could have one really big model of the organization in some CASE tool and every time we needed to change some aspect of the system we could just regenerate the whole system?

FC: I am not sure how to answer that question. In theory, maybe. If in theory you are correct, then what the current software crisis means is that we just haven't built big enough CASE tools yet - and I don't think that bigger CASE tools alone will solve the problem. At least in our experience with Information Engineering (IE) based approaches, project size is only one aspect of problem. Project complexity and IE's top-down (unified) approach to systems design often prove to be equally troubling aspects of the problem. Even in a completely static environment, there are just too many entities, relationships, and attributes in any large organization to get a single agreed-upon name and

meaning for all possible users and in a dynamic environment these problems are just compounded.

Of course the OO world believes that we can encapsulate things and therefore limit the effects of changes - which brings us back to this issue of embedding business rule in objects.

OS: I would like to comment on this idea of constant regeneration of the whole system. I do not think that constant regeneration of the whole system is really the only issue. I think that part of the issue is that we want to be able to allow our systems to be extended or tailored by end users as they see fit - and I see this issue as different from the idea that we want to be able to incrementally and very rapidly change small parts of the system as compared to the regeneration of the whole thing.

Regeneration of the whole system seems to suggest a legacy type approach to systems design and implementation where we build a highly structured or centralized system. One reason for going to OO based systems, it seems, would be the ability to have distributed OO systems where the objects have decentralized behaviors. As method or rules are either inherited or delegated to objects, local business units can specialize or overwrite these rules or method as they see necessary within their preview. Of course one would need to provide the necessary mechanisms for granting and checking permissions and logging actual changes. But that issue is secondary to the point that flexibility and extensibility are required in today's business systems which I think is a different issue than complete system regeneration.

#### A.10.2. Interview 10 Write-Up

SEMTECH's and the OMG's architecture are, in fact, built to provide plug-and-play capabilities. The OMG's architecture (i.e., the OMA) is built using a layered approach. SEMATECH's architecture (just like TI's) really represents only one layer. The architecture must provide flexibility, adaptability, and extensibility by managing the integration of changing business rules that define the application's behavior. In a layered approach, subclassing across layers will result in a situation where your application will break when higher layers are redefined. Instead, delegation or

dynamic typing must be used in order to provide flexibility, adaptability, and extensibility - some method that provides a mapping across layers. Within a layer (a horizontal slice), multiple views (a vertical slice) will exist. These multiple perspectives (vertical slices) must all be consistently and simultaneously integrated within the layer in which they exist in order to achieve interoperability.

## A.11. Interview 11

OMG/X3H7

January, 1996

F. M. - GTE

M. B. - GTE

T. D. - TI

H. K. - IBM

R. H. - IBM

### A.11.1. Interview 11 Notes

**WDB:** What do you think about the idea of a layered architecture?

**TD:** We strongly agree with the idea of a layered architecture because we think that it is necessary to encapsulate or hide some the complexities at each level from the level above. Yet our experience with trying to scale-up our manufacturing framework to the enterprise level suggest that a bottom-up approach may have as many problems as F. C. described with their a rigid top-down approach. We found that concepts existed at the enterprise level that we failed to account for at the manufacturing framework level and that specific viewpoints existed at the enterprise level that we had already hardwired into the manufacturing framework precluding the representation of other viewpoints.

**FM:** Even when you decompose the problem top-down in order to provide level by level encapsulation, you often discover that certain concepts that exist at lower levels of the decomposition also require representation at the higher levels. In a telephone company, the concept of a "dial tone" really exists at the switching software/hardware level. Yet, in every telephone company that exist today, the concept of a "dial tone" is used at every level of the organization including the highest business levels. What I am trying to say is that some concepts exist at multiple levels or that some concepts can not be encapsulated within a level because they leak across levels. (leakage's across levels)

**HK:** First of all, I like the idea of the use of abstraction - which is the reason that you would have a layered architecture. Since each layer is going to represent a different domain (business domain, technology domain, etc.), I think that you will also need to have multiple layers of detail within each domain layer. Secondly, I think that when one talks about mapping concepts at one layer of the architecture to another layer of the architecture that one should realize that one is really talking about the idea of mapping concepts between domains.

In mapping concepts between domains, there are a number of possible cases that arise that in general include 1) a concept exists in one domain that does not exist in the other domain, 2) the concept exist in both domains, and 3) the concepts exists in one domain and only partially in the other domain. Of course if a concept exist in one domain but not the other, then there is no mapping that one can make between domains. If the concept exist in both domains, then the mapping should exist. If the concept exists in one domain but only partially in the other domain, then a partial mapping should exist.

Now, when mapping between domains, one is always constrained by the concepts that exist within each domain. As a example, consider the idea of a variable in a problem domain, a programming language domain, and the hardware domain. In the problem domain, the variable will represent an instance of a specific domain (or type) like a customer number. In the programming language, the variable might be mapped onto the concept of a numeric, alpha-numeric, or a string and at the machine level the concept of a bit pattern, byte, or word.

In building mappings across the domains, designers must be aware of the specific requirements and limitations within each domain. In the customer number example, the mapping between the programming language and the machine level will be constrained by the word size of the processor - customer numbers (if carried as integers) cannot be larger than the machine's word size will allow - which means that the machine's word size does, in fact, leak across the boundary between the two layers or domains (word size cannot be encapsulated). Likewise, any business requirements that constrains the definition of a valid customer number or the operations on or with a customer number may have to be enforced by writing addition programming language statements.

In other words, the domain specific concept of a customer number defines not only a data type but also all the valid operations on or with a customer number (the idea of a abstract data type or an object if you like). Likewise, the domain specific concept of a integer defines not only a data type but also the valid operations on or with an integer. In mapping between the two domains, both the data type and the valid operations on these types must be mapped.

Now, one way to specify the valid operations is to specify the pre-conditions, post-conditions, and the operation's invariants (or what aspect of the concept remains unchanged by the operation) for each valid operation. In order to map the pre-condition, post-condition, and operation's invariant for each valid operation across domains - say from a business domain to a programming language domain, a designer must utilize the concepts within the programming language domain to build a composition (the abstract data type or object) that provides the set of emergent properties that meet the business level pre-conditions, post-conditions, and operation invariants. In building this composition, the business domain pre-conditions, post-condition, and operations invariants will represent constraints that the programming language domain must respect - which means that concepts within the business domain - just like concepts at the machine level - migrate across domains as constraints. In other words, migration goes both ways across domains - and, in some cases, can span multiple domain mappings - which means that in those cases the concept cannot be encapsulated.

Transactions represent the classic example of this mapping issue. A business transaction, say an accounting transaction, is specified within the business domain by specifying the accounting transaction's operation invariant - which at a minimum is that BOTH the debit and credit side of the transaction must be recorded simultaneously. However, in the machine level domain, the simultaneous writing of BOTH the debit and credit to two different files is virtually impossible. So to provide a mechanism for enforcing the business domain invariant, software systems - the layer between the business domain and the machine domain - must have specific additional mechanisms built into them to support the idea of a transaction - which, of course, is why database systems have features like commit, rollback, transaction logging, recovery, etc.

Now, there are a couple of important points that one should understand about mappings and invariants. First, since invariants are defined within one domain and mapped onto another domain, these mapping may not be one-to-one mappings - which means that any specific mapping could result in a situation where multiple inheritance or multiple delegation is encountered. Second, since invariants are defined in one domain, enforcement of the invariant must be viewed from that domain - which means that at a different level of abstraction, or within a different domain, the invariant may be violated.

**WDB:** In your example, you used the idea of a business invariant, specifically an accounting invariant. Couldn't other groups within the business domain, like marketing or engineering, have rules or views that created their own invariants?

**HK:** Yes, absolutely. In fact, each viewpoint in a domain can have its own unique set of invariants. Which means that a systems designer must really integrate

both a horizontal and a vertical set of invariants into a consistent whole. However, the invariants representing different viewpoints can, in fact, not be integrated into a consistent whole. When a designer is faced with integrating different viewpoints, each with its own invariant, specified by some predicate, then the integration of all of the viewpoints requires the conjunction of the properties of each of the different viewpoints. If the conjunction is false, then the two viewpoints cannot exist simultaneously. In such a case, management must provide a partial ordering of which viewpoint takes priority over the other for specific cases.

**MB:** I would like to comment on the issue of mapping between domains. I strongly agree with H. K.'s statement that when we map across domains we are really developing mappings between two concepts. In order to develop these mappings, one must understand the concepts at both a syntactic and semantic level. In fact, multiple semantic levels. As an example, consider translating a document in English to French. To achieve an accurate translation, one is really interested in mapping the concepts in the English version of the document into concepts in the French version of the document as compared to a word-by-word substitution across versions.

To achieve an accurate concept to concept mapping, each word - a first semantic level, each sentence - a second semantic level, each paragraph - a third semantic level, and the whole document - a fourth semantic level must be utilized to help establish the semantic invariant for each lower level.

In terms of a bottom-up vs. a top-down approach, I would agree that neither approach will work by itself. First of all, a purely federated approach will not work because of the semantic impedance introduced by both missing and partial concept representations across domains. Even with complete concept representations, the number of mappings between domains under a purely federated approach increases as  $N(N-1)/2$  where  $N$  is the number of concepts. This means that the number of mappings increases as  $N^2$  which renders a purely federated approach impractical for any size mapping. Likewise, a purely top-down approach suffers from over specification which introduces rigidity and hence inflexibility in addition to the practical limitations of specifying tens of thousands of items within most major organizations.

Instead, I think a hybrid approach is needed where one uses a top-down approach to define the overall structure while allowing for a bottom-up or local instantiation of the structure. By defining invariants at a sufficiently high level of abstraction, a top-down approach should provide enough structure to overcome semantic impedance without introducing rigidity. Even something as simple as defining a single standard set of concepts using a top-down approach reduces the mapping problem to  $N + 1$  mapping.

WDB: How close do you think that the OMG's architecture is to this hybrid type architecture or what would need to be change about it?

RH: Maybe I should answer that question, because part of my job has been to develop a formal evaluation of the OMG's architecture for IBM. Because the OMG is developing a standard set of interfaces, one might think that they are moving towards a  $N + 1$  mapping idea as opposed to  $N ( N + 1 ) / 2$  mapping. However, our formal analysis of the OMG's architecture reveals that IDL (their standard interface definition language) represents, at best, a syntactic standard for interchange - and an incomplete syntactic standard at that. They have not addressed the semantic issue at all. Therefore, we feel that IDL is not a sufficient standard for developing the required mapping for interoperation across domains.

#### A.11.2. Interview 11 Write-Up

Viewed as a mapping across layers in a layered architecture, both the top-down and the bottom-up approaches suffer from problems. The bottom-up approach fails to identify higher level concepts that are important as we move up layers of abstraction. Because of this failure, higher level concepts are not open for binding when the mapping between layers takes place. The top-down approach often fails because designers specify one binding for all cases precluding flexibility - the difference here is one of subtyping versus subclassing in the OO world.

The ability to map between views or domains requires that one establish an operation invariant - a statement of what variables are bound and will not change over a specified time. Within a layer, each view can have it own invariant or set of invariants. Before one can map across layers, these multiple perspectives must all be consistently and simultaneously integrated within the layer in which they exist. If viewpoints are not consistent, partial ordering or specification of how to implement the



viewpoints will be required. In mapping across layers, an invariant at one layer of abstraction may be temporarily violated during the operation execution but restored before the operation is completed.

A mapping between layers is really a mapping between concepts. In order to accurately map one concept onto another concept, both a semantic and syntactic understanding of the concept is required - or stated in terms of invariants - you need to establish both a semantic and syntactic invariant to successfully map concepts. IDL interfaces are not sufficient achieving interoperability because they do not have a mechanism for expressing semantic requirements.

## A.12. Interview 12

OMG/X3H7

May, 1996

H. K. - IBM

J. M. - MCI

F. C. - EDS

G. H. - SEMATECH

### A.12.1. Interview 12 Notes

**WDB:** I would like to get some more comments on what would be required to have a complete plug-and-play architecture. Specifically, I would like to review what we have talked about already and see if there are any new ideas or comments that anyone would like to make.

**HK:** In general, I think that what you have presented is fine. However, I have some questions regarding extensibility and flexibility. In the object model, lower levels of abstraction are allowed to specialize a concept that already exists at a higher level of abstraction. The lower levels are not allowed to introduce new concepts that are not present at a higher level. Which suggests that the object model can only provide flexibility with the predefined set of concepts that already exist. To extend the object model to provide anything else would require a redesign of the classes with the model. Therefore, flexibility is a concept that should be defined as adaptability within the scope of the invariants and extensibility should be defined as adaptability across the scope of the invariants.

**JM:** The idea that components must establish invariants a long common dimensions to achieve interoperability is very interesting. In the telecommunication industry, we have a problem called feature interaction that results when two features interfere with each other operation. As an example, consider what happens when you have both call waiting and call forwarding on your telephone line.

Here is the problem:

The call waiting feature is programmed to look at your phone line and determine if it is in use. The programming uses a simple rule that says if the line is in use place the call in waiting.

Likewise, the call forward feature is programmed to take any incoming calls and forward them. The programming uses a simple rule that states that if you receive a call for a forwarded number, forward that call to the specified number.

Now, consider that you are making a outgoing call and someone tries to call you, should that call be forwarded, placed in waiting?

Lets look at the possible outcomes.

- 1) the call forwarding feature is always executed first when activated. The inbound call is forwarded.
- 2) the call waiting feature is always executed first. In this case, the line is busy so the call is placed in waiting.

Now consider that situation where you line is free. Let look at the possible outcomes.

- 1) the call forwarding feature is always executed first when activated. The inbound call is forwarded.
- 2) the call waiting feature is always executed first. In this case, the line is free so the call is forwarded.

As you can see, the behavior or the systems depends on the interaction of the two features based upon the partial ordering that are selected for their application. It seems that this idea of dimensions could be expanded and used to look at the idea of features and feature interactions because I think that we will have the same type of feature interaction problems when we plug together software components.

FC: I like the inclusion of multiple view within each layer. I think that this represent a major contribution to the architecture. The way the rules flow into each of the objects with the local resolution of all views really makes the whole thing very dynamic. I think that the major issues here is how to identify inconsistencies and then resolve them.

**GH:** I am not sure that I understand what your architecture means in terms of building frameworks. Based upon my understand of SEMATECH's framework, some of the requirement of your architecture are inconsistent with what we have been doing.

**HK:** I would like to add one or two things to you list of requirements. When an operations is specified, you must also specify its pre-conditions, post-conditions, premissions, prohibition, and obligations besides it operation invariants.

#### A.12.2. Interview 12 Write-Up

The idea of extensibility shuold be viewed as both a within scope and across scope concept. The object model only provides within scope extensions. The idea of defining dimensions could be useful in defining software features and looking at the problem of feature interactions. The specification of business operations - the integration of each view's invariants or rules--must include not only the specification of the operations invariants but also the pre-conditions, post-conditions, premissions, prohibition, and obligations for that operation.

## A.13. Interview 13

SEMATECH

October, 1996

S. H. - SEMATECH

A. M. - IBM

M. K. - IBM

### A.13.1.1. Interview 13 Notes

**WDB:** Could you give me a quick overview of the structure of SEMATECH's CIM Framework?

**SH:** The CIM Framework is based upon a partitioning of a MES (Manufacturing Execution System) into seven functional groups or components. For each of these groups, the CIM Framework specifies the group's purpose, the services provided by the group, and the collaborations required to provide these services by defining a set of external interfaces for each group. From a software development perspective, the CIM Framework serves as a guideline for the development of plug compatible applications. The approach taken with the framework specification is to define an abstract model for semiconductor manufacturing that consists of a set of generic abstractions, services, and protocols representing the minimum required for supporting the development of compatible plug compatible applications in a multi-supplier scenario. The specification is not intended to define all of the applications that can or should be supplied, nor is it intended to specify any specific physical realization of the CIM Framework. SEMATECH believes that the CIM Framework can support many diverse implementations, including centralized and distributed architectures, using numerous system technologies. In a real CIM Framework compliant implementation, applications developers can choose to implement and specialize the framework's abstract groups in any appropriate manner as long as the specified external interfaces remain the same.

**WDB:** Now that the project is complete, what can you tell me about the ideas behind the framework and how well they actually worked?

**MK:** In general, I would have to say that the project showed that the framework specification can be implemented. However, we discovered a number of issues

with the framework that are causing us a lot of concern. First of all, you must realize that we took only one off-the-shelf framework component and integrated this component into a existing MES.

To get the new component to interoperate with the existing MES required us to reorder the existing MES system's flow-of-control. Building a wrapper/interface around the existing MES components was not sufficient. We found that we had to rework an number of the internals of the existing components to support the required interfaces. When we started reworking the existing components, we found that as we changed the internal structure of one component we often introduced a new requirement on another or second existing component--which required us to change that second component to meet the new requirement--which cause a new requirement on other existing components--etc. To simultaneously solve all of these problems, we had to use a concurrent design approach where all of the component's designs could be determined at the same point in time.

AM: In reality we had two choices, 1) we could have either modified the new component to match our old MES systems--which would not have given us any new functionality, and 2) we could have modified the rest of the MES system to fit our new component. We chose the second choice for the obvious reasons.

WDB: What if you wanted to add some functionality to a MES system that was not CIM Framework compliant? You would run into the same set of problems unless the extension was completely contained within one of the already defined components.

AM: Yes, I think that you are correct. Any extension that crosses a component boundary would require a new concurrent solution.

WDB: The CIM Framework represents one layer in the layered architecture. What would happen if something in the layer below the CIM Framework, like a machine in the fab, were to be completely redesigned?

AM: The internals of the components would have to be redesigned but the external functionality would stay the same--meaning that the external interfaces would also stay the same.

WDB: I think that would be true only if the new capabilities of the machine were isolated within the functionality of the component's abstract description. Looking at what happened between components within a single layer (the MES layer described by the CIM Framework) when the new functionality extended across the component boundary, I think that any functionality that extends across layers of abstraction would result in the same set of problems. Let me

try to explain what I mean by asking, Does the functionality of a machine in the fab end up being represented in more than one CIM Framework component?

AM: Yes. All the time.

WDB: So viewed from the layer of abstraction below the CIM Framework looking up at the CIM Framework layer, the abstract representation of a machine's functionality in the CIM Framework is split across the CIM Framework's components--which are decomposed based upon a functional decomposition of an MES system. Looking at the machine from the CIM Framework layer, the functionality of the machine is split into multiple viewpoints each represented by a functionality decomposed component. Which means that the machine's functionality from the CIM Framework perspective must simultaneously meet all of the requirements of the different viewpoints. If you extend the underlying functionality of the machine and that functionality flows into more than one of the CIM Frameworks viewpoints, then the functionality flows across components boundaries in the CIM Framework.

AM: Yes, that would be true.

#### A.13.2. Interview 13 Write-Up

To provide interoperability between components within a layer, components must be built under the same solution approach--in addition to having syntactic and semantic invariance. When extending the functionality of the layer, some extensions will not be able to be implemented that are completely encapsulated with a single component's boundary. In that case, reciprocal or transitive dependencies will be created as other components within the layer are changed to provide meet the new requirements--"how" you implement a component will, in part, determine "what" functionality it requires from other components. Since the emergent properties of the system result from the interrelationships between components, "how" you implement the invariants of those relationships determines the emergent properties of the system.